An Interactive Framework for Visually Realistic 3D Motion Synthesis using Evolutionarily-trained Spiking Neural Networks

IOANNIS POLYKRETIS, Accenture Labs and Rutgers University
ADITI PATIL, Rutgers University
MRIDUL AANJANEYA*, Rutgers University
KONSTANTINOS MICHMIZOS*, Rutgers University

We present an end-to-end method for capturing the dynamics of 3D human characters and translating them for synthesizing new, visually-realistic motion sequences. Conventional methods employ sophisticated, but generic, control approaches for driving the joints of articulated characters, paying little attention to the distinct dynamics of human joint movements. In contrast, our approach attempts to synthesize human-like joint movements by exploiting a biologically-plausible, compact network of spiking neurons that drive joint control in primates and rodents. We adapt the controller architecture by introducing learnable components and propose an evolutionary algorithm for training the spiking neural network architectures and capturing diverse joint dynamics. Our method requires only a few samples for capturing the dynamic properties of a joint's motion and exploits the biologically-inspired, trained controller for its reconstruction. More importantly, it can transfer the captured dynamics to new visually-plausible motion sequences. To enable user-dependent tailoring of the resulting motion sequences, we develop an interactive framework that allows for editing and real-time visualization of the controlled 3D character. We also demonstrate the applicability of our method to real human motion capture data by learning the hand joint dynamics from a gesture dataset and using our framework to reconstruct the gestures with our 3D animated character. The compact architecture of our joint controller emerging from its biologically-realistic design, and the inherent capacity of our evolutionary learning algorithm for parallelization, suggest that our approach could provide an efficient and scalable alternative for synthesizing 3D character animations with diverse and visually-realistic motion dynamics.

CCS Concepts: • **Computing methodologies** → *Computer graphics*; Physical simulation.

Additional Key Words and Phrases: motion synthesis, spiking neural networks, evolutionary learning

ACM Reference Format:
Ioannis Polykretis, Aditi Patil, Mridul Aanjaneya, and Konstantinos Michmizos. 2023. An Interactive Framework for Visually Realistic 3D Motion Synthesis using Evolutionarily-trained Spiking Neural Networks. *Proc. ACM Comput. Graph. Interact. Tech.* 6, 1 (May 2023), 19 pages. https://doi.org/10.1145/3585509

1 INTRODUCTION

The synthesis of realistic motion for 3D human characters is a challenging problem that has attracted much interest over the years [Arikan et al. 2003; Fang and Pollard 2003; Lee et al. 2002; Li et al. 2002; Ren et al. 2005]. The difficulties arise due to the human skeleton's intricate structure, which

*Joint supervising authors.

Authors' addresses: Ioannis Polykretis, Accenture Labs and Rutgers University; Aditi Patil, Rutgers University; Mridul Aanjaneya, Rutgers University; Konstantinos Michmizos, Rutgers University.

2577-6193/2023/5-ART $15.00
https://doi.org/10.1145/3585509

must be taken into account with all its details for the motion to look realistic [Popović and Witkin 1999]. Despite being underactuated, the human musculoskeletal system introduces many degrees of freedom (DOF), whose motion appears realistic when consistent with the laws of physics.

Forward dynamics methods have targeted this equivalence of physical and visual realism and have been widely deployed for generating physics-constrained motions of objects [Arnaldi et al. 1989; Park and Fussell 1997; Ramakrishnananda and Wong 1999; Wu and Popović 2003]. Such approaches demonstrate remarkable performance in the motion animation of rigid bodies [Baraff 1990, 1994] and textured surfaces (e.g., cloth) [Baraff and Witkin 1998; DeRose et al. 1998]. However, the complexity of physics-based models increases dramatically as the number of DOFs scales up [Popović and Witkin 1999], limiting their applicability for animating more detailed characters.

Capturing human motion data with sensors and using them as a reference can simplify the problem. Effective sampling methods have been proposed for discretizing the analog sensor signals, reducing their dimensions, and overcoming their inherent noise [Herrmann et al. 2019; Pullen and Bregler 2002; Rajamäki and Hämäläinen 2017]. Then, the samples can be used to reconstruct the original motions in animation. The reconstruction methods range from the use of simple cubic spline interpolation [Brotman and Netravali 1988; Liu and McMillan 2006] to the deployment of dedicated proportional-derivative (PD) controllers [Liu et al. 2015]. Statistical Machine Learning models [Chai and Hodgins 2007; Min and Chai 2012] have been proposed to automate and improve the in-betweening of motion key-frames. With the recent advances in deep learning, this trajectory generation task is addressed by sophisticated tools such as adversarial recurrent neural networks [Harvey et al. 2020] or diffusion models [Tevet et al. 2022]. While such methods effectively reconstruct the original motions, they separate the samples from the musculoskeletal dynamics that generate the continuous motions. This separation can degrade the realism of the reconstruction when the methods are required to generalize to new (and previously unseen) motion sequences.

In this work, we attempt to bridge this gap between the samples and the continuous motion signals by proposing a method that combines the sample interpolation with accurate reconstruction of the intermediate trajectories, by exploiting the underlying musculoskeletal dynamics that generate the motion sequences. We draw inspiration from the spiking neural networks (SNNs) that provide joint control in primates and rodents [Fink et al. 2014; Polykretis et al. 2023; Seki et al. 2003] and formalize their architecture in a context-free grammar, which encapsulates different network variants. We then propose an evolutionary algorithm that allows the joint controller to learn the network architecture that best approximates the motion dynamics of the joint. We use this learnable joint controller as a building block and scale it up to capture the synthetic motion of a multi-DOF 3D humanoid character. We demonstrate how our controller can reconstruct the original movement and transfer the learned dynamics to unknown, visually-realistic motion sequences. We also verify the ability of our method to capture human motion dynamics by applying it to a dataset consisting of human gesture recordings, which we reconstruct in animation. Moreover, we develop an interactive framework for modifying the driving controller and visualizing the resulting motion in real-time.

In summary, our main contributions are as follows:

- The formalization of complicated, biologically inspired spiking neural network (SNN) architectures in a concise context-free grammar,
- A novel evolutionary training algorithm that allows the SNN controller for each joint to capture and learn the dynamics of its corresponding movement,
- An end-to-end method for inverse motion synthesis that captures the motion dynamics of a 3D character and uses them to reconstruct known motions and generalize to unknown ones,

- An interactive framework to modify the learned motion dynamics and visualize the effects of the modifications in real-time.

Our bioinspired method captures the motion dynamics accurately and allows for effective visually-plausible reconstructions. More importantly, it generalizes to realistic motion sequences that follow the captured dynamics and allows for interactively modifying the dynamics in a user-dependent fashion without sacrificing the motion realism.

2 RELATED WORK

The animation of 3D human characters has been of interest to the graphics community for many decades [Sturman 1994]. The spectrum of animation detail and applications is vast: from simplified humanoid silhouettes performing physically valid human motions (e.g., jumps, flips, gymnastic exercises) [Fang and Pollard 2003] and choreographed dancing sequences [Li et al. 2002] to detailed avatars performing human manipulation tasks [Yamane et al. 2004] and forming crowds to interact with each other in dynamic environments [Lau and Kuffner 2005]. The ever-growing need for realistic human character animation in movies, video games, and even the metaverse has led to the development of a multitude of different solutions.

A prominent approach due to its remarkable success in diverse tasks has been Reinforcement Learning (RL). For the continuous action space of character animation, RL mainly relies on policy gradient [Silver et al. 2014] or continuous actor-critic learning [Van Hasselt and Wiering 2007]. Combined with recent advances in deep learning [Arulkumaran et al. 2017], deep RL provides remarkably elegant behaviors in animated characters [Liu and Hodgins 2018; Peng et al. 2018]. The requirement of these methods for stabilizing techniques [Mnih et al. 2015], but most importantly, their long training times and high computational resource requirements, have forced researchers to explore alternative solutions.

