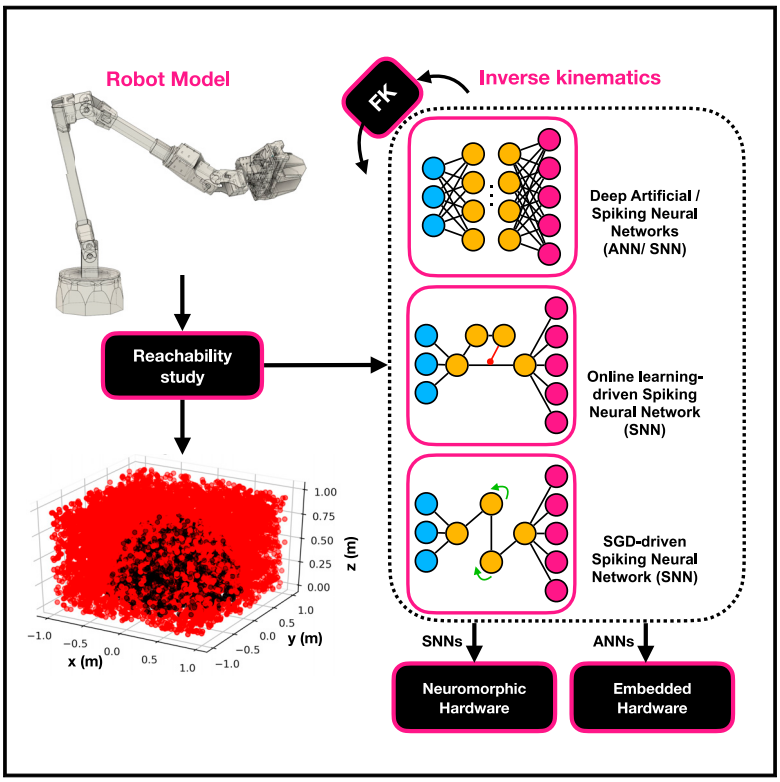
Article

Patterns

Data-driven artificial and spiking neural networks for inverse kinematics in neurorobotics

Graphical abstract



Highlights

- Inverse kinematics lies at the foundation of robot motion planning
- Artificial and spiking neural networks were used for data-driven inverse kinematics
- A data-driven approach allows robust performance in convoluted environments

Authors

Alex Volinski, Yuval Zaidel, Albert Shalumov, Travis DeWolf, Lazar Supic, Elishai Ezra Tsur

Correspondence

elishai@nbel-lab.com

In brief

We demonstrate a data-driven approach for robotic motion planning in complex environments, relying on the versatility of neural networks. We extend the discussion to brain-inspired neuronal architectures, where spiking neural networks constitute the computational framework. Spiking neural networks underlie the field of neurorobotics. They are argued to enable a robotic control that outperforms conventional paradigms in terms of robustness to perturbations and adaptation to varying conditions. A brain-inspired efficient implementation of inverse kinematics is, therefore, an important stepping stone in neurorobotics.

Volinski et al., 2022, Patterns 3, 100391
January 14, 2022
https://doi.org/10.1016/j.patter.2021.100391

CellPress

Patterns

CellPress
OPEN ACCESS

Article

Data-driven artificial and spiking neural networks for inverse kinematics in neurorobotics

Alex Volinski,[1] Yuval Zaidel,[1] Albert Shalumov,[1] Travis DeWolf,[2] Lazar Supic,[3] and Elishai Ezra Tsur[1,4,*]
[1]Neuro-Biomorphic Engineering Lab, The Open University of Israel, Ra'anana, Israel
[2]Applied Brain Research, Waterloo, Canada
[3]Accenture Labs, San Francisco, USA
[4]Lead contact
*Correspondence: elishai@nbel-lab.com
https://doi.org/10.1016/j.patter.2021.100391

THE BIGGER PICTURE Although artificial intelligence has emerged as the focal point for countless state-of-the-art developments, in many ways, its performance is nullified when compared with biological intelligence, particularly in terms of energy efficiency, robustness, versatility, and adaptivity. Therefore, neuromorphic (brain-inspired) computing has been utilized in numerous applications, particularly in robotics. Here, we uniquely address one of the most fundamental challenges in robotics: inverse kinematics in convoluted environments. Inverse kinematics is an underdetermined computational process for deriving a robot's configuration given its desired target position in space. A brain-inspired efficient implementation of inverse kinematics is, therefore, an important stepping stone in neurorobotics. Underdetermined inverse problems are also fundamental in other fields, ranging from medical imaging to hydrology and pharmacokinetics.

Proof-of-Concept: Data science output has been formulated, implemented, and tested for one domain/problem

SUMMARY

Inverse kinematics is fundamental for computational motion planning. It is used to derive an appropriate state in a robot's configuration space, given a target position in task space. In this work, we investigate the performance of fully connected and residual artificial neural networks as well as recurrent, learning-based, and deep spiking neural networks for conventional and geometrically constrained inverse kinematics. We show that while highly parameterized data-driven neural networks with tens to hundreds of thousands of parameters exhibit sub-ms inference time and sub-mm accuracy, learning-based spiking architectures can provide reasonably good results with merely a few thousand neurons. Moreover, we show that spiking neural networks can perform well in geometrically constrained task space, even when configured to an energy-conserved spiking rate, demonstrating their robustness. Neural networks were evaluated on NVIDIA's Xavier and Intel's neuromorphic Loihi chip.

INTRODUCTION

In the past few decades, multi-joint open-chain robotic arms have been utilized in a diverse set of applications, ranging from robotic surgeries[1] to space debris mitigation.[2] While the position of a robotic arm's End-Effector (EE) is often defined in Cartesian $\mathcal{R}^3$ task space, the arm is configured in $\mathcal{R}^{DoF}$ configuration space, spanned by the robot's joint angles accounting for the robot's Degrees of Freedom (DoF). While $\mathcal{R}^{DoF} \rightarrow \mathcal{R}^3$ mapping can be trivially elucidated by realizing the robot's Forward Kinematics (FK) with transformation matrices, the inverse $\mathcal{R}^3 \rightarrow \mathcal{R}^{DoF}$ mapping, termed Inverse Kinematics (IK), is computationally challenging, as the same reached point in task space can be realized with different configurations. IK is usually numerically optimized to achieve redundancy resolution using Jacobian inverse[3] or fuzzy logic.[4] While analytical descriptions of IK are generally limited to relatively simple robotic systems, numerical methods can be used to optimize solutions for intricate scenarios. While proven incredibly useful, numerical methods are subjected to complete mechanical descriptions and known environments. Several numerical approaches for IK were developed to handle geometrical and environmental constraints.[5] However, numerically handled

Table 1. ANN number of parameters and inference time

	Number of parameters	Inference (ms)
Numerical	–	16.52 ± 1.26
ResNet 4 blocks	3,141,893	6.05 ± 0.29
ResNet 2 block	183,173	2.55 ± 0.44
FC 10x128	149,765	1.34 ± 0.23
FC 8x128	116,741	1.06 ± 0.14
FC 6x128	83,717	0.87 ± 0.13
FC 6x256	331,269	0.92 ± 0.12
FC 5x128	67,205	0.79 ± 0.12
FC 5x256	265,477	0.82 ± 0.12
FC 4x128	50,693	0.67 ± 0.11
FC 4x256	199,685	0.71 ± 0.11
FC 3x128	34,181	0.58 ± 0.1
FC 3x256	133,893	0.62 ± 0.1
FC 2x128	17,669	0.49 ± 0.15

constraints should be mathematically formulated and integrated. More recently, efforts toward utilizing Artificial Neural Networks (ANNs) for data-driven IK have been explored.[6] As long as an FK is available, a neural network-powered IK provides a unified framework, supporting robotic systems with arbitrary complexity, operating in arbitrarily convoluted environments, subjected to any set of constraints. A data-driven approach alleviates the requirement for a full mechanical and mathematics descriptions.

In this work, we critically re-evaluated the utilization of ANNs for IK, with various loss functions, activations, and architectures. We further extend the discussion to IK with geometrical constraints. Robotic arms often operate in a convoluted operational space or in collaboration with a human operator. As a result, IK can be constrained to work in a sub-$\mathcal{R}^{DoF}$ configuration space. For example, a robotic arm should be prevented from invading a human's personal space or hitting obstacles. Recently, Chembuly and colleagues extended conventional IK optimizers to support collision avoidance.[7] Here, we've taken a data-driven approach, relying on the versatility of neural networks, to propose a robust framework for non-constrained and geometrically constrained IK.

Uniquely, we utilized Spiking Neural Networks (SNNs), which closely emulate the nervous system's computational properties[8] for IK. SNNs stand at the foundation of neurorobotics, an important frontier in neuromorphic computing research. It provides biologically inspired energy-efficient control of robotic systems.[9] We evaluated SNNs for IK with three different approaches: (1) converting ANNs to SNNs using spikes-tailored activation functions; (2) neuromorphically implementing Stochastic Gradient Descent (SGD), with recurrent neural connections; and (3) utilizing neuromorphic online learning. ANNs were defined using Keras, and SNNs were simulated using the Neural Engineering Framework (NEF).

