CycleIK: Neuro-inspired Inverse Kinematics

Jan-Gerrit Habekost(✉), Erik Strahl, Philipp Allgeuer, Matthias Kerzel, and Stefan Wermter

Knowledge Technology, Department of Informatics, University of Hamburg, Hamburg, Germany
{jan-gerrit.habekost,stefan.wermter}@uni-hamburg.de

Abstract. The paper introduces CycleIK, a neuro-robotic approach that wraps two novel neuro-inspired methods for the inverse kinematics (IK) task—a Generative Adversarial Network (GAN), and a Multi-Layer Perceptron architecture. These methods can be used in a standalone fashion, but we also show how embedding these into a hybrid neuro-genetic IK pipeline allows for further optimization via sequential least-squares programming (SLSQP) or a genetic algorithm (GA). The models are trained and tested on dense datasets that were collected from random robot configurations of the new Neuro-Inspired COLlaborator (NICOL), a semi-humanoid robot with two redundant 8-DoF manipulators. We utilize the weighted multi-objective function from the state-of-the-art BioIK method to support the training process and our hybrid neuro-genetic architecture. We show that the neural models can compete with state-of-the-art IK approaches, which allows for deployment directly to robotic hardware. Additionally, it is shown that the incorporation of the genetic algorithm improves the precision while simultaneously reducing the overall runtime.

Keywords: Neuro-inspired Inverse Kinematics · Humanoid Robots · Genetic Algorithms · Generative Adversarial Networks

1 Introduction

The inverse kinematics task searches for suitable joint configurations for a kinematic chain in order to achieve a specified end-effector Cartesian pose. Recent collaborative and humanoid robot designs often rely on redundant manipulators with more than six degrees of freedom (DoF). The complexity of the inverse kinematics task is therefore increased, as the problem is then under-determined and a set of redundant solutions for a single pose can be found, referred to as the nullspace. The Python-based genetic IK solver Gaikpy [10], originally developed for the child-sized NICO robot [11] with 6-DoF arms, requires a long runtime in order to deal with the 8-DoF manipulators of the recently developed Neuro-Inspired COLlaborator [9], pictured in the top-left image in Fig. 1.

L. Iliadis et al. (Eds.): ICANN 2023, LNCS 14254, pp. 457–470, 2023.
https://doi.org/10.1007/978-3-031-44207-0_38

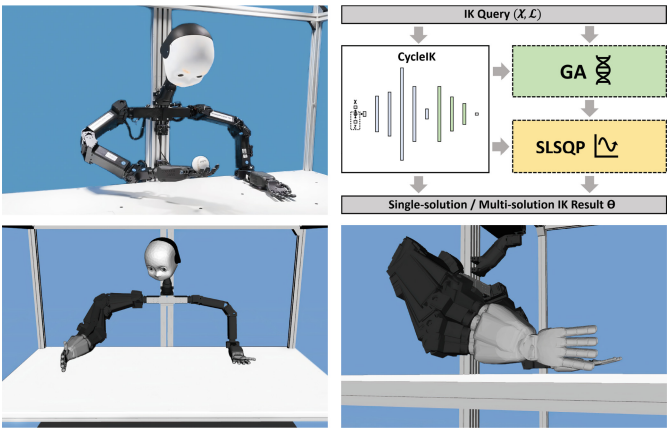


Fig. 1. CycleIK deployed to physical NICOL hardware (top-left). CycleIK hybrid neuro-genetic inverse kinematics pipeline (top-right). Visualization of the nullspace manifold from the CycleIK Generative Adversarial Network (bottom).

Traditionally, Jacobian-based methods are utilized for the IK task, such as KDL [17] and Trac-IK [5] which are popular plug-ins in the MoveIt [7] framework and can currently be seen as the industry standard. Both analytical solvers require a high runtime when deployed to NICOL, and have a higher error than Gaikpy [9]. We initially configured BioIK [18] to be the default solver, a popular state-of-the-art genetic approach, which was also deployed via Moveit. MoveIt, however, does not return a solution for an IK query, when the error is higher than the internal threshold, leaving the control cycle of the robot with no action.

Neural inverse kinematics is a field that unites a wide range of neuro-robotic applications that control the configuration space of a robotic system. The inverse kinematics task is fundamentally embodied in every action-generating neural architecture that takes data from Cartesian space as input. Explicit neural approaches to the task, however, rarely show results with high precision and are distributed over the different application domains of inverse kinematics ranging from robotics to character animation.