In contrast to the task-agnostic approach of RL, physics-based models benefit from our knowledge of the constraints that the laws of physics impose on the motion of objects in space [Arnaldi et al. 1989; Park and Fussell 1997; Ramakrishnananda and Wong 1999; Wu and Popović 2003]. Forward dynamics models achieve aesthetically pleasing animations of characters exhibiting diverse behaviors [Geijtenbeek and Pronost 2012; Popović and Witkin 1999] by exploiting the equivalence of physical and visual realism. The increase in the DOF of the character to be visualized challenges such methods since determining the required forces for a large number of objects becomes extremely difficult [Popović and Witkin 1999]. Spacetime constraint methods [Cohen 1992; Rose et al. 1996; Witkin and Kass 1988] attempt to simplify the problem by introducing animator-defined pose constraints (e.g., the initial, final, and possibly intermediate poses) that the motion sequence needs to satisfy. Beyond the sensitivity of the methods to the selection of the pose constraints [Popović and Witkin 1999], defining the intermediate poses by hand becomes increasingly tedious with the increasing complexity of the motion sequences.

A solution to the definition of intermediate pose constraints comes from human motion capture (MoCap) data, which provides a continuum of intermediate poses for the motions to be performed. Sampling methods [Herrmann et al. 2019; Liu et al. 2015, 2010; Pullen and Bregler 2002; Rajamäki and Hämäläinen 2017] optimize the selection of keyframes, which then can be combined to reconstruct the motion in animation. Past approaches have used methods of increasing complexity and elegance to gel the keyframes into a continuous movement: While even cubic spline interpolation provides reasonable results [Brotman and Netravali 1988; Liu and McMillan 2006], others have used dedicated proportional-derivative (PD) controllers to control the animated character [Liu et al. 2015]. This introduction of additional tunable parameters for the reconstruction has led to simplification attempts [Herrmann et al. 2019; Rajamäki and Hämäläinen 2017], which exploit the transformation

of the samples into the angular velocity representation space. However, the reconstruction methods disregard the musculoskeletal dynamics that gave rise to the captured continuous signals, limiting the visual realism for reconstructions of unknown motion sequences.

Our work attempts to capture these musculoskeletal dynamics of human joints and use them to not only reconstruct the recorded motions but also generalize to visually-realistic animations of unknown sequences. For this, we use a simple sampling method to identify keyframes for the reconstruction, but focus on developing a bioinspired method to learn the dynamics of the continuous MoCap data and allow for their realistic combination into motion sequences.

3 TECHNICAL APPROACH

3.1 Background

While task-agnostic methods such as PID control or RL perform remarkably in a wide range of tasks, biological agents can provide valuable inspiration for visually realistic animation of articulated human characters. Interestingly, biological agents such as primates and rodents employ dedicated neuronal structures to control their joints [Fink et al. 2014; Seki et al. 2003]. Neuroscientific evidence has characterized these neuronal subnetworks and abstracted them into high-level control modules [Fink et al. 2014]. Recently, computational modeling works have attempted to simulate the dynamics of these control modules with networks of spiking neurons to endow robotic joints with human-like control [Polykretis et al. 2023].

3.2 A Brief Primer on Spiking Neural Networks (SNNs)

Unlike neurons in artificial neural networks (ANNs) that sum continuous-valued inputs, apply threshold functions, and provide continuous-valued outputs [Bishop 2006], spiking neuron models attempt to better approximate biological neurons. Specifically, they first integrate their synaptic inputs in a state variable representing the continuous neuronal membrane voltage. When this voltage exceeds a threshold value, the neurons emit a binary, all-or-none output called a spike and reset their membrane voltage. Similar to ANNs, spiking neurons connect through weighted synapses. While only learning modifies synaptic weights in ANNs, in SNNs, weights can change transiently in response to different factors. Increasing spiking activity of the presynaptic neurons may either increase or decrease the synaptic weights until saturation, a phenomenon known as synaptic facilitation or depression [Tsodyks et al. 1998]. The synaptic weights can also transiently change in response to the activity of other modulatory neurons [Fink et al. 2014]. These complicated dynamics are then combined with intricate architectures to control high-level behaviors.

$S = \{Controller\}$	$T = \{eN, Syn, Fac, Adapt, MN\}$
$1) Controller \rightarrow Inp - Net - Out$	$7) StaticFfPath \rightarrow ECM - Con$
$2) Net \rightarrow Bus // Bus$	$8) AdaptFfPath \rightarrow (ECM - Con) // Adapt$
$3) Bus \rightarrow PathEns - MNpool$	$9) FbPath \rightarrow ECM - Syn$
$4) PathEns \rightarrow FfPath \mid FfPath // FbPathEns$	$10) MNpool \rightarrow MN \mid MN // MNpool$
$5) FfPath \rightarrow StaticFfPath \mid AdaptFfPath$	$11) ECM \rightarrow eN$
$6) FbPathEns \rightarrow FbPath \mid FbPath // FbPathEns$	$12) Con \rightarrow Syn \mid Syn - Fac$

Fig. 1. Our Context-free Grammar for automatically generating SNNs with desired properties.

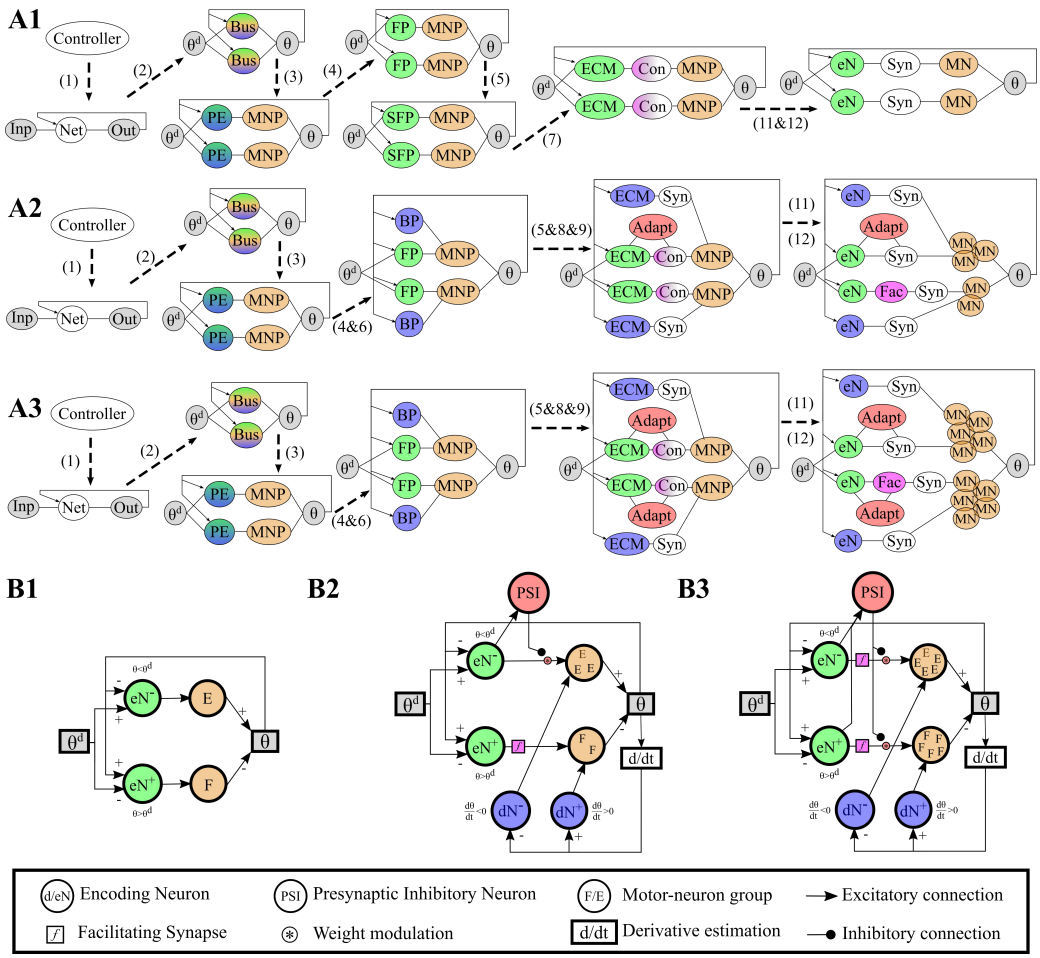

Fig. 2. **A.** Exemplary derivations of three controllers as words of the language (B1-shortest, B2-intermediate, B3-longest word). Some symbols of the grammar are abbreviated as follows: FP-*FfPath*, PE-*PathEns*, MNP-*MNpool*, FP-*FfPath*, SFP-*StaticFfPath*, BP-*FbPath*. **B.** SNN implementations of the corresponding controller architectures, which can be obtained following the procedure described in Section 3.4.