NEF brings forth a theoretical framework for a neuromorphic representation and transformation of mathematical constructs with spiking neurons, allowing the implementation of functional large-scale neural networks. NEF is extensively used to design neuromorphic systems capable of visual perception[10] and motor control.[11] It serves as the foundation for Nengo, a Python-based "neural compiler," which translates high-level descriptions to low-level neural models.[12] NEF-inspired neuromorphic hardware designs[13] have been implemented in both analog and digital circuitry. A version of it has been compiled to work on the most prominent neuromorphic hardware architectures available, including Intel's neuromorphic Loihi circuit.[14] Here, we converted ANNs to SNNs using NengoDL[15] and implemented neuromorphic SGD using the Nengo-based Gyrus framework. We trained the networks offline and evaluated them on specialized hardware. ANNs and SNNs were evaluated on NVIDIA's Xavier board. SNNs were also assessed on Intel's Loihi chip.

We demonstrate how ANNs with no prior joint data can provide IK with fast inference time and accurate results. While their spiking counterparts were shown to be less accurate, they provide sufficient neuromorphic approximations.

RESULTS

Neural network size and inference time

In this work, we examined the performance of various neuronal architectures (Figure 1), each with a different number of parameters and inference time. Data are summarized in Table 1. Table 1 also includes the inference time for conventional numerical optimization (with the AGX Xavier) using Jacobian inverse for comparison.

We evaluated the loss from Equations 1 and 2 with two Fully Connected (FC) layer ANNs (Figure 2A). Results demonstrate the superior performance of the regularized loss functions for target accuracy within 1 cm and 1 mm distance across Tanh, Rectified Linear Unit (ReLU), and leaky ReLU activations (Figure 2B). We used shallow networks to evaluate the regularization scheme, thus justifying regularization in further assessment. We further explored IK performance for ANNs with varying depths and widths across the three activation functions. We measured the mean accuracy and the percentage of points for which accuracy of >1 mm and >1 cm has been reached for each architecture. Results for 200,000 training points, 2 to 10 layers, across Tanh, ReLU, and leaky ReLU are shown in Figure 2C. Results demonstrate the superior performance of the 5 × 128 FC architecture (<1 ms inference time, < 1 cm accuracy).

We further explored this architecture with a width of 256 neurons (Figure 2D) and 200,000 data points (Figure 2E), demonstrating superior performance with a six-layer network. This network has 0.18% < 1 cm accuracy with <1 ms inference time (Table 1). To further investigate the computational capacity of ANNs for IK, we utilized ResNets, featuring >3^6parameters, and skip links (Figure 2F; Table 1). ResNets were found to have comparable performance with 1 mm accuracy specification (Figure 2G). We explored our data-driven energy-based loss function (described in Equation 3) with varying depth (Figures 2H–2J), demonstrating dramatic improvement compared with the performance gained with traditional loss definitions. We show that a six-layer ANN's mean error distance was improved by ∼10x (∼2 mm–0.2 mm) and that the percentage of points above the 1-cm threshold dropped by ∼10x (∼60%–6%). We evaluated the energy-driven network with Swish and Mish activations, demonstrating further performance improvement, with a 33% drop in the percentage of points above the 1-cm threshold (4%) (Figure 2K).

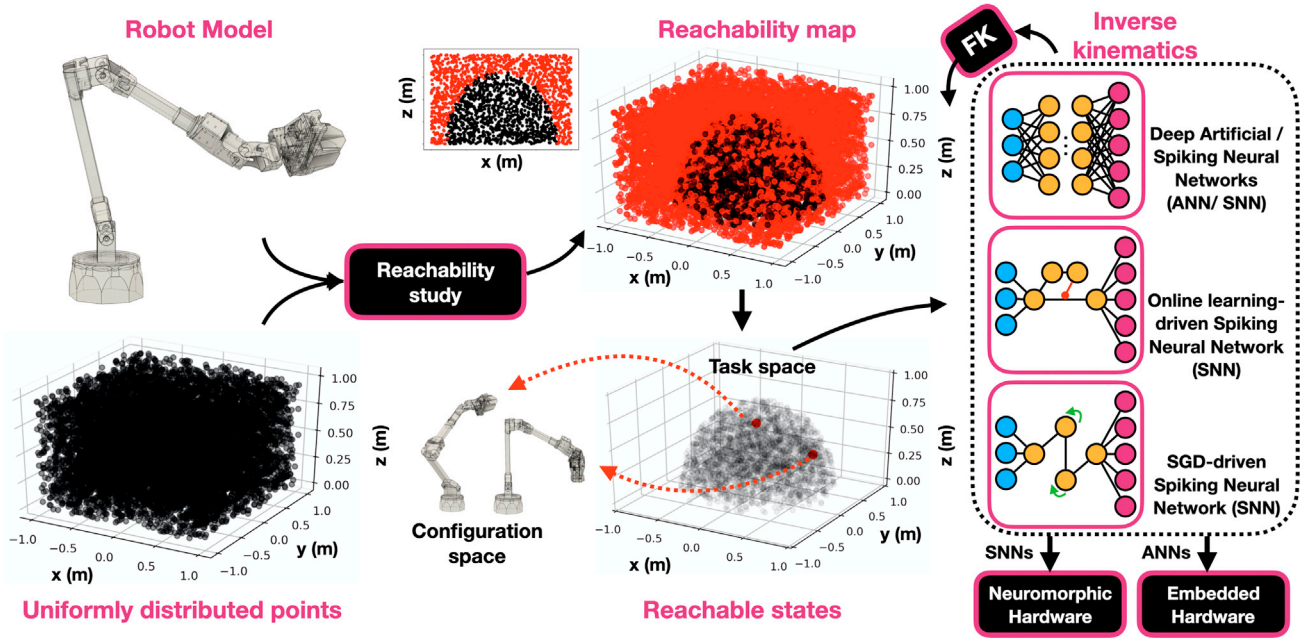


Figure 1. System design

A reachability study was conducted using a robotic model and evaluated with numerical optimization reaching to 200,000 uniformly distributed points across a 2 × 2 × 1-m space. Reachable points and the forward kinematic model are used to train deep ANNs and SNNs as well as recurrent and learning-based SNNs, providing predictive models for IK.

We investigated the performance of NengoDL-based deep SNNs (Figure 3A) for IK. We used ANN to SNN transfer learning by modulating neurons' tuning curves into a differentiable form (Figure 3B, see methods for details). Network performance with and without transfer learning from their artificial counterparts produced similar results. ANNs can be therefore translated to SNNs by defining neurons as soft leaky-integrate-and-fire (LIF) spiking neurons and temporally integrating spikes. In such implementation, the spiking neurons' differentiable approximations are used during training, while the spiking neurons themselves are used during inference.

We explored deep SNNs with varying depths, synaptic smoothing factors, and maximal fire rates (see methods for details). As expected, as the neurons' firing rates increase, they approximate their non-spiking versions more closely as their integrated (or convolved) spike trains more closely resemble a non-spiking behavior, thus producing lower mean error and a higher percentage of targets within 1-cm accuracy (Figure 3C). Smoothness was shown to be most effective with a 20-ms time constant. A heatmap demonstrating the results is shown in Figure 3D. As expected, a comparison between spiking to non-spiking networks shows the superior performance of conventional ANNs. Note that the high frequency of spikes required for accurate predictions is not attainable with current neuromorphic hardware. Therefore, a more neuromorphic-appropriate design would utilize recurrent or learning-based neuronal designs.

Learning-based approaches can be either pre-trained to achieve a better initial weight scheme or naively initialized to zero. Pre-training was implemented by Prescribed Error Sensitivity (PES) on the entire training set, where the resulted weights were averaged and saved for network initialization. Note that network training does not constitute a general solution for IK. IK is resolved using PES-driven online learning for each target point, as was described above. We show that pre-trained models have faster inference. Inference performance is shown in Table 2. SNNs were deployed on Intel's Loihi circuit (with a 1-kHz spiking rate).

Recurrent and learning-based SNN

Gyrus-based (Figure 4A) derivation of J_a^+ is demonstrated in Figure 4B (at point [0.17, 0.17, 0.35]), where the 3x5 = 15 Jacobian's values are approximated. Reaching that point via the derivation of the appropriate joint configuration is shown in Figure 4C. Another approach would be using a learning-based derivation of IK via the PES learning rule (Figure 4D). The network architecture is discussed in detail in Zaidel et al.[11] Error convergence and reaching point [0.17, 0.17, 0.35] are demonstrated in Figures 4E and 4F. The comparison between learning and SGD-recurrent implementation is shown in Figure 4G.