Two neural architectures, an auto-regressive Multi-Layer Perceptron (MLP) and a normalizing flow-based Generative Adversarial Network, are proposed in this work. The models solve the inverse kinematics task for a given pose in the reachability space of NICOL and can be deployed directly to robotic hardware, or alternatively be optimized with Gaikpy. The MLP returns exactly one solution for the IK task, while the GAN allows for the exploration of the nullspace manifold. The method is inspired by CycleGAN [20], which trains a dual-GAN architecture in an unsupervised fashion, to transform between two image domains. The positional and rotational errors are measured in Cartesian space by calculating the forward kinematics (FK) for a set of IK solutions that are inferred from the neural models. The FK function calculates the end-effector pose from a

given robot configuration and has a short runtime of below $1ms$. Consequently, a second generator as in the original dual-GAN setup of CycleGAN, that approximates the FK function to transform from configuration to Cartesian space, is not necessarily needed for this application.

2 Related Work

The most similar normalizing flow-based approaches to ours are IKFlow from Ames et al. [3] and the work of Kim and Perez [12]. IKFlow is a recent and promising neural IK approach. The authors propose a conditional normalizing flow network for the inverse kinematics task, a form of Invertible Neural Network (INN) [4], introduced by Ardizzone et al. for invertible problems. Samples from a simple normal distribution are transformed into valid solution manifolds in the configuration domain through coupling layers that consist of multiple simple invertible functions. The solution manifold can optionally be further optimized with Trac-IK[5].

The approach of Kim and Perez [12] has a very similar architecture to IKFlow. Compared to IKFlow, which calculates the error with analytical forward kinematics, Kim and Perez use a second neural network to approximate the FK function in an autoencoder architecture. The approach of Kim and Perez has a comparably high error in the centimeter range and requires further optimization with the Jacobian, while IKFlow reaches a millimeter range of error.

Lembono et al. [14] present an ensemble architecture in which multiple GAN generators learn to sample from disjunct patches of the configuration space. A single forward kinematics discriminator is used that also checks for further constraints, e.g. minimal displacement of the arms. A more detailed investigation of GANs in the context of IK is given by Ren and Ben-Tzvi [16]. The paper modifies four different types of GAN architectures to solve the inverse kinematics problem. The discriminator produces binary output, while most GAN designs perform regression and calculate the continuous error to the target pose.

Bensadoun et al. [6] introduce a Gaussian Mixture Model (GMM) ensemble to calculate multiple solutions for the IK problem. A GMM is created for every joint in the kinematic chain. A hypernet parameterizes the GMMs conditioned to the target pose. Volinski et al. [19] utilize Spiking Neural Networks (SNN) to solve the inverse kinematics problem. The approach trains three different variations of simple SNN architectures. ProtoRes [15] was introduced by Oreshkin et al. to reconstruct natural body poses from sparse user input for animation tasks. The framework consists of a pose encoder that creates a latent embedding from the user input and then solves the IK task with a pose decoder.

3 Method

We propose CycleIK, a neuro-inspired inverse kinematics solver that makes use of the cyclic dependency between the transformation from configuration to Cartesian space and its inverse. An overview of the architecture is given in Fig. 2. The framework enables either training a single-solution auto-regressive Multi-Layer Perceptron or a normalizing flow-based GAN architecture that allows the parallel inference of multiple redundant solutions within 1 ms. Furthermore, the approach can be utilized as a neuro-kinematic toolbox. The default networks can be substituted by any end-to-end or multi-stage robotic control architecture that predicts joint angles and provides a Cartesian pose as a label. CycleIK is implemented in PyTorch, to be as openly available as possible. Most IK solvers are implemented in C++ and generally rely on iterative numerical methods for the optimization process, often leading to a higher runtime compared to the inference of a neural network.

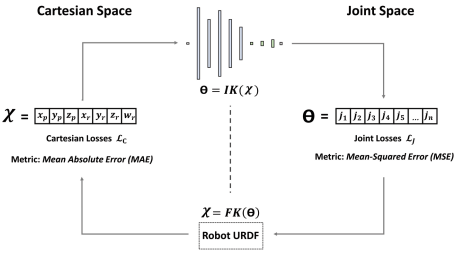


Fig. 2. CycleIK neuro-inspired training and architecture overview. A batch of Cartesian poses $\mathcal{X}$ is inferred by the network to predict a set of valid robot configurations Θ under constraints $\mathcal{L}$.