3.2.1 Notational Terminology. In this section, we introduce some notations for various features in SNNs that we will subsequently use when describing our context-free grammar for generating SNNs in Section 3.3. We encourage the reader to refer to Figure 2 when reading our notation.

Motor control networks (*Net*) are recurrent architectures. They receive external inputs (*Inp*) and propagate them through processing alleys (*Buses*) in two steps. The first step utilizes ensembles of feedforward pathways (*FfPathEns*) to reach the output layer of the *Net*. This layer consists of dedicated cells called motor neurons (*MN*). Groups of *MN* (*MNpools*) modify the controlled variables (e.g., muscle length or joint angle) that constitute the network's outputs (*Out*). In the second step of the propagation, estimates of these outputs loop back into the *Net* using dedicated ensembles of feedback pathways (*FbPathEns*). Both *Ffpath* and *FbPath* are essentially processing modules (*ECM*), i.e. neurons (*eN*), that form connections (*Con*) to the *MNpools*. These connections may be synapses (*Syn*) with static or dynamic weights. Dynamic weights may transiently fluctuate

due to synaptic facilitation (*Fac*) or depression. They may also fluctuate in response to the activity of parallel modulatory pathways (*Adapt*).

While the structure and properties of the individual components are understood, their networks introduce challenges. The inverse problem of deciphering how such networks give rise to high-level behaviors is challenging, even for neuroscience experts that slowly characterize such architectures [Fink et al. 2014; Seki et al. 2003]. But even the forward problem of designing SNNs for motion control that are not tailored to a single set of motion dynamics is not straightforward.

In the following sections, we aim to formulate the properties of such networks in a structured way and allow for capturing the dynamic properties of different joint movements. To achieve this, we first define a context-free grammar (see Section 3.3) that produces a language for generating SNNs. Then, we propose an evolutionary algorithm (see Section 3.6) that uses the grammar as a genetic code to produce controller mutations that best capture the dynamics of a joint.

3.3 A Context-free Grammar for Automatically Generating SNN Controllers

Our context-free grammar for generating SNN controllers is shown in Figure 1. The goal of the grammar is to define controllers that follow the general structure of motor control networks described in Section 3.2.1, while they provide different control dynamics. For this, the first group of rules (*Structure Rules*) in our grammar ensures that the words of the language follow the overall architecture. Then, the second group of rules (*Dynamics Rules*) introduce different control dynamics to capture diverse motion sequences. We explain the rule definition in more detail below.

3.3.1 Structure Rules. Rule 1 ensures that the *Net* follows the recurrent architecture of motor control networks that compares the input, desired value, and the fed-back actual value of the controlled variable. Rules 2 and 3 are also restrictive to ensure that the *Net* consists of the two parallel *Buses* that allow for bidirectional modifications of the controlled variable through processing pathways (*PathEns*) and actuation components (*MNpool*).

3.3.2 Dynamics Rules. The remaining rules are more flexible and introduce variability in the architecture and diversity in the network dynamics. For example, when applied, rules 4 and 6 enrich the processing capabilities of the *Net* by introducing one or more pathways to feed properties of the controlled variable back into the *Net*. For simplification purposes, we limited the number of feedback pathways to one to account for the derivative of the controlled variable. However, this restriction could be lifted to include higher moments. Several rules (7, 8, 9, 12) are defined to allow for static or dynamic connections in the *Net*. Again for simplicity, we restricted dynamic connections only to the feed-forward pathways (rules 5 and 9), a design choice that could easily be extended. Lastly, we introduced rule 10 to allow for different strengths of the actuation components (*MNpool*) corresponding to different muscle sizes and innervation levels. While we limited our grammar to a few discrete sizes, our design could be extended to a continuous size spectrum. More importantly, the neuron population approach could be expanded beyond the actuation to the processing neurons (*ECM*), whose number we here limited to one (rule 11).

We illustrate the ability of our grammar to produce controller architectures with different properties through a series of derivations shown in Figure 2. The response of the derived SNN controllers to a step function is shown in

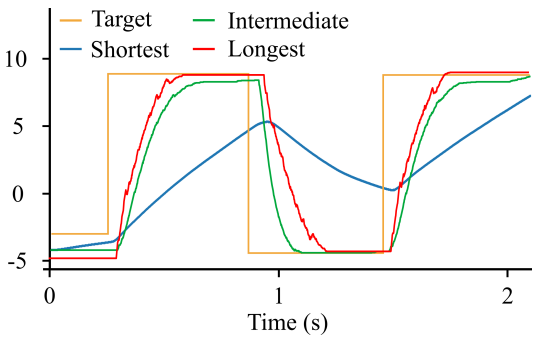

Fig. 3. Response of the three SNN controllers derived from our grammar in Figure 2 to a step function.

Figure 3. Note that even though the response of the longest "word" (i.e., SNN controller) derived from our grammar looks closest to the step function, one should not naïvely conclude that the longest word is always better for every given function. We have empirically verified this claim through our experiments on the hand gesture MoCap data set, described in Sections 3.7 and 4.6.

First, we present the derivation of the controller that corresponds to the shortest word in our language (Fig. 2, A1). This controller's external input *Inp* is the desired joint angle θ^d, while its output is the controlled joint angle θ. *Inp* propagates through the *Net* to the *Out* through two parallel *Buses* (step 2). Our grammar always produces two parallel *Buses*, one for increasing the controlled variable and one for decreasing it. An estimate of *Out* was fed back to the *Net* to allow error computation and correction. In general, the *Buses* could consist of a more complex *PathEns* connected to the output layer of *MNpool* (step 3). However, for the shortest word of the language, they were substituted with only one *FfPath* (step 4). For similar brevity reasons, each *FfPath* was substituted with its shortest static version *StaticFfPath* (step 5). For the substitution of each *StaticPath*, there was only one choice resulting in an *eN* and its corresponding *Con* to the *MNpool* (step 6). However, both the *Con* and the *MNpool* were substituted with their shortest possible versions: a static *Syn* and a single *MN*, respectively (step 7). Functionally, this SNN controller was equivalent to a conventional proportional (P) controller. The two *eN* received input and feedback with opposite signs. This forced them to fire in a mutually exclusive fashion. The upper one encoded the presence of a negative ($\theta < \theta^d$), while the lower one that of a positive ($\theta > \theta^d$) error. Their firing rate was proportional to the magnitude of the error and drove the corresponding MN to modify the controlled variable and correct the error.

3.4 Translating the Controller Architecture to Spiking Neural Networks

To translate the abstract architectures that our grammar produces into SNN formulation, we followed the approach and the parameter definitions presented in detail in [Polykretis et al. 2023]. In this section, we briefly describe the main components of the SNN controllers shown in Fig. 1, C.