Results demonstrate the learning-based approach's superior performance. While the mean error is higher with a learning-based approach, it is biased by points to which reach was not successfully computed. The mean error for the points to which the learning-based approach was able to converge was approximately 1 mm, as is demonstrated in the histogram. Note that it takes time for the network to accurately compute the FK and the Jacobian used to calculate IK in the online learning-based method due to the neurons' response dynamic (synaptic time constant). A raster plot for the error representing neurons in the learning-based SSN is shown in Figure 4H, showing convergence pattern of spikes, reaching

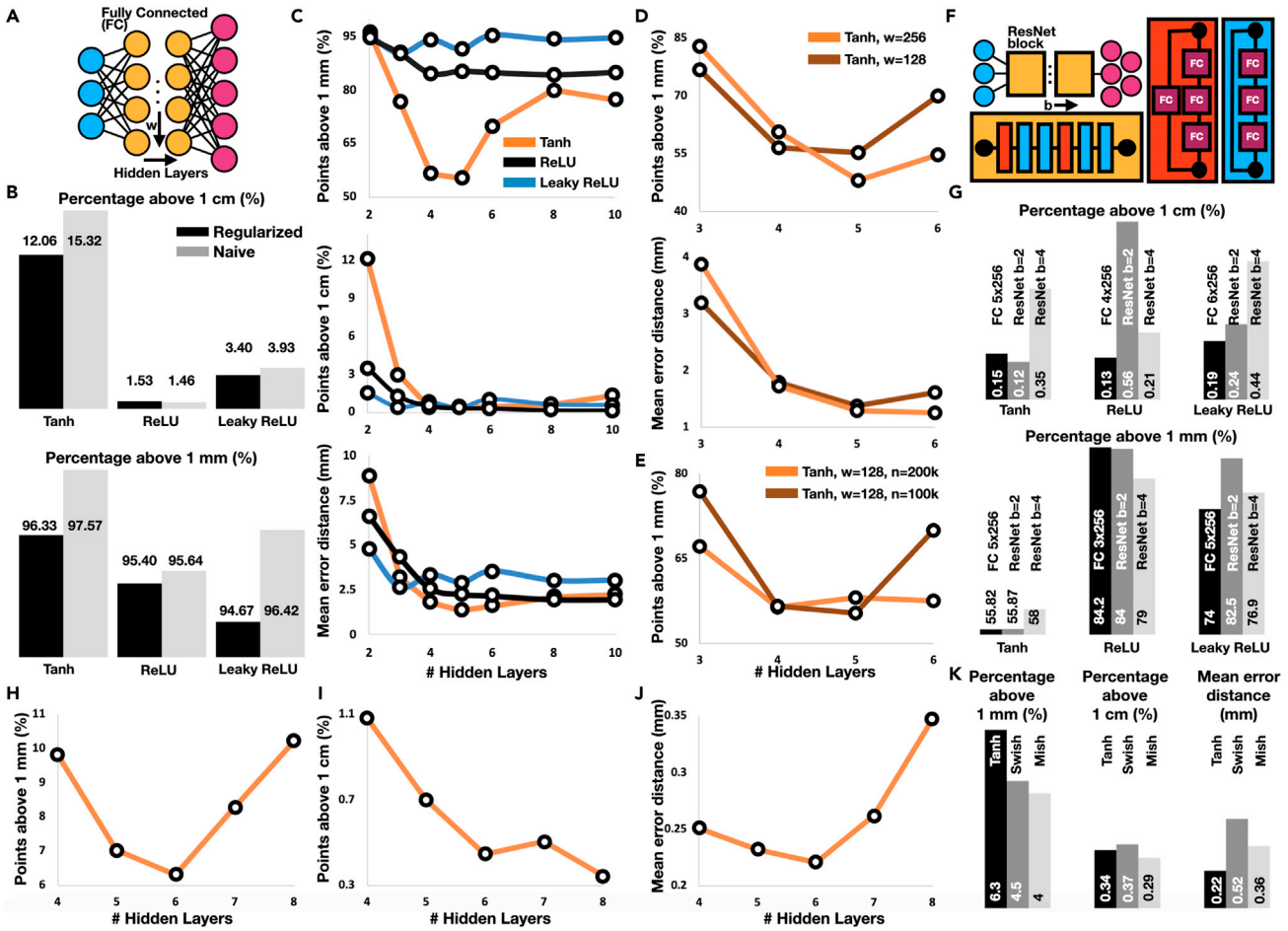


Figure 2. Artificial neural networks for inverse kinematics

(A) ANN schematic.

(B) Superior performance of regularized loss functions with shallow 2 × 128 neural networks.

(C) The percentage of target points within 1 mm (top), 1 cm (middle), and the mean error distance (bottom) for ANNs with varying depths and activations.

(D and E) Performance for Tanh activated ANN with varying width, depth (D) and training data size (E).

(F) Residual neural networks architecture.

(G) Residual networks performance compared with FC ANNs with varying architectures and activations.

(H–J) The percentage of target points within 1 mm (H) and 1 cm (I), as well as mean error distance (J) for FC 5 × 128 ANNs, Tanh activated, with energy loss function.

(K) Comparison between FC 5 × 128 ANNs with energy loss function with Tanh, Swish, and Mish activations.

homogeneous activation pattern after ~2.4 s. Note that the error is rate-coded following the neurons' tuning curves. Some neurons decrease their firing rate while others increase it with reduced error. Results show a significantly higher inference time of SNNs (increasing from milliseconds to seconds' range) required for computing convergence. The SGD-based network requires a relatively high number of neurons for reasonable results, pointing out the learning-based approach's advantages. Inference time was calculated for points for which 1-cm convergence was achieved. We found that higher accuracy is difficult to attain in our current configuration (number of neurons, high DoF). The number of parameters and inference times are detailed in Table 2. Note that we've limited the evaluation metric to a 1-cm threshold due to the limited accuracy attained using SNNs (mean error distance is in the order of a few mm). While a few millimeters of deviation, for many applications, might be sufficient, it highlights the need to use further adaptation (e.g., from sensors), allowing more accurate targeting (see discussion).

Geometrically constrained IK

We used our best-performing five hidden layers ANN to calculate geometrically constrained IK. The network was trained to avoid collision with radial obstacles featuring radii of 10 and 20 cm, following Equation 5. Reachability map (see methods for detailed description) and example of 100 configurations, calculated to avoid a 20-cm obstacle, are shown in Figure 5. We measured the mean accuracy and the percentage of points for which accuracy of >1 mm and >1 cm was reached with 10- and 20-cm spherical obstacles. Results across Tanh, ReLU, leaky ReLU, Swish, and Mish activations are shown in Figure 6A. Results demonstrate the superior performance of Mish activation. To

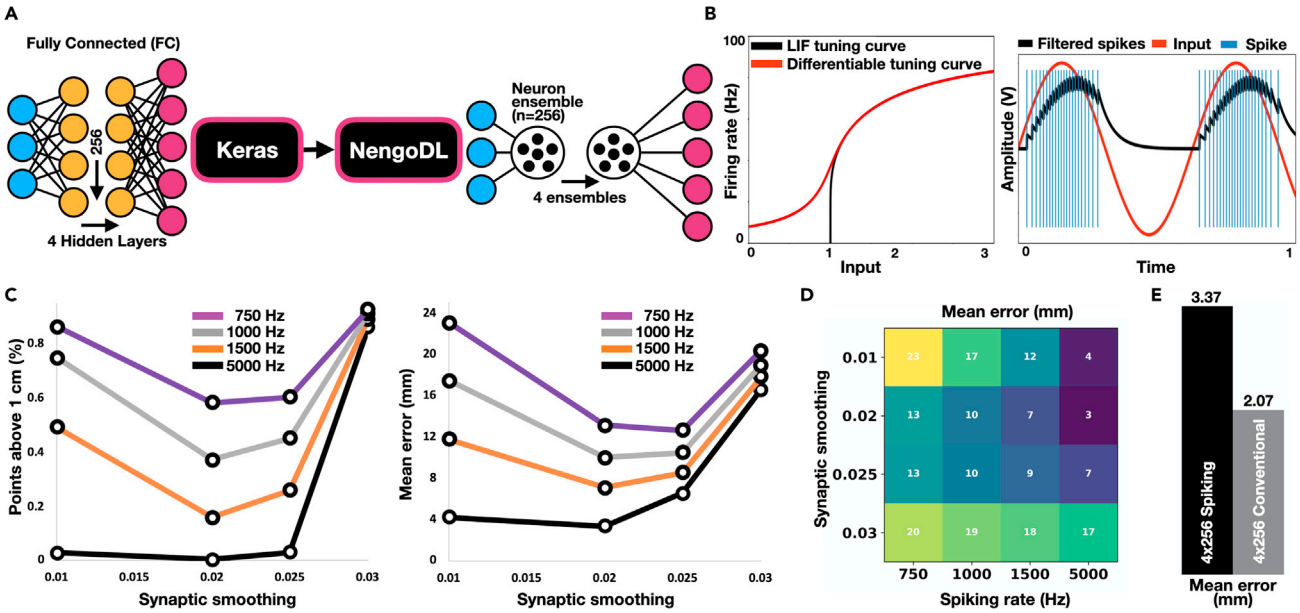


Figure 3. Deep spiking neural networks for inverse kinematics

(A) ANNs can be converted to SNNs using the NengoDL framework, where neurons are defined in ensembles.

(B) Differentiable temporally integrated activation of LIF-based spiking neurons.

(C and D) The percentage of target points within 1 cm (left) as well as mean error distance (right) for SNN with varying depth, synaptic smoothing, and maximal firing rate. Mean error heatmap is shown in (D).

(E) Mean error comparison between the best-performing SNN and its ANN counterpart.

illustrate the importance of network optimization for obstacle avoidance, we compared its performance when optimized to minimize Equations 3 and 5. Our comparison shows that the number of intersections with a 20-cm spherical obstacle is dramatically reduced by a factor of 4 when the loss function considers obstacle avoidance (Figure 6B). We further compared the intersections with obstacles across the different activations, showing that while Mish activation outperformed all other activations for large obstacles, performance plateaued at 10% for small obstacles (Figure 6C). Given our relatively small dataset, we compared our ANN performance to a two-layer ResNet, demonstrating ANN's superior performance (Figure 6D).