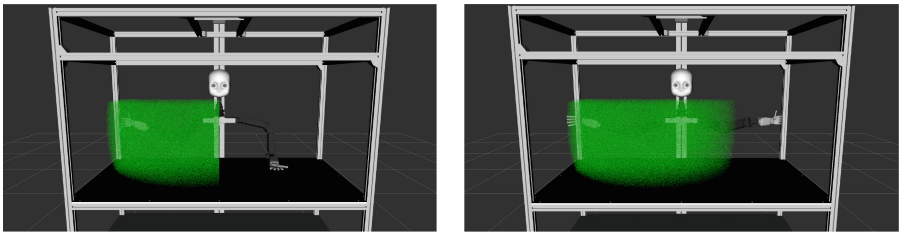
CycleIK treats the joint space as a semi-hidden domain, and calculates positional and rotational losses only in Cartesian space, by inferring a full cycle back to Cartesian space, as shown in the following equations (Eq. 1 and 2):

$$\hat{\mathcal{X}} = FK(IK(\mathcal{X})) \tag{1}$$

$$e_{IK} = \|\hat{\mathcal{X}} - \mathcal{X}\| \tag{2}$$

where $\mathcal{X}$ is a batch of an arbitrary natural number of target poses, and e_{IK} is the linear Cartesian error. While learning a one-to-one mapping between data from Cartesian space and corresponding joint angles θ can work for lower-DoF manipulators [10], the approach shows a high error for redundant manipulators like on the NICOL robot [9], as these manipulators have a one-to-many mapping in the form of the redundant nullspace manifold Θ. Thus, we minimize the linear Cartesian error e_{IK} instead, which in our experience learns and generalizes more smoothly.

Similar to neuro-inspired multi-solution solvers like IKFlow and CycleIK, genetic algorithms produce multiple solutions for an IK query, and have shown good results for the IK task [1,10,18]. The most popular genetic IK approach is BioIK [18], which is available in both MoveIt and Unity. The method supports genetic algorithms by hybridization with particle swarm optimization (PSO). The architecture allows generic IK queries through a weighted partial cost function $\phi(\Theta, \mathcal{L})$ that is applied to the set of IK solutions Θ under the constraints $\mathcal{L}$. The constraints can be reformulated at every IK query, so complex dynamic tasks



$Small_{1000}$ dataset front view $Full_{1400}$ dataset front view

Fig. 3. Visualization of NICOL's right arm workspace, with the $Small_{1000}$ dataset on the left and the $Full_{1400}$ dataset on the right.

such as collision avoidance in motion planning can be performed. Different goal types can be set for either the links or joints of the robot. We adapt the weighted partial cost function from BioIK for both of our models. CycleIK's single-solution model optionally makes use of a set of weighted constraints $\mathcal{L} = \mathcal{L}_C \cup \mathcal{L}_J$, that consists of specified goals, either in Cartesian or joint space. The constraints are applied by the multi-objective function in every training step. Both, the single-solution MLP as well as the multi-solution GAN, can optionally be further optimized by the Python-based genetic IK Gaikpy [10] or non-linear sequential least squares quadratic programming [13], where again a partial weighted cost function can be used to select the optimal solution. An overview of the neuro-genetic IK pipeline is given in the top-right image of Fig. 1.

3.1 Dataset

Three datasets were collected from NICOL's workspace: $Small_{1000}$, $Full_{1000}$ and $Full_{1400}$. Uniform random collision-free robot configurations were sampled. The $Small_{1000}$ and $Full_{1400}$ dataset are shown in Fig. 3. The $Small_{1000}$ dataset contains 1,000,000 samples and is limited to the right side of the tabletop, which is located 80cm above the ground. The $Full_{1000}$ and $Full_{1400}$ datasets with $1,000,000$ and $1,400,000$ poses are sampled from the whole workspace of the right arm over the tabletop. We built test sets with 10% size and validation sets with 1% size for each of the training datasets. In all datasets, a 20cm safety margin was included at the back of the workspace on the x-axis, as well as a 10cm safety margin on the y-axis on the right-hand side of the robot workspace. All properties of the datasets can be seen in Table 1. A convex hull was generated around the data points to approximate the Cartesian volume of each dataset.

Table 1. Overview of the training datasets and the corresponding 10% test sets ($Small_{100}$, $Full_{100}$ and $Full_{140}$) and 1% validation sets ($Small_{10}$, $Full_{10}$ and $Full_{14}$).