We simulated all neurons on the feedforward (*eN*) and feedback (*dN*) pathways, and the MN pools (*E*/*F*) using the Leaky-Integrate-and-Fire (LIF) neuron model, which is defined as follows:

$$u_i^{(t)} = u_i^{(t-1)} \cdot d_u + \sum_j w_{ij} \cdot s_j^{(t-1)}, \quad \text{and} \quad v_i^{(t)} = v_i^{(t-1)} \cdot d_v + u_i^{(t)}, \tag{1}$$

where t is the time step, u_i is the neuron's input current, v_i is the neuron's voltage, d_u and d_v are current and voltage decay factors, w_{ij} are the connection weights between the presynaptic neuron j and the postsynaptic neuron i, and $s_j^{(t-1)}$ is a binary variable denoting the spikes of neuron j at time step $t-1$.

Following [Polykretis et al. 2023], we set the static values of all the synaptic weights between neurons to 1, except for the connections to the MN pools. For these connections, the presynaptic neurons were fully connected to the pool of MN, and the weights followed a uniform distribution $U(0,1)$. We introduced synaptic dynamics to the static weights through two time-dependent, multiplicative factors $f^{(t)}$ and $g^{(t)}$ that adapted the weights W between the *eN* and the motor neurons, as shown below:

$$w^{(t)} = W \cdot f^{(t)} \cdot g^{(t)}, \tag{2}$$

The first factor $f^{(t)}$ corresponded to the presynaptic-dependent facilitation of the weights [Tsodyks et al. 1998] to enforce the gradual transfer of the presynaptic activity to the motor neurons. The efficacy $f_{ij}^{(t)}$ of the synaptic connection between the j^{th} *ePPC* and the i^{th} motor neuron increased with each presynaptic spike until saturating at a maximum value and decayed otherwise with a factor d_{fac}, as shown in equation (3) below:

$$f_{ij}^{(t)} = f_{ij}^{(t-1)} \cdot d_{fac} + U_{fac} \cdot s_j^{(t-1)}, \tag{3}$$

where d_{fac} is the facilitation decay time constant, U_{fac} is the facilitation factor increment, and $s_j^{(t-1)}$ is the spike of the presynaptic neuron. When the synaptic efficacies are initially low, abrupt increases in the activity of the eN do not directly affect the motor neurons. As they slowly increase, they gradually change the activity of the motor neurons and, in turn, the controlled amount.

The second factor $g^{(t)}$ ensured the accurate control of both small and large-range movements by allowing for weight adaptation to the movement magnitude. This function is based on presynaptic inhibition, an experimentally identified biological mechanism for adaptation [Fink et al. 2014]. For this, we utilized an additional spiking neuron (PSI), whose activity reflected the activity of the eN and, therefore, the magnitude of the error. The spikes of the PSI modulated the connection weights through a factor g as described below:

$$g^{(t)} = g_{max} - \left[g^{(t-1)} \cdot d_{PSI} + U_{PSI} \cdot s_{PSI}^{(t-1)} \right], \tag{4}$$

where g_{max} is the maximum value of the modulatory factor g, d_{fac} is the presynaptic inhibition decay time constant, U_{PSI} is the presynaptic inhibition increment, and $s_{PSI}^{(t-1)}$ represents the spike of the PSI neuron. During the control, g decreased by U_{PSI} with each PSI spike $s_{PSI}^{(t-1)}$ and otherwise decayed to its maximum value g_{max} with a factor d_{PSI}. Using g as a multiplicative factor in Eq. 2, we emulated the divisive effect of presynaptic inhibition on the weights of biological synapses [Fink et al. 2014].

Algorithm 1 Sampling

for all joints **do**
  $S = \{0, T\}$
  $G = \{\theta_0, \theta_T\}$
  $\dot{\theta} \leftarrow Grad(\theta)$
  $z = arg\{\dot{\theta}_i \dot{\theta}_{i-1} < 0\}$
  $S \leftarrow S \cup \{z\}$
  t = 0
  while $t \in S$ **and** t ≠ T **do**
    $G \leftarrow G \cup \{\theta_{t+1}\}$
    $t \leftarrow t + 1$
  end while
  return S, G
end for

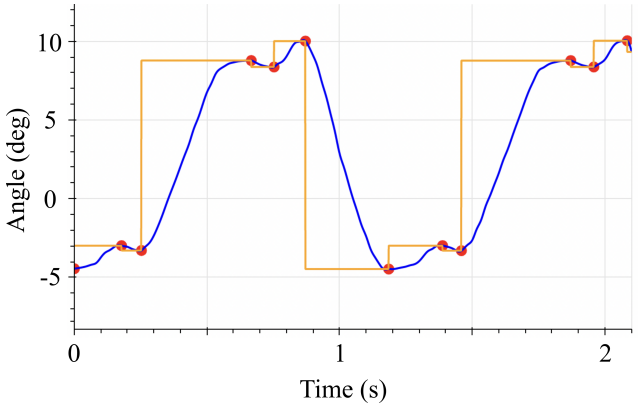

Fig. 4. Application of Algorithm 1 to discretize a given joint angle (blue) into target angles (yellow) that are provided as inputs to the joint controller. At each trigger point (red), where the sign of the angle's derivative changes, the target angle is updated to the value of the joint angle at the next trigger point.

3.5 Input Generation for the Joint Controller

To use our SNN architectures for joint control, we need to provide the desired/target angle (θ^d) to the SNN and let the network minimize the error between that and the actual value of the controlled variable. Our method requires only a few time-target pairs (s, g) (Fig. 4, red dots) to delineate the desired trajectory of a joint (Fig. 4, blue) over time T. Then, the biologically plausible architecture and dynamics of the SNN controller can effectively approximate the intermediate parts of the trajectory, resulting in visually realistic joint motions for 3D human characters. To specify these pairs (s, g), we used a simple algorithm (Algorithm 1), which is similar to classical

methods for sampling points of interest from continuous motion sequences [Pullen and Bregler 2002]. Specifically, two pairs were selected for each joint by default (steps 2-3). For these, s was selected as the start ($t = 0$) and end ($t = T$) of the motion, while g was given by the value of the desired angle at the respective time instances (θ_0) and end (θ_T). The s for the intermediate points was selected as the time instances when the sign of the desired joint angle's derivative changed (steps 5-6). The g-value of each intermediate point was set to the value of the desired angle at the next point of interest (steps 8-11). For all other time steps that were not points of interest, we set the value for the desired angle to the value at the next point of interest and obtained the discretized angle values (Fig. 4, yellow) that we provided as input to our joint controller.