We evaluated this dataset with SNNs via transfer learning, as was discussed above. Interestingly, we show that the obstacle intersection performance is similar across the different maximal firing rates. In contrast to the constraint-free scenario, where network performance was found to heavily rely on spiking rate (Figure 3C), spiking rate is not a critical factor for obstacle avoidance (Figure 7A). This result demonstrates the robustness of an SNN to efficiently avoid obstacles (a low spiking rate constitutes reduced energy consumption). However, when accuracy is considered, SNN's performance heavily relies on the neurons' maximal firing rate and synaptic time constant (Figure 7B). As the maximum firing rate increases, the more accurate IK becomes (Figure 7C). Similar to the non-constrained case (Figure 3C), we show that a synaptic smoothing factor of ~20 ms outperforms a faster and slower synapse configuration (Figure 7D).

We further evaluated our data-driven (200k sample points) best-performing five layers (x 128) FC Mish-activated ANN in a convoluted space featuring two to five randomly positioned 10 cm in radius obstacles. For training, we used Equation 6, a generalized form of Equation 5 we used for a single obstacle. A demonstration of the resulting 100 targets reaching in a five-obstacle environment is shown in Figure 8A. As expected, the number of out-of-reach targets in multiple obstacle environments increased with the number of obstacles (Figure 8B). However, as the obstacles were randomly positioned, some obstacles may constrain the arm's configuration space to a greater degree than others, as is demonstrated in the number of reachable points in the three-obstacle scenario, which is higher than in the four-obstacle scenario (Figure 8C). The resulting distances to targets in the one- to five-obstacle environments demonstrate the modularity and high performance of the ANN-based IK (Figures 8D–8F). This is particularly evident in the number of obstacle intersections, which is kept lower than 1% in all tested scenarios (Figure 8G). While network performance generally increases with fewer obstacles, it is not always true, as is evident with the similar

Table 2. SNN number of parameters and inference time

	Hardware	# parameters	Inference (s)
Deep SNN 4x256	Intel's Loihi	199,685	0.4
Learning-based SNN	Xavier	5,000	2.9 ± 1.22
Learning-based SNN	Intel's Loihi		3.4 ± 0.61
Learning-based SNN (pre-trained)	Intel's Loihi	5,000	2.6 ± 0.04
SGD-recurrent SNN	Xavier	300,000	3.8 ± 0.62

Inference time includes the standard deviation computed on the entire test set.

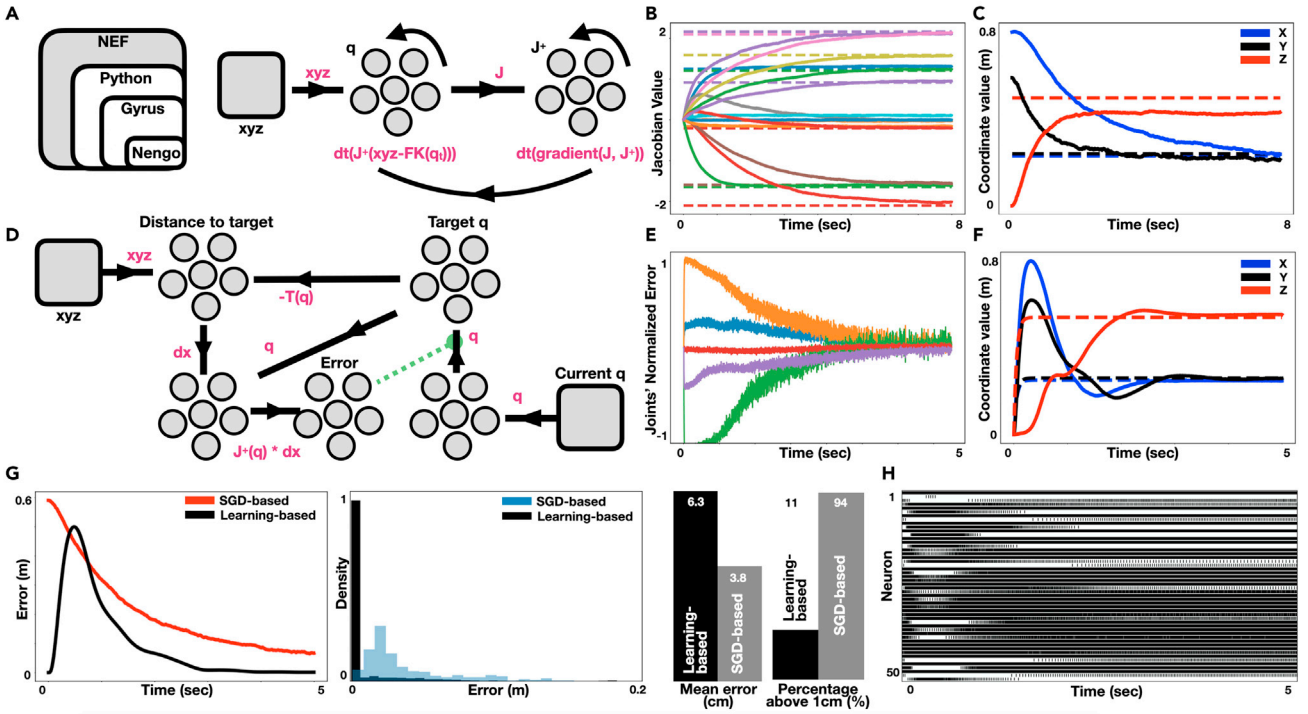


Figure 4. Learning and recurrent SNNs for inverse kinematics

(A) Simplified recurrent SNN for IK.

(B) Inverse Jacobian approximation with SNN (15 values approximating the 3 × 5 Jacobian matrix).

(C) Exemplified reaching to point p = [0.17, 0.17, 0.35] with a recurrent SNN.

(D) Learning-based SNN for IK.

(E and F) Error convergence and reaching p with learning-based SNN.

(G) Compared error convergence (left), accuracy histogram (middle), the percentage of target points within 1 cm, and mean error distance (right), for learning and recurrence-based SNNs.

(H) Raster plot for error representing neurons in the learning-based SNN.

performance measured for the three- and four-obstacle scenarios. Thus, pointing out again the importance of the obstacle position on the ability of the arm to navigate efficiently.

DISCUSSION

Biological motor control is considered far superior to our most advanced robotic systems, which predominantly rely on analytical and numerical control. Therefore, there is a great promise in neurorobotics that strives to use neuronal architectures to guide robotic systems' behavior. Neural networks have the enticing capacity to be trained to perform tasks without an analytical solution; however, the successful application of neural networks to robotics, beyond sensory processing, remains limited.[16] In contrast to numerical and analytical descriptions of IK, as long as an FK is available, a neural network-powered IK provides a unified framework for motion planning for robotic systems with arbitrary complexity, operating in arbitrarily convoluted environments, subjected to any definable constraints.

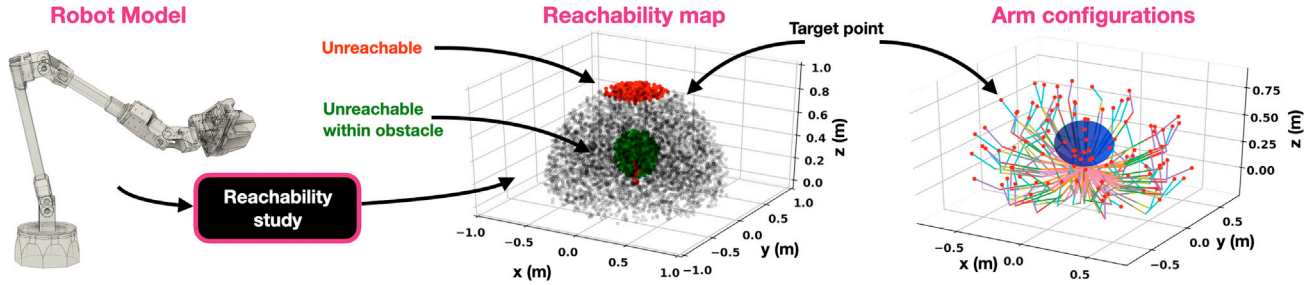


Figure 5. Geometrically constrained IK

A reachability map indicates the location of a 20-cm spherical obstacle (colored green) and out of reach points (colored blue). A demonstration of 100 configurations to target locations (red dots) is shown on the right. Each of the robot's links is colored differently.