Dataset	Workspace [x, y, z] (m)	Samples	Volume (cm^3)	Sample Density ($samples\ per\ cm^3$)
$Small_{1000}$	$\begin{bmatrix} 0.2 & -0.9 & 0.8 \\ 0.85 & 0.0 & 1.4 \end{bmatrix}$	10^6	$295.56 \cdot 10^3$	3.383
$Small_{100}$		10^5	$293.34 \cdot 10^3$	0.341
$Small_{10}$		10^4	$287.11 \cdot 10^3$	0.035
$Full_{1000}$	$\begin{bmatrix} 0.2 & -0.9 & 0.8 \\ 0.85 & 0.48 & 1.4 \end{bmatrix}$	10^6	$420.11 \cdot 10^3$	2.38
$Full_{100}$		10^5	$415.13 \cdot 10^3$	0.241
$Full_{10}$		10^4	$401.75 \cdot 10^3$	0.025
$Full_{1400}$	$\begin{bmatrix} 0.2 & -0.9 & 0.8 \\ 0.85 & 0.48 & 1.4 \end{bmatrix}$	$1.4 \cdot 10^6$	$420.43 \cdot 10^3$	3.33
$Full_{140}$		$1.4 \cdot 10^5$	$416.09 \cdot 10^3$	0.336
$Full_{14}$		$1.4 \cdot 10^4$	$405.07 \cdot 10^3$	0.035

3.2 Architecture

The basic network architecture is very similar for both models. The pose is encoded as a 7-dimensional vector, i.e. the 3-dimensional position $[x_p, y_p, z_p]^T$ concatenated with the rotation represented as a 4-dimensional unit quaternion $[x_r, y_r, z_r, w_r]^T$, as shown in Fig. 2. The output of the network has the same dimension as the robot DoF, so every field of the output vector corresponds to a motor position in the kinematic chain. The GAN additionally concatenates the pose with a second input, a random uniform noise vector that is utilized to sample from the nullspace manifold. The models utilize two different activation functions. While Gaussian-Error Linear Units [8] (GELU) are generally used for all the layers, the Tanh activation is applied to the last one to three layers of the network, as this highly improves the results. The data is normalized to lie in the interval $[-1, 1]$, which is equivalent to the limits of the network input and output. Thus, the method cannot push the joint angles through their joint limits, which is a shortcoming of a lot of Jacobian-based IK solvers. Visualizations of the two network architectures for the NICOL robot can be found in Fig. 4.

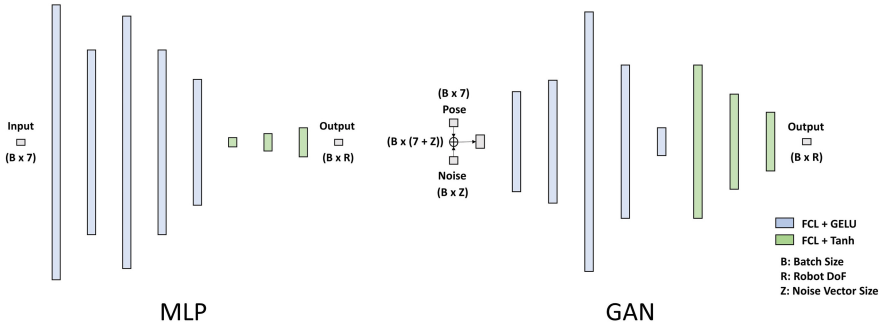


Fig. 4. Neural architectures optimized for the $Small_{1000}$ dataset, Multi-Layer Perceptron (left) and Generative Adversarial Network (right).

3.3 Training

In every training step, a batch of poses $\mathcal{X}$ is inferred by the network. For the single-solution network, the training step is straightforward—after inference, forward kinematics are applied to the batch of solutions Θ to determine the reached poses $\hat{\mathcal{X}}$ and then apply the multi-objective loss function, as in Eq. 3:

$$loss_{\mathcal{L}} = \phi(\Theta, \hat{\mathcal{X}}, \mathcal{L}) \tag{3}$$

Here, $\mathcal{L}$ holds at least the positional and rotational error. For the NICOL robot, we applied a zero-controller goal that minimizes the displacement of the motor position from the zero position of the selected subset of redundant joints in the kinematic chain. Our preliminary experiments showed the best performance by using the mean absolute error for Cartesian space losses and mean squared error for the joint space losses, as the error increased for all other evaluated error terms. The learning rate is decreased linearly at the end of each epoch.

The training process for the multi-solution GAN extends the training process of the MLP. After calculation of the positional and rotational loss for a batch of Cartesian samples from the training set, one of the poses is randomly chosen from the batch. A tensor of the same size as the training batch is created and filled with the chosen pose. Random uniform noise $\mathcal{Z}$ of the required batch size and noise vector size is then generated and used for the forward pass. The training aims to maximize the variance in the solution batch Θ. The normalizing flow method is applied, as the network is not being forced to regress to only one solution, but instead fit the nullspace distribution Θ to the noise $\mathcal{Z}$, as in Eq. 4:

$$loss_{var} = MSE(var(\Theta) - var(\mathcal{Z})) \tag{4}$$

The method can produce multiple valid solutions for the NICOL robot with millimeter-level accuracy. One possible extension would be to combine Kullback-Leibler divergence for the loss and normally distributed noise in the input, as done by IKFlow [3] and Kim and Perez [12].