3.6 Evolutionary Algorithm for Capturing Joint Dynamics with SNNs

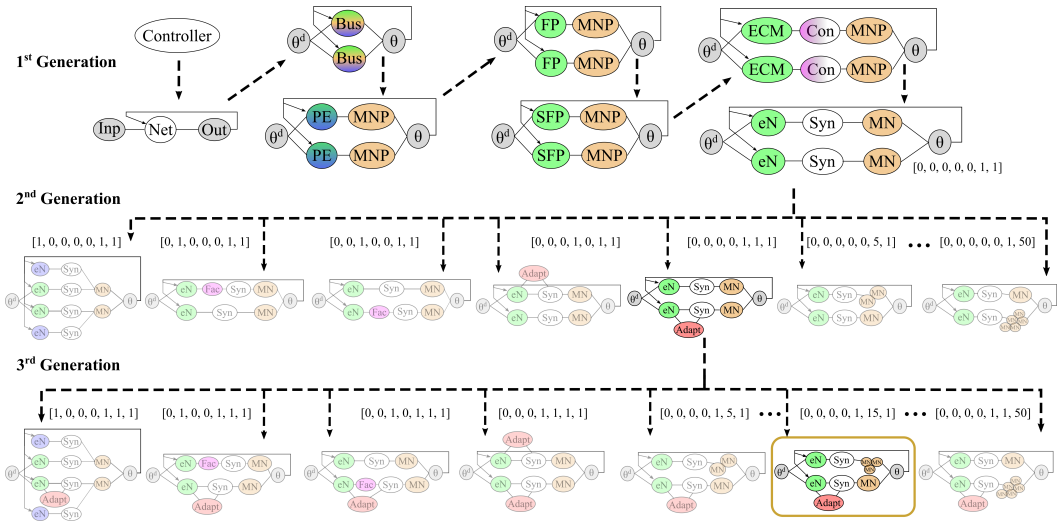

Fig. 5. **Visualization of the genetic algorithm (Algorithm 2) for selecting the best-performing controller architecture.** The search starts from the shortest word produced by our context-free grammar that constitutes the first generation. The second generation of controllers results from all possible mutations of the shortest word. Then, the best-performing controller of the first two generations is selected as the parent of the third generation. A final architecture is selected (golden square) when either its performance exceeds a predefined threshold or a number of architectures have been examined. For illustration purposes, we only show three generations, we don't show all the possible variations of the architecture, and we select a controller from the most recent generation as the best-performing controller.

To effectively capture the dynamics of different joints, we need to identify which components from our grammar allow the controller to closely approximate the given joint angles. We drew inspiration from the classical A* algorithm to develop an evolutionary algorithm for SNNs [Schmidgall et al. 2022] (Algorithm 2) to select the best-performing architecture among generations of controller words with different components and, hence, different dynamics. A simplified visualization of the algorithm's execution is shown in Figure 5.

To formalize the representation of the different controller architectures, we abstracted them into 7-dimensional vectors. The first five binary-valued dimensions represent the presence (1) or absence (0) of a specific component: (i) speed control through the dedicated feedback pathway, (ii-iii) facilitation in each feedforward pathway, and (iv-v) range adaptation in each feedforward pathway. The last two dimensions can take values in a given range $\{1, 5, 10, 15, 20, 35, 50\}$ and represent the

number of neurons in the MN pool of each feedforward pathway. Given this formulation, the three controllers shown in Fig. 1, C are represented by $[0, 0, 0, 0, 0, 1, 1]$, $[1, 0, 1, 1, 0, 15, 15]$, and $[1, 1, 1, 1, 1, 50, 50]$ respectively.

For each joint, the first generation of controllers (step 3) consists of only the SNN whose architecture corresponds to the shortest word in the language (Fig. 1, C1 - $[0, 0, 0, 0, 0, 1, 1]$). We evaluate the controller's performance by estimating the error between the actual (θ) and the controlled (θ^c) values of the joint angle, normalized over the range of the actual angle, as shown below:

$$ERR(\theta, \theta_c) = \frac{\sum_{i=0}^{T} \sqrt{(\theta_i - \theta_i^c)^2}}{max\{\theta\} - min\{\theta\}} \tag{5}$$

If the error of the given controller exceeds a predefined threshold (step 5), the algorithm proceeds to the next options. This step is similar to that of A* where the heuristic is evaluated at the closest neighbors of the current vertex. At this step, our algorithm considers the next generation of controllers, which are all possible mutations of the parent controller (step 12). As a mutation, we define a single allowed value change in the representation vector of the parent controller. As a result, each parent controller can mutate into 180 different offsprings (5 possible binary mutations x 6^2 MN pool size mutations). Algorithm 2 goes through the new controller generation (step 13) and evaluates their performance using Equation (5) (step 17). Simultaneously, it inserts the evaluated offsprings and their respective errors in a record sorted by ascending error ($\mapsto$ denotes insertion to a sorted list) to avoid repeated future evaluations (steps 17-18). The best-performing offspring, whose offsprings have not been evaluated (found always in the first entry of the sorted record), is selected as the parent of the next generation of controllers (steps 6-11). This step corresponds to the optimality-preserving step of A*, where the exploration proceeds through the vertex with the minimal cost. We stop searching for the best-performing controller when the dynamics are captured appropriately, or we have tested a predefined number of controllers (step 5). Again, this corresponds to A* stopping when either the goal vertex has been reached or a number of edge transitions has been exceeded.

In this way, our algorithm trains the controller architecture on different joint dynamics. For example, joints with gradual movement initiation and gradual deceleration towards the target would benefit from synaptic facilitation, while joints that had to achieve both small and large-range movements would benefit from the PSI-induced range adaptation. At the same time, joints with small or slow movements would only employ a few MN, while large and fast movements would require larger populations, as observed in muscle fiber innervation of humans with age [Luff 1998] or strength training [Wilson et al. 2012].

3.7 Data Processing of Motion Sequences

We used two types of motion sequences to examine our controller's ability to capture their dynamics: synthetic and real human motion data. As synthetic motion data, we used the activity recordings of the publicly available, pre-trained 3D humanoid[1] exhibiting different behaviors (i.e., walking, running, and their "sad" and "sneaky" variations). The motion sequences consist of 67 humanoid bone rotations around three axes over time (201 DOF in total). For the real-world data, we used a human MoCap dataset that consists of recordings of human subjects performing hand gestures while wearing a glove with recording sensors [Jarque-Bou et al. 2020]. The motion gestures were repeated several times by each subject and captured by the sensor glove. Then, the sensor readouts were calibrated as described in [Jarque-Bou et al. 2020] to obtain joint angle estimates for 22 joints.

[1] https://threejs.org/examples/#webgl_animation_skinning_additive_blending

Algorithm 2 Evolutionary training of the SNN joint controller

Set ϵ; $counter\_limit$
for all joints **do**
  $counter \leftarrow 0$; $k \leftarrow [0,0,0,0,0,1,1]$; $parents \leftarrow \{\}$;
  $offspring\_list \leftarrow \{k\}$; $offspring\_heap \leftarrow \{k\}$; $offspring\_err \leftarrow \{ERR(\theta, \theta_c(k))\}$
  while $min\{offspring\_err\} > \epsilon$ **and** $counter < counter\_limit$ **do**
    for $\ell \in offspring\_list$ **do**
      if $\ell \notin parents$ **then**
        $parents \leftarrow parents \cup \{\ell\}$
        break
      end if
    end for
    $offspring\_heap \leftarrow offspring\_heap \cup MUTATE(\ell)$
    while $|offspring\_heap| > 0$ **do**
      $x \leftarrow POP(offspring\_heap)$
      if $k \notin offspring\_list$ **then**
        $counter \leftarrow counter + 1$
        $offspring\_err \leftarrow ERR(\theta, \theta_c(x))$
        $offspring\_list \leftarrow x$
      end if
    end while
  end while
  return $offspring\_list(argmin(offspring\_cost))$
end for

To obtain the joint angles of a representative hand gesture, we first averaged the repetitions of a gesture by each subject and then calculated the mean across different subjects.

4 RESULTS

To evaluate the applicability and performance of our approach, we attempted to approximate both synthetic joint dynamics of pre-trained 3D animated characters and real-world joint dynamics recorded in a human motion capture (MoCap) dataset. We first trained two controllers and used them to reconstruct two behaviors of the animated character with drastically different motion speeds and ranges. Additionally, we examined how our trained controllers transferred the learned dynamics to motions with similar or different speeds and ranges. Then, we trained our controllers on the joint dynamics of a human hand and reconstructed the gestures recorded in the MoCap dataset with our 3D animated character. Finally, we demonstrate the interactive functionalities of our framework, allowing for the user-dependent modification of the control dynamics.

4.1 Comparison to related methods

We first evaluated our method against a simple alternative that has been used for motion sequence reconstruction [Pullen and Bregler 2002]. To do so, we compared the dynamics generated by our controllers with those generated by cubic polynomial interpolation, as shown in Figure 6.