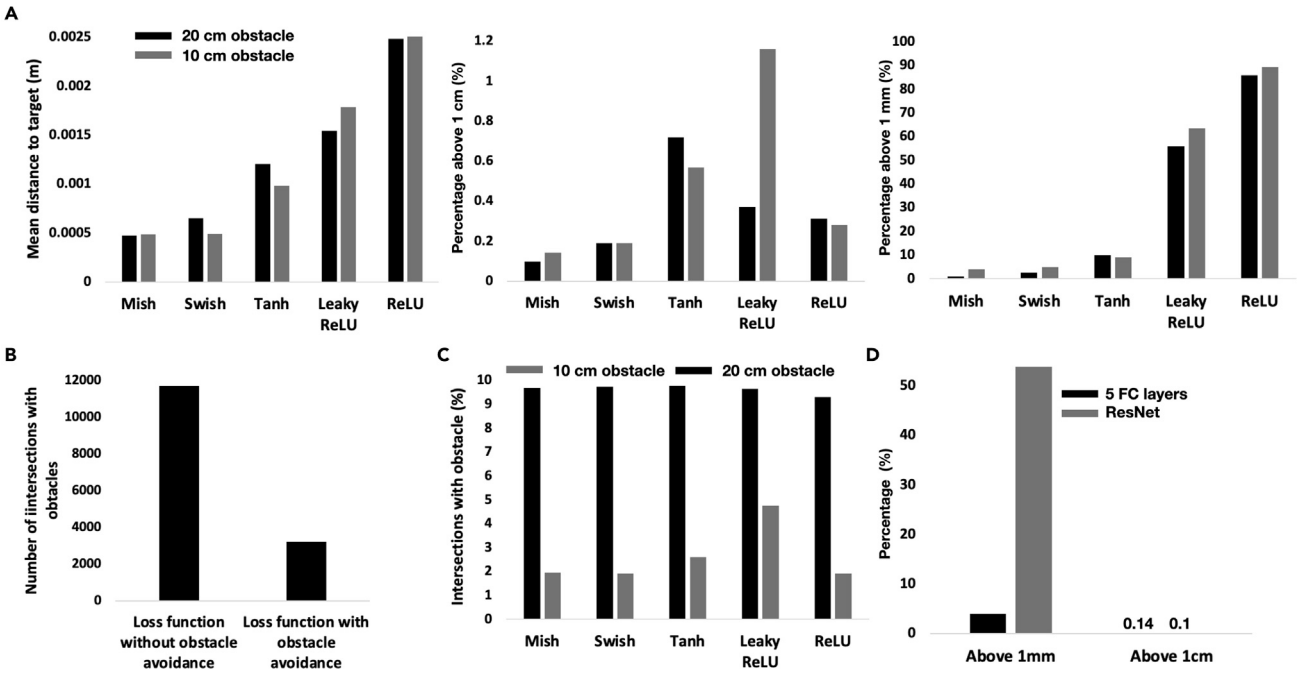


Figure 6. Geometrically constrained IK with ANNs

(A) Mean accuracy and the percentage of points for which accuracy of <1 mm and <1 cm was not achieved with 10- and 20-cm spherical obstacles. Results achieved with a 5-layer ANN with Tanh, ReLU, leaky ReLU, Swish, and Mish activations.

(B) Number of the robot and the spherical obstacle intersections with (Equation 3) and without (Equation 5) consideration for obstacle avoidance.

(C) Number of the robot and the spherical obstacle intersections yielded with a five-layer ANN with Tanh, ReLU, leaky ReLU, Swish, and Mish activations.

(D) Percentage of points for which accuracy of <1 mm and <1 cm was not achieved with a five-layer Mish-activated ANN and two layers ResNet.

In this work, we investigate the utilization of ANNs and SNNs for IK, the most fundamental problem in robotics. The utilization of ANNs for IK has been vastly explored. For example, Almusawi and colleagues enhanced a deep ANN with known joint configurations to solve IK for a 6-DoF robotic arm[17]; Wand and colleagues recently enhanced conventional ANNs with a damped least square optimization to provide faster convergence to the desired IK threshold[18]; Duka and colleagues generalize ANNs for EE trajectory planning[19]; and Li and colleagues[20] further generalized ANNs to constrained trajectories[20]. Here, we further explored ANN-based solutions to IK, with various configurations and activations, having only the FK model and a reachability map as inputs. Our models presented here were not trained to elucidate known joint configurations but rather elucidate robot

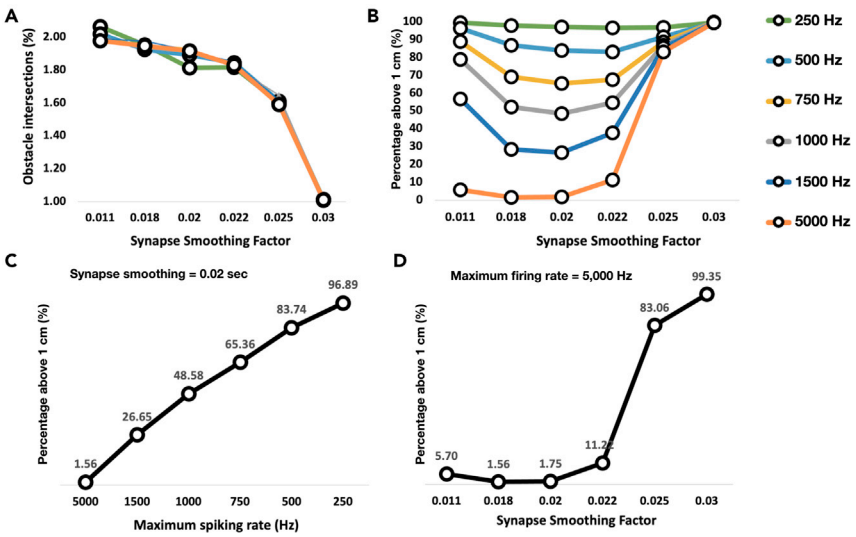


Figure 7. Geometrically constrained IK with SNNs

(A) Obstacle intersections with five-layer SNNs featuring various maximal firing rates across different synapse smoothing factors.

(B) Percentage of points for which accuracy of <1 cm was not achieved with five-layer SNNs featuring various maximal firing rates across different synapse smoothing factors.

(C) Percentage of points for which accuracy of <1 cm was not achieved with five-layer SNNs featuring various maximal firing rates and a synapse smoothing factor of 20 ms.

(D) Percentage of points for which accuracy of <1 cm has was not achieved with a five-layer SNN featuring various synapse smoothing factors and a maximal firing rate of 5,000 Hz.

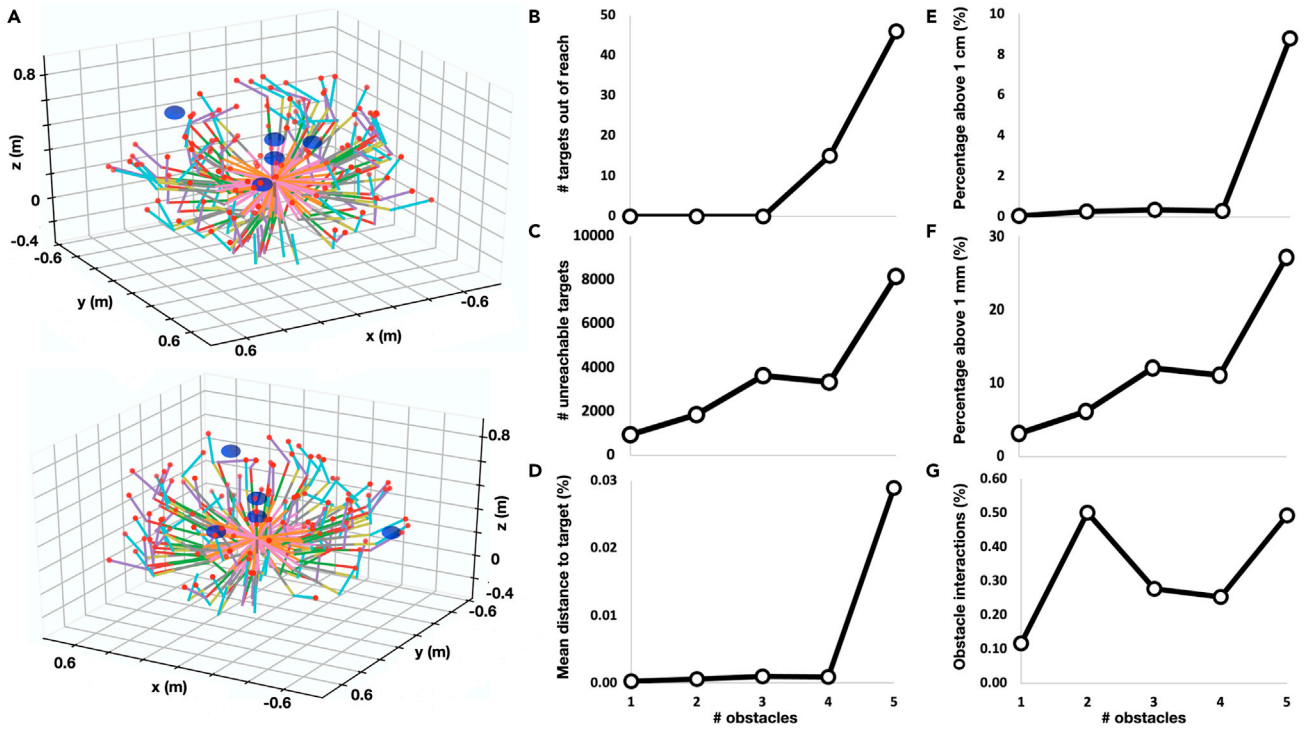


Figure 8. Multiple obstacles constrained inverse kinematics

(A) A demonstration of 100 targets reaching in a five-obstacle environment.

(B and C) Number of out-of-reach (B) and reachable targets (C) in one- to five-obstacle environments.

(D) Mean distances to targets in one- to five-obstacle environments.