3.4 Optimization

Each of the models was optimized over 250 trials for both the Small and Full workspace. The results are shown in Table 2. For the Full workspace, we chose to optimize the models with the $Full_{1400}$ dataset. We used the Optuna framework [2] to optimize the models with a Tree-structured Parzen Estimator (TPE) for sampling, and a hyperband pruner. Four parameters were defined for the optimization process, which are the batch size, learning rate, number of layers in the network, and the number of layers with tanh activations at the end of the network. Additionally, we optimized the number of neurons in every layer. An overview of the exact network layouts can be found in Table 3, and a visualization of the network structures for the Small workspace is shown in Fig. 4. For the GAN only, we also optimized the size of the input noise vector.

Table 2. Training parameters for the different network types, optimized for the $Small_{1000}$ and $Full_{1400}$ datasets.

Parameter	MLP		GAN		Parameter	Limits
	Small	Full	Small	Full	min./max.	step size
Batch Size	150	300	350	300	100 / 600	50
Learning Rate	$1.6 \cdot 10^{-4}$	10^{-4}	$2.1 \cdot 10^{-4}$	$1.9 \cdot 10^{-4}$	10^{-5} / 10^{-3}	10^{-5}
Number Layers	8	8	8	8	7 / 9	1
Number Tanh Layers	3	3	3	2	1 / 3	1
Noise Vector Size	–	–	8	10	3 / 10	1

Table 3. Network structures of the different network types optimized for the $Small_{1000}$ and $Full_{1400}$ workspace.

Model	Workspace	Neurons per Layer
MLP	Small	[3380, 2250, 3240, 2270, 1840, 30, 60, 220]
	Full	[2200, 2400, 2400, 1900, 250, 220, 30, 380]
GAN	Small	[790, 990, 3120, 1630, 300, 1660, 730, 540]
	Full	[1180, 1170, 2500, 1290, 700, 970, 440, 770]

4 Results

The application of the weighted partial cost function on the MLP network and the variance loss on the GAN created stability issues in the training process of differing severity for the two models. The MLP rarely shows stability issues during the training process, but they sometimes occur when trained for more

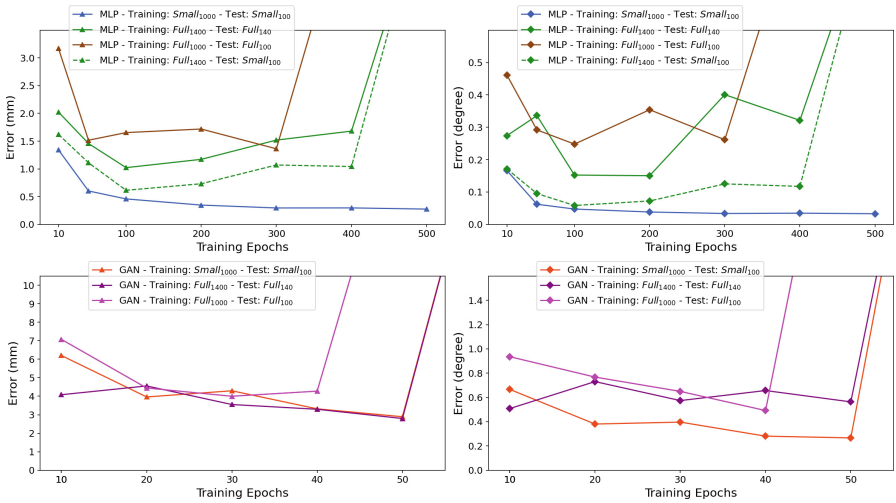


Fig. 5. Average positional and rotational error of the MLP and GAN model under training for varying numbers of epochs.

than 100 epochs, and can be dealt with using gradient clipping. The GAN suffers more severe stability issues, and could not be trained for more than 50 epochs in our experiments. We hypothesize it is due to the competition of maximizing the nullspace manifold variance while maintaining precise IK regression. Gradient clipping cannot be applied as easily in the case of the GAN because it prevents learning proper minimization of the variance loss.