Due to the properties of cubic interpolation, the reconstruction resulted in a smooth motion that was accurate at the sampled keyframes. However, the fixed dynamics of the cubic polynomials deteriorated the reconstruction of the motion dynamics between samples, which lagged behind

the original movement. In contrast, our method did not presuppose any fixed dynamics and the grammar gave rise to multiple controllers, ranging from its shortest to its longest words. The best-performing controller generated by our grammar performed better than the cubic interpolation in capturing the motion dynamics between consecutive samples, avoiding the lag. As a result, the diverse dynamics provided by our grammar and the selection of the best-performing controller by our evolutionary algorithm can indeed improve the motion reconstructions.

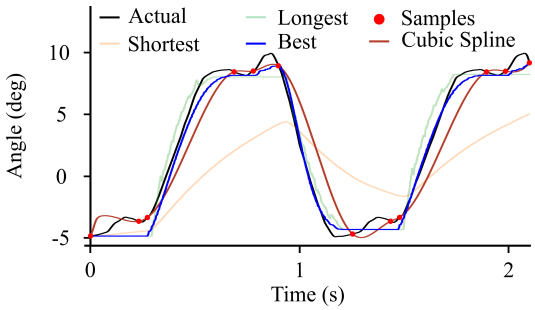

Fig. 6. Comparison of the controllers generated by our method against interpolation with cubic polynomials.

4.2 Reconstruction of Captured Dynamics

To examine how our method can capture motion dynamics with different speeds and ranges, we used our approach to train controllers for two behaviors with distinct dynamics: a slower and smaller-range walking motion and a faster and larger-range running motion, as shown in Figure 7. Our training algorithm successfully generated controller architectures that recreated the reference joint motions (Fig. 7, B). Although the resulting joint motions were not identical to the reference ones, the overall behavior of the animated character closely resembled the reference and appeared visually realistic for both walking and running (Fig. 7, A). This example shows that our method can generate controller architectures that do not simply learn the given behaviors but capture the motion dynamics of the character joints that give rise to the behaviors.

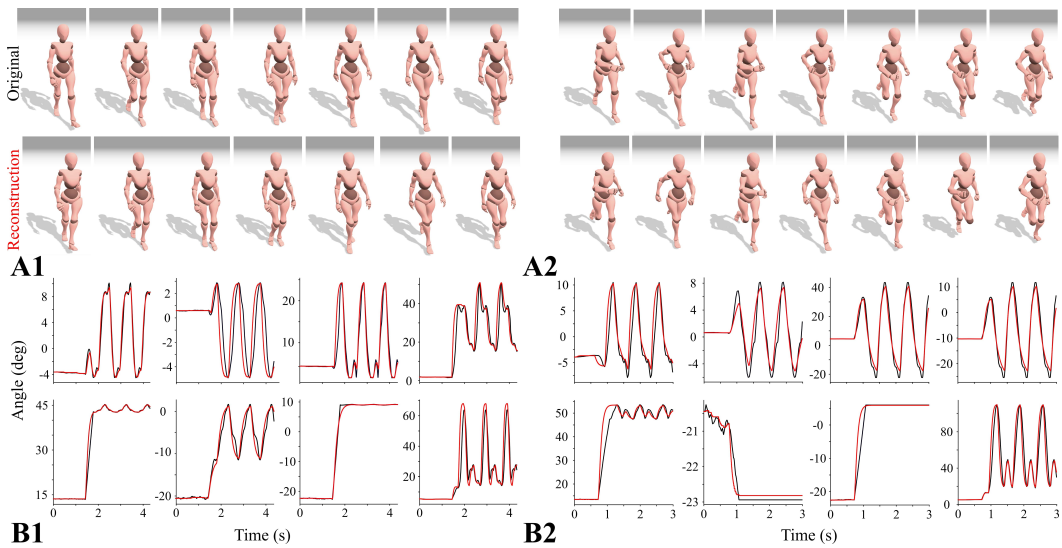

Fig. 7. **Reconstruction of motions with captured dynamics. A.** Visualization of a walking (A1) and a running (A2) motion and their reconstructions based on the captured joint dynamics. **B.** Exemplary joint angle recordings and their reconstructions.

4.3 Generalization to Similar Dynamics

After capturing the underlying motion dynamics, we wondered whether our controllers could transfer them to different behaviors requiring motions of similar speed and range. For this, we used our controller trained on walking (Section 4.2) to drive two variants of the walking behavior: "sad" and "sneaky" walking (Fig. 8). When required to drive these two variants of walking behavior, our controller accurately recreated the reference joint motions (Fig. 8, B) and gave rise to behaviors that closely resembled the reference motion sequences, although not explicitly trained on the dynamics of these motions. Our results suggest that our trained controllers can exploit their final architecture to generalize to previously unseen motion sequences with similar dynamics. As a reminder, our approach does require the trigger points to be re-computed for new motion sequences and provided as input to our SNN controllers.

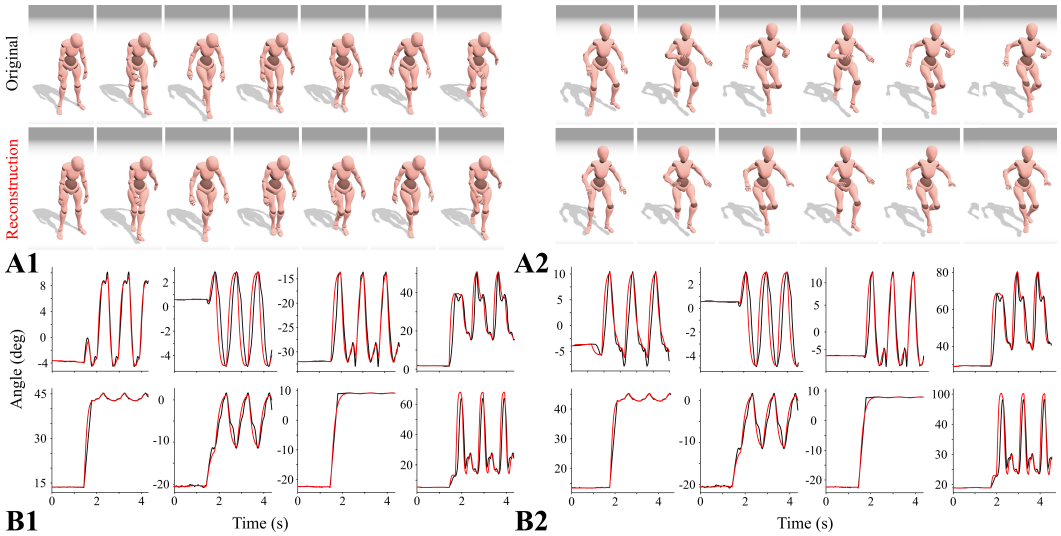

Fig. 8. **Reconstruction of motions with dynamics similar to the captured ones. A.** Visualization of a sad walking (A1) and a sneaky walking (A2) motion and their reconstructions based on the joint dynamics captured during regular walking. **B.** Exemplary joint angle recordings and their reconstructions.

4.4 Generalization to Different Dynamics

After the successful generalization of the trained controllers to motions with similar dynamics, we went one step further to examine whether our controllers could also generalize to previously unseen motion sequences with different speeds and ranges.

For this task, we first attempted to drive the running motion using the controller trained on walking (Section 4.2) to control the running motion (Fig. 9, A1-B1). While appearing visually realistic, the resulting behavior resembled “jogging” with a smaller range of joint movements, similar to the high-effort run of an older or unfit person (Fig. 9, A1). Looking at the individual joint movements of the character (Fig. 9, B1), we observed that for some joints, the reconstructed motions could not cover the full range of the reference ones. Even the joints that moved across the required range consistently lagged behind the respective recordings, suggesting that the walking controller was too weak to recreate the running movement accurately.