(E and F) Percentage of points for which accuracy of <1 cm (E) and <1 mm (F) was not achieved in one- to five-obstacle environments,

(G) Number of obstacle intersections in one- to five-obstacle environments.

configurations by optimizing EE's Euclidean distance from its target using known FK. Our training data are therefore ambiguous, as multiple joint configurations can realize the same position in task space. We demonstrated that training a neural network with ambiguous data results in low accuracy, mainly when a fine threshold is set (1 mm in contrast to 1 cm). A common practice is to resolve this ambiguity by defining a default configuration and using Mean Square Error (MSE) to that set of joint angles so that all solutions are attempting to be near the same configuration (constituting a "null controller").[21] As long as none of the points in the null space are at a singularity, this method usually works well.

Here we have taken a different, data-driven approach. We reduced ambiguity by using an energy-driven loss function and training the network with numerous neighboring joints' configurations such that they will be mapped to close points in task space. This method does not require having one reference joint's configuration to which all configurations relate. Note that since we use many pairs across the robot's reachable map and that across the configuration space a transitive property holds, the network is driven to overall consistency. This energy-driven loss resulted in dramatically improved accuracy (10x in terms of mean error distance and a smaller percentage of points above the 1 cm accuracy threshold). This loss function allows for IK-based motion planning with smooth transitions between targets. Using an energy term to achieve redundancy resolution in IK does not require a new dataset but rather a rational choice of neighboring target points for training. This data-driven approach has the flexibility of handling complex scenarios, as we showcased with geometrically constrained IK, discussed below.

We further demonstrate Swish and Mish activations' superior performance for energy-driven ANNs and explored overparameterized architectures such as ResNets. While it is often unattainable to explain the underlying reasons for the differences in performance across the different activation functions, we show that across the traditional activations, Tanh has better performance. We show that the Tanh-dependent Mish activation further improves accuracy. Probably due to its positively unbounded, negatively bounded, smooth, and nonmonotonic characteristics.[22] We used ResNets, which are very rich in parameters and feature skip links for IK. These skip links allow "skipping" non-contributing parameters, thus, allowing a better estimation of the capacity of ANNs to solve the problem given a dataset. We show that millions of parameters ResNets underperformed conventional FC ANNs, suggesting that we maximized the capacity of ANN to resolve IK with our current dataset size.

Uniquely, we extend the discussion to spiking neuronal architectures. In recent years, neurorobotics, in which SNNs are the underlying computational framework, have gained increased attention.[23] Neurorobotics are argued to outperform conventional control paradigms in terms of robustness to perturbations

and adaptation to varying conditions.[11,16] However, neuromorphic implementation of robotic control is usually tailored toward adaptive control[9] rather than utilized to perform exact localization. This is since spanning high-dimensional space (as required by robotic systems with high DoF) requires a large number of neurons.[13] Recently, learning-based IK was implemented with SNNs, demonstrating how carefully tuned neuromorphic encoding can be used to perform high-dimensional nonlinear computations.[11] Here we further extend the discussion. We investigate deep SNNs, recurrent and learning-based SNNs for IK. Deep SNNs were shown to perform well for various perception tasks.[24]

By performing ANN to SNN transfer learning, we show that SNNs require a high spiking rate to approximate traditional ANN performance. Spiking rate is correlated with increased energy expenditure, implying a non–energy-efficient utilization of their capacity. Rate-coded SNNs are known to be limited in their capacity to perform exact computations. They are much better when realizing a learned behavior. With deep SNNs, we achieved a few millimeters of mean deviation from the target. While a few millimeters distance from the target might be sufficient for many applications, it highlights the need to use further adaptation (e.g., from sensors), allowing more accurate targeting. We further show that while highly parameterized neural networks with tens to hundreds of thousands of parameters exhibit very high accuracy, learning-based spiking architectures can provide reasonably good results with merely a few thousand neurons. Note that considering a typical robotic hardware response time and accuracy, we consider 1-mm accuracy is sufficient and millisecond-range response time as good enough in most applications.

To further evaluate SNN for IK, we assessed a spiking SGD. For the first time, we used the Nengo-based Gyrus environment for IK. Gyrus provides a numerical computing framework for SNNs, with which we designed gradient-based approximation of IK (following the traditional Jacobian inverse guidelines). While successfully implemented, we show that for IK, SNNs are more efficiently utilized with real-time learning rather than used to approximate numerical methods. Finally, we show that when the number of intersections with an obstacle is considered, SNNs can perform very well, even when configured to an energy-conserved spiking rate, demonstrating their robustness. In this work, we show that SNNs underperform ANNs in terms of accuracy and inference time. However, we believe that an exploratory utilization of SNN is of high importance when such a fundamental problem in neurorobotics is addressed, mainly when a pure neuromorphic robotic control is desirable.

This work can be further extended to include other SNN-based optimization methods, such as adding a firing rate regularization parameter during training or using amplitude scaling. Moreover, it could be used to investigate its utilization in trajectory planning and in dynamic scenarios where obstacles are in motion.

We note that the methodology described here can be utilized in other areas featuring redundancy resolution, where one solution should be chosen from an infinite set. Underdetermined systems are of interest to a broad range of disciplines. For example, recently, Hyun and colleagues highlighted that underdetermined inverse problems had become one of the major concerns in the medical imaging domain (e.g., under-sampled magnetic resonance imaging, interior tomography, and sparse-view computed tomography).[25] Their work laid down the mathematical foundations for utilizing neural networks to handle such underdetermined problems. Similar challenges are addressed in hydrology and geophysics,[26,27] pharmacokinetics,[28,29] and image reconstruction.[30]

EXPERIMENTAL PROCEDURES

Resource availability

Lead contact

Elishai Ezra Tsur at elishai@nbel-lab.com.

Materials availability

This study did not generate new unique reagents.

Data and code availability

Data and code is available at https://github.com/NBELab/Patterns_2021.

Robotic model

The simulated robotic arm model has six DoF, out of which, here, we controlled five (the sixth DoF accounts for gripping). The model follows the physical arm design described in Zaidel et al.[11] and is visualized in Figure 1. The arm features six links l_{1-6} (12-, 30-, 6-, 20-, 10-, and 20-cm long, respectively) and five joints q_{1-5} (rotated around the y, x, x, z, x axes, respectively). The arm has an 82-cm reach and a 1.64-m span. Visualization and measurements were performed using Autodesk's Fusion 360 CAD software.

FK was implemented using transformation matrices. For our five joints robotic arm, FK takes the form of $T = T_{01}T_{12}T_{23}T_{34}T_{45}T_{56}$, where T_{ij} is the transformation matrix (rotation, translation) in homogeneous coordinates, mapping coordinates at joint i axis to coordinates at joint j axis (indices 0 and 6 refer to the world and EE axes, respectively). We initialized T with the appropriate set of rotations and translations and multiplied it by a zero vector $[0,\ 0,\ 0,\ 1]^T$, resulting in an FK model $T_x(q)$. $T_x(q)$ returns the EE position in the world's coordinate system, where the origin is at the robot's base.

Jacobian inverse

IK can often not be analytically solved, and it is usually numerically optimized using the Jacobian inverse. The Jacobian J_a relates the change of the EE position x to the evolution of joint angles q using $J_a(q) = \delta x/\delta q$, constituting $\dot{x}(q) = J_a(q)\dot{q}$, where $\dot{x}$ is the change in EE position, resulting from a shift in the robot's configuration $\dot{q}$. As the Jacobian is not necessarily invertible, a common practice is to use its pseudo-inverse form $J_a{}^+$ constituting $\dot{q} = J_a{}^+\dot{x}$. Therefore, given an error in space coordinates x_d as the distance between the EE current position x_c, and its target position x_y, the appropriate change in joint space can be computed using $d(q) = J_a{}^+(q)x_d$, where $d(q)$ is the difference in joint angles, with which the robot's EE will get closer to its target. In each iteration, this equation is re-evaluated until x_d is within some accuracy threshold. The Jacobian inverse is more elaborately discussed in Buss.[31]

Artificial neural networks

Deep ANNs comprise layers of computational entities characterized with differentiable, nonlinear activation functions and weighted inputs. ANNs can be optimized, with gradient-based training, to provide predictive models.[32] ANNs can be defined using the open-source Python-based software library Keras, providing a layer of abstraction for the TensorFlow library.[33] Here, we used Keras to define FC ANNs with varying depth (number of "hidden" neural layers), width (number of neurons per layers), and activation functions. For activation, we evaluated the performance of standard Tanh, ReLU, and leaky ReLU activations, as well as the more recently defined functions, Mish and Swish. Swish activation is defined using $f(x) = x(1+e^{-\beta x})^{-1}$, where β is a trainable parameter. Depending on the value of β, Swish activation interpolates the Sigmoid-weighted linear function and the ReLU activation function.[34] Mish is a smooth nonmonotonic function defined using $f(x) = x\cdot\tanh(\ln(1+e^x))$.[22]

We further utilized Residual neural Networks (ResNets) for IK, first introduced for image classification and recently adopted for other tasks, including nonlinear regression.[35] A ResNet block can be described as a stack of one *density* and two *identity* blocks, integrating a varied and preserved number of features, respectively (Figure 1). Here, we started with 64 features (neurons),

doubling in every dense block. All ANNs had a five-neuron FC output layer, corresponding to the configuration space dimensionality.