4.1 Optimal Number of Epochs

To determine the optimal number of epochs for the training process, we trained both presented models for each of the three datasets that were generated, so that six models in total were evaluated under different epoch configurations. A standalone training was performed for every individual model and number of epochs. To handle the stability issues of the models, we gave every evaluated epoch configuration a number of restarts in case stability issues occur. Each choice of maximum epochs was allowed two restarts for the MLP and nine for the GAN. If exploding gradients occurred in every observed training, the combination was considered to have failed. The results of our experiments are shown in Fig. 5. We calculated the positional and rotational error for the MLP by first taking the average over the three corresponding axes of the 6-DoF pose error, and then averaging the results for the whole 10% test sets. For the multi-solution GAN, we first calculated the average error over single batches of nullspace solutions, before taking the mean over the whole test set. We take the success definition for the inverse kinematics task from Kerzel et al. [10], which allows $10mm$ positional and 20-degree rotational error.

GAN. It can be seen that the training process of the $Small_{1000}$ dataset had the lowest error for most of the epoch configurations when compared to the training of the $Full_{1000}$ and $Full_{1400}$ datasets. For a higher number of epochs, the positional error of the GAN models behaves similarly for the $Small_{1000}$ and $Full_{1400}$ datasets. Instabilities occur for short training and with regard to the rotational error. The training of the $Full_{1000}$ GAN model already starts to fail when training for more than 40 epochs, while the rotational error can compete with the loss of the $Full_{1400}$ model for a lot of configurations.

MLP. The results of the single-solution MLP for the training on the *Small* and *Full* workspace differ more strongly than for the GAN. Different from the GAN, where the exact same loss is used for both workspaces, the zero-controller goal that we set for the training of the MLP has to be tuned for a specific workspace and therefore differs. The additional joint space goal can therefore explain the differences in training behavior to some degree. The training with the $Full_{1000}$ dataset can also for the MLP compete with the $Full_{1400}$ training for some epoch configurations. Overall, the best model for the $Full_{1400}$ dataset exceeds the best model for the $Full_{1000}$ dataset.

The training with the $Small_{1000}$ dataset proceeded the smoothest, and we did not experience any stability issues. In contrast to the GANs, where the smallest positional error is achieved after 50 epochs for both workspaces, with slightly below $3mm$ average error, the MLP models differ in the ideal training length as well as in the smallest error. While the lowest positional error for the Full workspace is achieved after 100 epochs, the best results for the Small workspace are found after 300 training epochs. The best results for the $Full_{1000}$ dataset are also achieved with 300 training epochs, but cannot compete with the best model of the $Full_{1400}$ dataset. We evaluated the performance of the different models for the $Full_{1400}$ dataset on the $Small_{100}$ test set to make the $Small_{1000}$ and $Full_{1400}$ models directly comparable. It can be seen from the green dotted line in Fig. 5 that the $Full_{1400}$ models perform very similarly when evaluated on the same test data as the $Small_{1000}$ models. Positional and rotational errors only show small differences until 100 training epochs are exceeded.

Overall, we focus more on the positional error rather than the rotational error, as the rotational error is far below our success limit of 20 °C in almost all cases. Especially the models that were only trained for 10 or 20 epochs can show up to 5 mm average positional error, and therefore a lot of solutions around the upper bound of the error exceed the limit of 1 cm.

4.2 Precision Analysis

From the previous experiment, the best-performing models were selected and evaluated for the $Small_{100}$ and $Full_{140}$ test sets. We seeded 50% of Gaikpy's initial population with solutions from the neural models in a follow-up experiment and filled the other half of the population with uniform random robot configurations within the joint limits. As a baseline, errors for standalone BioIK and Gaikpy were evaluated. The results of our IK experiments on the Small

and Full workspace can be seen in Table 4. The framework offers SLSQP for further optimization but we did not use it in the experiments, as the solutions are already precise enough to be deployed to the physical hardware.

The performance of BioIK on the test sets was measured via MoveIt. Since MoveIt reports an exception for solutions whose error lies over a specified threshold, as this threshold cannot be influenced, no solution can be evaluated for the failed requests. This behavior is different from all other methods that are utilized in this work, as they always report at least some kind of solution. For the failed MoveIt requests, we calculated the distance between the initial rest end-effector pose and the target pose, which increases the average positional and rotational error in comparison to the other methods. The positional error lies around 0.02 to $0.05mm$ for the successful requests and would therefore outperform the presented Python-based methods.

For the GAN, 500 solutions for the same pose were generated, and the average error for every pose was calculated before the mean was taken over the whole test set. For all other methods, we only analyzed the error of the best solution for every test pose. For the GAN results, the average error of the best solution for every test pose is the average minimum error reported in Table 4.