The inability of the walking controller to effectively drive the more intense running motion made us wonder whether the controllers of physically intense behaviors could effectively reproduce

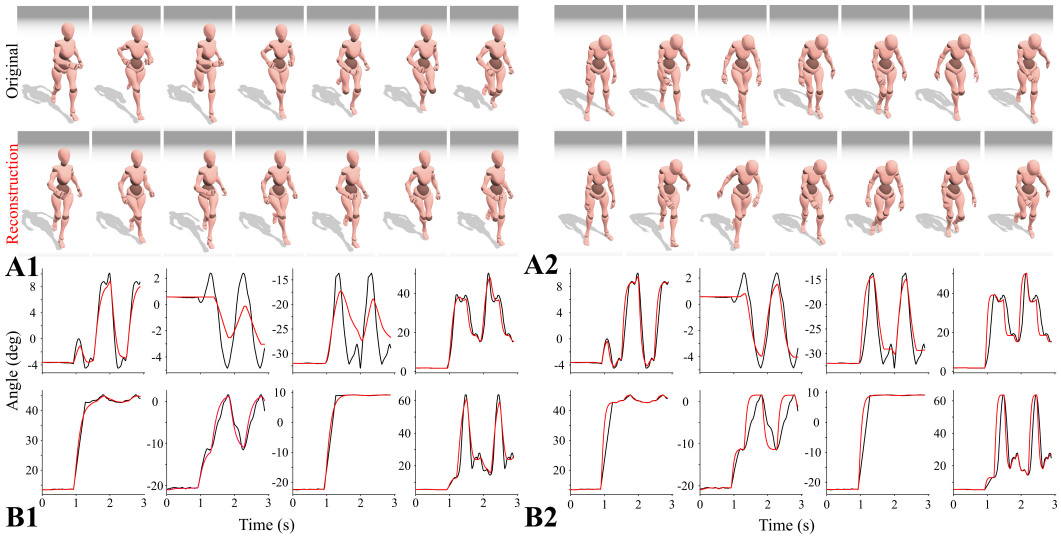

Fig. 9. **Reconstruction of motions with dynamics different from the captured ones. A.** Visualization of a running (A1) and a sad walking (A2) motion and their reconstructions based on the joint dynamics captured during regular walking and regular running, respectively. **B.** Exemplary joint angle recordings and their reconstructions.

milder ones. To verify this hypothesis, we used the controller trained on running (Section 4.2) to drive the "sad" variation of the walking motion (Fig. 9, A2-B2). Once again, the motion appeared visually realistic but preserved the dynamic properties of the behavior used for the controller's training. Namely, the "sad" walking appeared "heavier" than the reference, similar to the walking of a muscular, bulkier human (Fig. 9, A2). Looking at the individual joint movements again (Fig. 9, B1), we observed that the reconstructed motions consistently preceded the recorded ones. This effect was opposite to the one observed in the walking-to-running generalization discussed above. We attribute these observations to the increased strength of the running controller, which attempts to correct the angle errors faster than the "sad" walking behavior required.

4.5 Interactive Modification of Joint Controller Dynamics

The difference between the reference and the reconstructed motions of specific joints when generalizing to sequences with different dynamics (Section 4.4) highlighted our need to modify the controller dynamics of specific joints. Our interactive framework (Fig. 10) allows us to select the joints of interest, edit the number and location of trigger points, retrain the joint controller based on the new trigger points, and visualize the effect of the retrained controller on the motion sequence in real-time. As illustrated in Figure 11, we show how a number of user-added trigger points and the retraining of the controller based on them can modify the controller dynamics. In the supplementary video we also present the real-time visualization of the effects of the controller modification on the motion sequence.

4.6 Hand Gesture Animation using Human MoCap Data

After capturing the dynamics of synthetic data, we examined whether our controller could capture real human joint dynamics and animate them using the controlled 3D character. To investigate this, we used our method to train controllers on the dynamics of human hand joints from subjects that performed hand gestures while wearing a glove with recording sensors [Jarque-Bou et al.

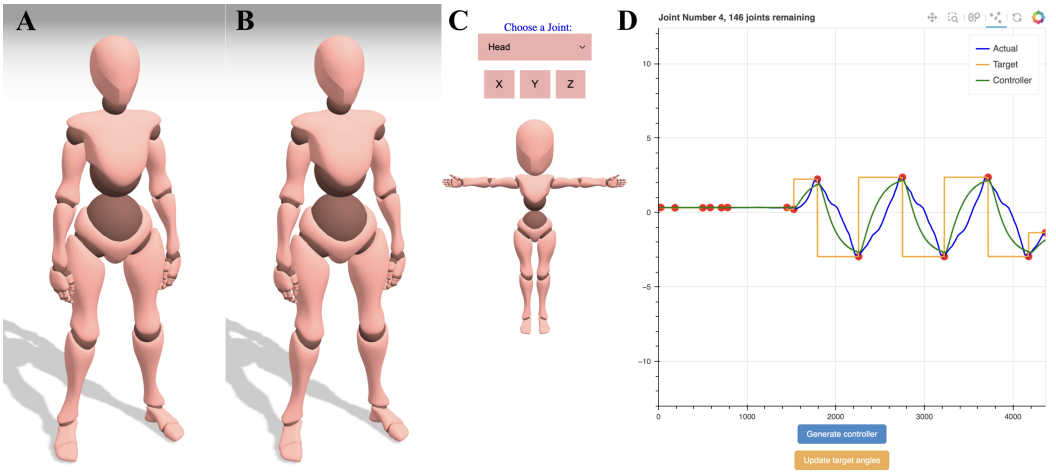

Fig. 10. **Screenshot of our interactive framework. A.** Visualization of the reference motion sequence. **B.** Visualization of the reconstruction. **C.** Panel for joint selection **D.** Panel for interactive editing of the joint dynamics.

2020]. Our evolutionary algorithm resulted in controller architectures that successfully captured the dynamics of the human hand joints (Fig. 12, B). To visualize the reconstruction of the recorded hand gestures, we translated the controlled angles to the respective DOFs of our 3D character. As a result, the hand of our character animated the flexion of the little (Fig. 12, A1) and middle finger (Fig. 12, A2) that were represented in the gesture dataset recordings. Notably, our animation captured the coupling between neighboring fingers of the hand (e.g., the ring and the little fingers). These results suggest that our training method can successfully capture real human joint dynamics and could potentially be useful for the 3D animation of human MoCap data.

4.7 Timings

As a reference, we measured the training time that our method required to capture the dynamics of a joint. Our evolutionary algorithm stopped either when the error was below 0.08 (see Eq. 5) or when 270 possible architectures had been tested. To capture the dynamics of one joint over a sequence with a duration of 1s, this process took 40.12s with a standard deviation of 32.24s when run on an Intel i5-1035G1 CPU with 8 cores at 1 GHz. The large standard deviation emerged because of the

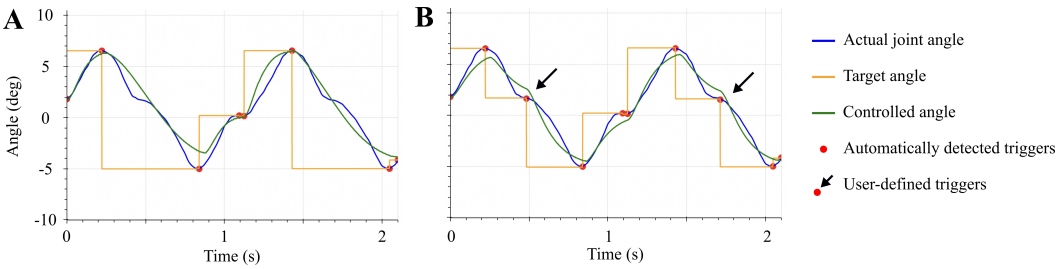

Fig. 11. **User-dependent modification of joint controller dynamics in our interactive framework. A.** Capture and reconstruction of the joint dynamics (blue) by our trained controller (green) based on the target angles (yellow) that are specified by the automatically detected trigger points (red dots) **B.** The addition of user-defined triggers (arrows) in the interactive framework initiates a retraining of the controller to capture the updated dynamics and fine-tune the movement.