Network optimization

A reachability study was conducted using the robotic model, evaluated with numerically Jacobian-based optimized reaching to 200,000 uniformly distributed points across a 2 × 2 × 1-m space. Numerical reachability was defined with a threshold of 1 mm proximity. To synthetically generate datasets for geometrically constrained task spaces, we implanted spherical obstacles of various radii and uniformly sampled 200,000 target points. We evaluated spheres with 10 and 20 cm in radius located at the target space's origin. Every target point was evaluated for reachability. We geometrically evaluated if the arm's total length is long enough to reach that target point when bent around the obstacle. Target points within obstacles were also defined as unreachable.

The dataset was divided into three parts: training (70%), validation (15%), and test (inference) (15%) data. We defined three loss functions: naive, regularized, and energy-based. We define the naive cost function L using:

$$L = \frac{1}{3}x_i - FK(\hat{y}_i)_1, \qquad \text{(Equation 1)}$$

where FK is a $\mathcal{R}^5 \rightarrow \mathcal{R}^3$ mapping, $\hat{y}$ is the predicted joint configuration, and x_i is the target EE position. IK is not an injective function due to the angles' 2π periodicity and the fact that several joint configurations can realize the same target in task space (the system is underdetermined). Underdetermined systems have infinitely many least-squares solutions. Like pseudo-inverse techniques, L2 provides the least-squares solution with a minimum norm, which is unique:

$$L = \frac{1}{3}\|x_i - FK(\hat{y}_i)\|_1 + \lambda\|\hat{y}_i\|_2^2, \qquad \text{(Equation 2)}$$

where λ is a regularization coefficient, set here as 10^{-5}.

However, regularization does not resolve the ambiguities completely, since it does not consider joint identity nor sign (L2 cost for $q_1 = -\pi/2, q_2 = -\pi$, equals its cost for $q_1 = \pi/2, q_2 = \pi$). Therefore, we propose adding an energy term guided by a heuristic according to two close points in configuration space should be close in task space. This heuristic allows for abrupt transitions between targets, with minimum changes in the robot's configuration. Here, similarity in task space was defined by Euclidean distance, and configuration similarity was determined using MSE:

$$L = \frac{1}{3}\|x_i' - FK(\hat{y}_i')\|_1 + \frac{1}{3}\|x_i'' - FK(\hat{y}_i'')\|_1 + \lambda\|\hat{y}_i' - \hat{y}_i''\|_2^2, \qquad \text{(Equation 3)}$$

where y_i' and y_i'' are the model prediction for x_i' and x_i'' (close points in task space), respectively. Here we set $\lambda = 0.1$. In Equation 3, the first two terms minimize the distance between the predicted and target configurations. The third term aims to reduce the change in configuration space between neighboring points in the task space. With this loss function, the network is trained with points (x_i', x_i'') within a distance threshold (here, we chose 3 cm).

Equation 3 defines a loss function responsible for reaching a target point in task space such that the redundancy resolution is resolved. To account for geometrical constraints, we incorporate another term, which is augmented significantly when any part of the robotic arm approaches an obstacle. We would like that the closer any of the robotic arms' links get to the center of a spherical obstacle, the loss function gets larger. This loss function should consider penalty O_i for each link i. O_i should increase dramatically when the arm crosses the sphere's boundary. It can comprise the sum of the squared distances from each arm's links edges and midpoint to the sphere's center (three points were evaluated for each link). To implement rapid increase across the sphere's boundary, we define O using a parameterized inverse logistic function:

$$O = \frac{e^{s(d-r+pd)}}{1+e^{s(d-r+pd)}}, \qquad \text{(Equation 4)}$$

where s represents the curve slope, d is the squared distance from a link's point to the sphere's center, r is the squared sphere's radius, and pd is a penalty epsilon, which controls the value of x above, which the function initiates a rapid ascent and s is the ascent's rate. Here we empirically set $s = -1.5$ and $pd = 2.5$. When $x = 0$, the distance between a link's point and sphere boundary (its radius), and O rapidly climbs. Following Equation 3, Equation 2 can be further extended as:

$$d = x_t' - FK(q')_1 + x''_t - FK(q'')_1 + \lambda q' - q'_2^2 + \sum O_i \qquad \text{(Equation 5)}$$

To handle a convoluted environment featuring multiple obstacles, Equation 5 can be generalized to:

$$d = x_t' - FK(q')_1 + x''_t - FK(q'')_1 + \lambda q' - q'_2^2 + \sum_{o=1}^{n}\sum O_i \qquad \text{(Equation 6)}$$

where n is the number of obstacles.

For ANN training, we used batch SGD, with a batch size of 10 and an initial learning rate of 0.1. The learning rate was scheduled to reduce by 20% when an error plateau is reached, defined as a non-improving error in a 22-epoch interval, with a minimum relative improvement of 0.0005 and a minimum rate of 10^{-6}. Early stopping was scheduled to 750 epochs.

The NEF and online learning

NEF-based neuromorphic spikes' rate coding of numerical input vectors (or stimuli) x is defined as $a_i(x) = G_i[\alpha_i e_i \cdot x + J_i^{bias}]$, where a_i is the spike rate of neuron i, G is the LIF neuronal model,[36] α is a gain term, e is the encoding vector (the value for which the neuron is firing with the highest spike rate), and J^{bias} is a fixed background current. An ensemble of neurons collectively represents a stimulus x as $\hat{x}$ using $\hat{x} = \sum a_i * h d_i$, where d_i are linear decoders, which were optimized to reproduce x using least squared optimization and $a_i * h$ is the spiking activity a_i convolved with filter h. Similar to decoder optimization, it has been shown that any function $f(x)$ could be approximated using some set of functional decoders d^f.[37] Defining $f(x)$ in NEF can be made by connecting two neuronal ensembles A and B via neural connection weights $w_{ij}(x)$ using: $w_{ij} = d_i \otimes e_j$, where i is the neuron index in ensemble A, j is the neuron index in ensemble B, d_i are the decoders of ensemble A, which were optimized to transform x to $f(x)$, e_j is the encoders of ensemble B, which represents $f(x)$, and $\otimes$ is the outer product operation.

Connection weights, which govern the transformation between one neuromorphic representation to another, can also be adapted or learned in real time rather than optimized during model building. One efficient way to implement real-time learning with NEF is using the PES learning rule, which modifies a connection's decoders d to minimize an error signal e_r. e_r is calculated as the difference between the stimulus x and its approximated representation $\hat{x}$. PES applies the update rule: $\Delta d = \kappa e_r a$, where κ is the learning rate. It has been shown that when $1 - \kappa a^2$ (denoted γ; here, a is a vector of firing rates) is larger than -1, the error e_r goes to 0 exponentially with rate γ. PES is described at length in reference.[38]

Spiking neuronal architectures

We demonstrate three distinct spiking metho1ogies: transfer learning from ANNs to SNNs using a spike-tailored activation function, neuromorphically implemented SGD, and online learning.

With deep SNNs, NEF's non-differentiable neurons' response curves (or tuning curves) are modulated to a differentiable form. LIF neurons tuning can be described using: $a = [t_{ref} + \tau \ln(1 + u_{th}/(I_0 - u_{th}))]^{-1}$, where a is the neuron's spiking rate, t_{ref} is its refractory period, u_{th} is its threshold for spike initiation, and I_0 is its input current. A rectified version of it would be:

$$a = [t_{ref} + \tau \ln(1 + u_{th}/\rho(I_0 - u_{th}))]^{-1}, \qquad \text{(Equation 7)}$$

where $\rho(x) = \max(0, x)$. To provide a differentiable model, ρ can be defined as a soft-max function: $\rho(x) = \lambda \log(1 + e^{x/\lambda})$, where λ is a smoothing (low-pass) function.[39] ANNs were defined using Keras, and SNNs were defined using NEF. We used the Python-based Nengo framework to simulate our SNNs and deploy them on Intel's Loihi. Nengo has two useful abstractions: NengoDL and Gyrus, providing interfaces to Keras and Python's numerical computing library, NumPy. We used NengoDL to convert ANNs into SNNs. These deep, layered SNNs are defined with a synaptic time constant (specifying a low-pass filter) and a maximal firing rate.

We neuromorphically implemented Pseudo-inverse-based optimization (described above) using Gyrus. Gyrus recursively generates large-scale

Nengo models using NumPy semantics. However, since Gyrus does not currently support pseudo-inverse NumPy methods (e.g., linalg.pinv), we use SGD to calculate $J_a{}^+$. Following each time step, $J_a{}^+$ is recalculated, and a small joint correction signal is derived. The correction and the Jacobian calculation were implemented with a neuromorphic integrator, allowing past information to influence the current neuronal state. We can realize intricate dynamic behavior by integrating NEF's representation and transformation capabilities by recurrently connecting neuronal ensembles. NEF can be used to resolve the equation $dx/dt = f(x(t)) + u(t)$, where $u(t)$ is an input (the input can be from another neural population), by defining a recursive connection that resolves the transformation: $\tau \cdot f(x) + x$.[10] It can therefore be used to solve differential equations as required here for the derivation of SGD.

Our learning-based SNN follows the model proposed in Zaidel et al.[11] Briefly, the robot's current configuration is transformed to a target configuration via PES-driven learning. Transformational synaptic weights are modulated to minimize the decoded error, calculated from the derived distance to the target. As learning progresses, the error is continually minimized, and the new robot configuration is calculated.