It can be seen that the GAN model performs better for the Full workspace, while the MLP performs better for the Small workspace. The average error of the GANs is between three to ten times higher than for the MLPs. However, it was possible to improve the solutions of the GANs as well as the MLPs through optimization with Gaikpy. In general, the orientation errors of the MLPs increased while the positional errors decreased. Moreover, while the average maximum error of the GANs is near the upper limit we defined for the error, which is generally good as it indicates that most solutions are within the error limit, the success rate of the GANs can only compete with BioIK and the CycleIK MLP model through the genetic optimization. Both the MLP and GAN models can be deployed directly to real hardware without further optimization, as the positional error stays far below 1 cm on average.

The standalone Gaikpy method shows a lower average positional error than BioIK and a similar to slightly lower rotational error. The divergent success definition of BioIK is the reason that its success rate of over 98% outperforms the success rate of Gaikpy by about 3–5%, while the average error of BioIK is tremendously higher. When Gaikpy is seeded with the neural models from CycleIK, the error of the solutions can be reduced by around 60% to 90% while the timeout of the genetic algorithm can be reduced by over 98%, enabling the neuro-genetic method to directly compete with BioIK regarding success rate as well as average error. The standalone Gaikpy method overcomes both neuro-only architectures as well as the Gaikpy variant that was seeded with the GAN solutions with regard to the positional error. In contrast, Gaikpy's orientation error is higher than for all CycleIK setups, which indicates that the seeding with neural solutions increases Gaikpy's performance with regard to the orientation.

Table 4. Results of different CycleIK variants and standalone Gaikpy and BioIK on the $Small_{100}$ and $Full_{140}$ test set.

Model	Work-	Position (mm)		Orientation (°)		Success	Timeout
	space	Avg.	Min./Max.	Avg.	Min./Max.	Rate (%)	(ms)
CycleIK$_{MLP}$	Small	0.295	$5.36 \cdot 10^{-4}$ / 143.56	0.033	$2.39 \cdot 10^{-4}$ / 85.73	99.48	0.242
	Full	1.022	$1.12 \cdot 10^{-3}$ / 376.39	0.152	$3.92 \cdot 10^{-4}$ / 127.41	98.49	0.243
CycleIK$_{MLP}$ w. Gaikpy	Small	0.074	$4.83 \cdot 10^{-6}$ / 245.57	0.089	$2.74 \cdot 10^{-4}$/ 93.43	99.85	19.589
	Full	0.163	$3.44 \cdot 10^{-6}$ / 271.33	0.308	$1.59 \cdot 10^{-4}$ / 128.11	99.38	19.603
CycleIK$_{GAN}$	Small	2.892	0.602 / 11.87	0.266	0.046 / 2.82	92.07	0.458
	Full	2.795	0.7 / 10.84	0.563	0.134/ 3.56	94.77	0.448
CycleIK$_{GAN}$ w. Gaikpy	Small	0.525	$6.84 \cdot 10^{-6}$ / 169.33	0.308	$4.22 \cdot 10^{-4}$ / 127.27	98.46	19.922
	Full	0.4	$9.34 \cdot 10^{-6}$ / 366.22	0.407	$7.58 \cdot 10^{-4}$ / 133.89	98.97	19.572
Gaikpy	Small	0.113	$5.16 \cdot 10^{-6}$ / 62.83	8.066	0.09 / 139.45	93.33	1022.534
	Full	0.062	$3.19 \cdot 10^{-6}$ / 100.83	5.969	0.03 / 143.06	96.06	1106.849
BioIK	Small	33.487	$1.24 \cdot 10^{-6}$ / 654.83	7.625	$1.39 \cdot 10^{-6}$ / 142.48	98.72	1
	Full	41.468	$9.93 \cdot 10^{-6}$ / 575.57	8.349	$1.54 \cdot 10^{-6}$ / 147.08	98.05	1

5 Conclusion

This work presented two novel neuro-inspired architectures for the inverse kinematics task that deliver state-of-the-art performance when compared to other bio-inspired methods. We showed that the neuro-only architectures are precise enough to be directly deployed to real-world robots. It was also shown that the solutions from the GAN, as well as the MLP architecture, can additionally be used as seeds for a genetic algorithm. The results showed that seeding the GA with the CycleIK output did not only improve the Cartesian precision of the neural solutions, but also reduced the runtime of the GA by over 98%. The weighted multi-objective function that was applied during the training of the

MLP proved to successfully support the training and made it possible to influence the kinematic behavior of the model. Finally, the importance of the presented normalizing-flow method for the IK task is underlined, as the GAN model reaches a similar precision as IKFlow and therefore has better performance than most neuro-inspired IK approaches. CycleIK will be utilized for more sophisticated experimental setups in the future, such as collision-free motion planning in human-robot interaction and multi-modal grasping.