HARDWARE

Algorithms were evaluated on NVIDIA's AGX Xavier and Intel's neuromorphic Loihi chip. The AGX Xavier features 32 TeraOPS (TOPS) 512-Core Volta GPU with Tensor Cores, 8-Core ARM v8.2 64-Bit CPU, and 32 GB RAM. The Xavier is often used in robotic systems.[40] The Loihi chip comprises 128 neuron cores; each simulates 1,024 neurons. The chip includes x86 cores, which are used for spike routing and monitoring.[41] Nengo models were compiled on the Loihi using the nengo_loihi library (version 0.19).[14]

ACKNOWLEDGMENTS

The authors thank the Applied Brain Research (ABR) team for the support; Intel Labs for granting us access to their neuromorphic cloud and technical support; and Andreea Danielescu and Timothy Shea from Accenture Labs for their insightful comments. This research was funded by Accenture Labs as part of Intel's INRC (Intel Neuromorphic Research Community) initiative and by the Open University of Israel research grant.

AUTHOR CONTRIBUTIONS

E.E.T., A.V., Y.Z., and A.S. conceptualized the study; E.E.T., A.V., Y.Z., A.S., and T.D. developed the methodology; E.E.T., A.V., Y.Z., A.S., T.D., and L.S. performed the formal analysis; E.E.T. wrote the manuscript; E.E.T., A.V., Y.Z., A.S., T.D., and L.S. reviewed and edited the manuscript; E.E.T. supervised the study and acquired funding.

DECLARATION OF INTERESTS

The authors declare no competing interests.

Received: August 10, 2021
Revised: September 6, 2021
Accepted: October 21, 2021
Published: November 18, 2021

REFERENCES

1. Lanfranco, A.R., Castellanos, A.E., Desai, J.P., and Meyers, W.C. (2004). Robotic surgery: a current perspective. Ann. Surg. *239*, 14.
2. Nishida, S.-I., Kawamoto, S., Okawa, Y., Terui, F., and Kitamura, S. (2009). Space debris removal system using a small satellite. Acta Astronautica *65*, 95–102.
3. Lynch, K.P.F. (2017). Modern Robotics (Cambridge University Press).
4. Hagras, H. (2004). A hierarchical type-2 fuzzy logic control architecture for autonomous mobile robots. IEEE Trans. Fuzzy Syst. *12*, 524–539.
5. Grochow, K., Martin, S.L., Hertzmann, A., and Popovic, Z. (2004). Style-Based Inverse Kinematics (ACM SIGGRAPH).
6. A. Csiszar, J. Eilers and A. Verl (2017). On solving the inverse kinematics problem using neural networks. 24th International Conference on Mechatronics and Machine Vision in Practice (M2VIP).
7. Chembuly, V.S., and Voruganti, H.K. (2020). An efficient approach for inverse kinematics and redundancy resolution of spatial redundant robots for cluttered environment. SN Appl. Sci. *2*, 1–20.
8. Tsur, E.E. (2021). Neuromorphic Engineering: The Scientist's, Algorithm Designer's, and Computer Architect's Perspectives on Brain-Inspired Computing (CRC Press).
9. DeWolf, T., Stewart, T.C., Slotine, J.-J., and Eliasmith, C. (2016). A spiking neural model of adaptive arm control. Proc. R. Soc. B: Biol. Sci. *283*, 20162134.
10. Tsur, E.E., and Rivlin-Etzion, M. (2020). Neuromorphic implementation of motion detection using oscillation interference. Neurocomputing *374*, 54–63.
11. Zaidel, Y., Shalumov, A., Volinski, A., Supic, L., and Ezra Tsur, E. (2021). Neuromorphic NEF-based inverse kinematics and PID control. Front. Neurorobotics *15*, 631159.
12. Bekolay, T., Bergstra, J., Hunsberger, E., DeWolf, T., Stewart, T., Rasmussen, D., Choo, X., Voelker, A., and Eliasmith, C. (2014). Nengo: a Python tool for building large-scale functional brain models. Front. Neuroinformatics *7*, 48.
13. Hazan, A., and Tsur, E.E. (2021). Neuromorphic analog implementation of neural engineering framework-inspired spiking neuron for high-dimensional representation. Front. Neurosci. *15*, 627221.
14. Lin, C.-K., Wild, A., Chinya, G., Cao, Y., Davies, M., Lavery, D.M., and Wang, H. (2018). Programming spiking neural networks on intel's loihi. Computer *51*, 52–61.
15. Rasmussen, D. (2019). NengoDL: Combining deep learning and neuromorphic modelling methods. Neuroinformatics *17*, 611–628.
16. DeWolf, T. (2021). Spiking neural networks take control. Sci. Robotics *6*, eabk3268.
17. Almusawi, A.R., Dulger, L.C., and Kapucu, S. (2016). A New Artificial Neural Network Approach in Solving Inverse Kinematics of Robotic Arm (Computational Intelligence and Neuroscience), p. 5720163.
18. Wang, X., Liu, X., Chen, L., and Hu, H. (2021). Deep-learning damped least squares method for inverse kinematics of redundant robots. Measurement *171*, 108821.
19. Duka, A.-V. (2014). Neural network based inverse kinematics solution for trajectory tracking of a robotic arm. Proced. Technol. *12*, 20–27.
20. Li, Yaotong, and Zeng, Nan (1993). A Neural Network Based Inverse Kinematics Solution In Robotics. In Neural Networks in Robotics, *202*, George Bekey and Kenneth Goldberg, eds. (Boston, MA: Springer), https://doi.org/10.1007/978-1-4615-3180-7_6.
21. Dietrich, A., Ott, C., and Albu-Schäffer, A. (2015). An overview of null space projections for redundant, torque-controlled robots. Int. J. Robot. Res. *34*, 1385–1400.
22. Misra, D. (2019). Mish: a self regularized non-monotonic neural activation function. arXiv, 1908.08681.
23. Bing, Z., Meschede, C., Rohrbein, F., Huang, K., and Knoll, A.C. (2018). A survey of robotics control based on learning-inspired spiking neural networks. Front. Neurorobotics *12*, 35.
24. Ranjan, J., Kumar, A., Sigamani, T., and Barnabas, J. (2019). A novel and efficient classifier using spiking neural network. J. Supercomput. 1–16.
25. Hyun, C.M., Baek, S.H., Lee, M., Lee, S.M., and Seo, J.K. (2021). Deep learning-based solvability of underdetermined inverse problems in medical imaging. Med. Image Anal. *69*, 101967.

26. Cardiff, M., and Kitanidis, P.K. (2008). Efficient solution of nonlinear, underdetermined inverse problems with a generalized PDE model. Comput. Geosci. *34*, 1480–1491.

27. Liu, X., Zhou, Q., Birkholzer, J., and Illman, W.A. (2013). Geostatistical reduced-order models in underdetermined inverse problem. Water Resour. Res. *49*, 6587–6600.

28. Aoki, Y., Hayami, K., Sterck, H.D., and Konagaya, A. (2014). Cluster Newton method for sampling multiple solutions of underdetermined inverse problems: application to a parameter identification problem in pharmacokinetics. SIAM J. Sci. Comput. *36*, B14–B44.

29. Gaudreau, P., Hayami, K., Aoki, Y., Safouhi, H., and Konagaya, A. (2015). Improvements to the cluster Newton method for underdetermined inverse problems. J. Comput. Appl. Math. *283*, 122–141.

30. Martin, G., Macdonald, J., and März, M. (2020). Solving inverse problems with deep neural networks–robustness included? arXiv, 04268.

31. Buss, S.R. (2004). Introduction to inverse kinematics with Jacobian transpose, pseudoinverse and damped least squares methods. IEEE J. Robot. Automat. *17*, 16.

32. LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. Nature *521*, 436–444.

33. Martin, A., Paul, B., Jianmin, C., Zhifeng, C., Andy, D., Jeffrey, D., Matthieu, D., Sanjay, G., Geoffrey, I., Michael, I., et al. (2016). Tensorflow: A System for Large-Scale Machine Learnin. Symposium on Operating Systems Design and Implementation.

34. Ramachandran, P., Zoph, B., and Le, Q.V. (2017). Searching for activation functions. arXiv *1710*, 05941.

35. Chen, D., Hu, F., Nian, G., and Yang, T. (2020). Deep residual learning for nonlinear regression. Entropy *22*, 193.

36. Burkitt, A. (2006). A review of the integrate-and-fire neuron model: I. Homogeneous synaptic input. Biol. Cybernetics *95*, 1–19.

37. Eliasmith, C., and Anderson, C.H. (2003). Neural Engineering: Computation, Representation, and Dynamics in Neurobiological Systems (MIT press).

38. Voelker, A.R. (2015). A Solution to the Dynamics of the Prescribed Error Sensitivity Learning Rule (Centre for Theoretical Neuroscience).

39. Hunsberger, E., and Eliasmith, C. (2016). Training spiking deep networks for neuromorphic hardware. arXiv *1611*, 05141.

40. (2021). NVIDIA. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-agx-xavier/.

41. Davies, M., Srinivasa, N., Lin, T.-H., Chinya, G., Cao, Y., Choday, S.H., Dimou, G., Joshi, P., Imam, N., Jain, S., et al. (2018). Loihi: a neuromorphic manycore processor with on-chip learning. IEEE Micro *38*, 82–99.