Acknowledgements. The authors gratefully acknowledge support from the DFG (CML, MoReSpace, LeCAREbot), BMWK (SIDIMO, VERIKAS), and the European Commission (TRAIL, TERAIS).

References

1. Aguilar, O.A., Huegel, J.C.: Inverse kinematics solution for robotic manipulators using a CUDA-based parallel genetic algorithm. In: Batyrshin, I., Sidorov, G. (eds.) MICAI 2011. LNCS (LNAI), vol. 7094, pp. 490–503. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25324-9_42
2. Akiba, T., Sano, S., Yanase, T., Ohta, T., Koyama, M.: Optuna: a next-generation hyperparameter optimization framework. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, pp. 2623–2631. Association for Computing Machinery, New York, NY, USA (2019)
3. Ames, B., Morgan, J., Konidaris, G.: IKFlow: generating diverse inverse kinematics solutions. IEEE Robot. Autom. Lett. **7**(3), 7177–7184 (2022)
4. Ardizzone, L., Kruse, J., Rother, C., Kűthe, U.: Analyzing inverse problems with invertible neural networks. In: International Conference on Learning Representations (2019)
5. Beeson, P., Ames, B.: TRAC-IK: an open-source library for improved solving of generic inverse kinematics. In: 2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids), pp. 928–935 (2015)
6. Bensadoun, R., Gur, S., Blau, N., Wolf, L.: Neural inverse kinematic. In: Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., Sabato, S. (eds.) Proceedings of the 39th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 162, pp. 1787–1797. PMLR, 17–23 July 2022
7. Coleman, D., Sucan, I.A., Chitta, S., Correll, N.: Reducing the barrier to entry of complex robotic software: a moveit! case study. J. Softw. Eng. Robot. **5**(1), 3–16 (2014)
8. Hendrycks, D., Gimpel, K.: Gaussian Error Linear Units (GELUs). arXiv e-prints arXiv:1606.08415, June 2016
9. Kerzel, M., et al.: Nicol: a neuro-inspired collaborative semi-humanoid robot that bridges social interaction and reliable manipulation. arXiv e-prints arXiv:2305.08528 (2023)
10. Kerzel, M., Spisak, J., Strahl, E., Wermter, S.: Neuro-genetic visuomotor architecture for robotic grasping. In: Farkaš, I., Masulli, P., Wermter, S. (eds.) ICANN 2020. LNCS, vol. 12397, pp. 533–545. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-61616-8_43

11. Kerzel, M., et al.: NICO-neuro-inspired companion: a developmental humanoid robot platform for multimodal interaction. In: 2017 26th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN), pp. 113–120 (2017)
12. Kim, S., Perez, J.: Learning reachable manifold and inverse mapping for a redundant robot manipulator. In: 2021 IEEE International Conference on Robotics and Automation (ICRA), pp. 4731–4737 (2021)
13. Kraft, D.: A Software Package for Sequential Quadratic Programming. Deutsche Forschungs- und Versuchsanstalt für Luft- und Raumfahrt Köln: Forschungsbericht, Wiss. Berichtswesen d. DFVLR (1988)
14. Lembono, T.S., Pignat, E., Jankowski, J., Calinon, S.: Learning constrained distributions of robot configurations with generative adversarial network. IEEE Robot. Autom. Lett. **6**(2), 4233–4240 (2021)
15. Oreshkin, B.N., Bocquelet, F., Harvey, F.G., Raitt, B., Laflamme, D.: Protores: proto-residual network for pose authoring via learned inverse kinematics. In: International Conference on Learning Representations (2022)
16. Ren, H., Ben-Tzvi, P.: Learning inverse kinematics and dynamics of a robotic manipulator using generative adversarial networks. Robot. Autonom. Syst. **124**, 103386 (2020)
17. Smits, R.: KDL: kinematics and dynamics library. http://www.orocos.org/kdl
18. Starke, S., Hendrich, N., Zhang, J.: A memetic evolutionary algorithm for real-time articulated kinematic motion. In: 2017 IEEE Congress on Evolutionary Computation (CEC), pp. 2473–2479 (2017)
19. Volinski, A., Zaidel, Y., Shalumov, A., DeWolf, T., Supic, L., Ezra Tsur, E.: Data-driven artificial and spiking neural networks for inverse kinematics in neurorobotics. Patterns **3**(1), 100391 (2022)
20. Zhu, J.Y., Park, T., Isola, P., Efros, A.A.: Unpaired image-to-image translation using cycle-consistent adversarial networks. In: 2017 IEEE International Conference on Computer Vision (ICCV), pp. 2242–2251 (2017)