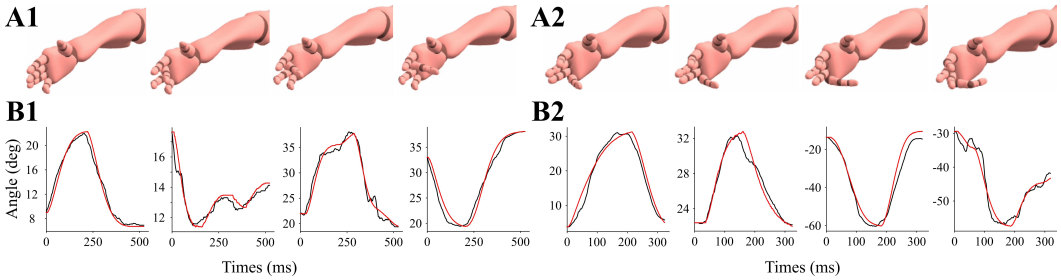

Fig. 12. **Animation of two human gestures based on human MoCap data. A.** The flexion of the middle (A1) and the little (A2) finger are recreated with the 3D animated character based on recordings of humans performing the gestures while wearing a sensor glove [Jarque-Bou et al. 2020]. **B.** The recorded dynamics of four representative joints (black) are properly captured and reconstructed by our controller (red) to give rise to the hand gesture.

two extreme training scenarios; Often the method identified a suitable controller very early after a couple of tested variations, while at other times, it only stopped after the maximum allowed number of controllers was tested. We note that this training time was measured without any parallelization of our evolutionary algorithm. Such an optimization of our method will allow for simultaneous evaluation of multiple controller variations belonging in the same generation, and would drastically decrease the training time. We have left this improvement for future work. We also measured the time that our trained controller required to reconstruct a given sequence based on its captured dynamics, corresponding to the inference time. The reconstruction of a sequence with a duration of 1s took on average 273.26ms with a standard deviation of 16.62ms.

5 LIMITATIONS AND FUTURE WORK

We presented a method to capture the dynamics of motion sequences and use them for synthesizing visually realistic 3D character animations. Unlike previous sampling methods, our focus was not on the keyframes, but on the spatiotemporal dynamics that govern the evolution of motion sequences between keyframes. While our method is endowed by its biological inspiration with realistic motion dynamics resembling those of human joints, it still has its own limitations.

First, the formalization of biological neuronal networks that drive motor controls into a simplified grammar is an abstraction of reality. Although we attempted to include enough biological realism to capture the main components of such networks, many components, mechanisms, and architectural details are not modeled. While the presented version of the grammar was sufficient to generate effective controllers, expanding its alphabet and production rules to incorporate further biological evidence would be fruitful for improving our method's performance.

Second, the evolutionary algorithm we propose allows for the accurate capturing of joint dynamics, but we do not fully exploit its potential. Specifically, while the nature of the algorithm allows for a parallel evaluation of multiple controller variants belonging to the same generation, our current implementation performs the evaluation sequentially. The sequential evaluation not only induced longer training times but, more importantly, limited the number of possible controller variations that we could test in a reasonable time frame. Therefore, investigating parallel implementations could further improve the performance of our method.

Lastly, the biologically plausible dynamics generated by our method can outperform simple methods such as interpolation with cubic polynomials (Fig. 6). Similarly to polynomial interpolation, our method did not aim to point-by-point reconstruct the original motion sequences. In contrast, it accurately captured the motion dynamics that could be transferred to different sequences. Our controller design method could perform well in cases where the goal is the optimization of specific

motion dynamics such as speed constraints, range adaptation, or overshoot avoidance (as demonstrated in [Polykretis et al. 2023] that targeted motion smoothness). However, it is still limited when compared against motion in-betweening methods that use deep learning [Harvey et al. 2020; Peng et al. 2018]. More specifically, these advanced methods can also generalize to transitions between different motions and synthesize complicated motion sequences such as gymnastics or parkour exercises, which patch fractions of different motion sequences. Therefore, the combination of our low-level controller with sophisticated high-order planning could further improve the overall performance.

6 CONCLUSION

In this work, we presented a method to capture the musculoskeletal dynamics of human joints during motion sequences and use them to synthesize visually realistic motions of 3D animated characters. We abstracted the intricate architectures of the SNNs that drive motor controls in primates and rodents and formalized them in a context-free grammar, encapsulating different network variants. We then proposed an evolutionary algorithm to allow the joint controller to learn the network architecture that best approximates the joint dynamics. We scaled up our approach to capture the synthetic motion of a 3D animated character and demonstrated the reconstruction of the original movement and generalization to previously unseen, realistic motion sequences. We also validated our method by capturing the joint dynamics of real human hands performing gestures and animated the gestures using the 3D character.

7 ACKNOWLEDGEMENTS

We thank the anonymous reviewers for their valuable feedback. I. P. was supported in part by the Onassis Foundation scholarship. M. A. was supported in part by the Rutgers University start-up grant and the National Science Foundation under awards CCF-2110861, IIS-2132972 and IIS-2238955. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

Okan Arikan, David A Forsyth, and James F O'Brien. 2003. Motion synthesis from annotations. In ACM SIGGRAPH 2003 Papers. 402–408.

Bruno Arnaldi, Georges Dumont, Gérard Hégron, Nadia Magnenat-Thalmann, and Daniel Thalmann. 1989. Animation control with dynamics. In State-of-the-art in Computer Animation. Springer, 113–123.

Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. 2017. Deep reinforcement learning: A brief survey. IEEE Signal Processing Magazine 34, 6 (2017), 26–38.

David Baraff. 1990. Curved surfaces and coherence for non-penetrating rigid body simulation. ACM SIGGRAPH Computer Graphics 24, 4 (1990), 19–28.

David Baraff. 1994. Fast contact force computation for nonpenetrating rigid bodies. In Proceedings of the 21st annual conference on Computer graphics and interactive techniques. 23–34.

David Baraff and Andrew Witkin. 1998. Large steps in cloth simulation. In Proceedings of the 25th annual conference on Computer graphics and interactive techniques. 43–54.

Christopher M. Bishop. 2006. Pattern Recognition and Machine Learning. Springer.

Lynne Shapiro Brotman and Arun N Netravali. 1988. Motion interpolation by optimal control. In Proceedings of the 15th annual conference on Computer graphics and interactive techniques. 309–315.

Jinxiang Chai and Jessica K Hodgins. 2007. Constraint-based motion optimization using a statistical dynamic model. In ACM SIGGRAPH 2007 papers. 8–es.

Michael F Cohen. 1992. Interactive spacetime control for animation. In Proceedings of the 19th annual conference on Computer graphics and interactive techniques. 293–302.

Tony DeRose, Michael Kass, and Tien Truong. 1998. Subdivision surfaces in character animation. In Proceedings of the 25th annual conference on Computer graphics and interactive techniques. 85–94.

Anthony C Fang and Nancy S Pollard. 2003. Efficient synthesis of physically valid human motion. ACM Transactions on Graphics (TOG) 22, 3 (2003), 417–426.

Andrew JP Fink, Katherine R Croce, Z Josh Huang, LF Abbott, Thomas M Jessell, and Eiman Azim. 2014. Presynaptic inhibition of spinal sensory feedback ensures smooth movement. Nature 509, 7498 (2014), 43–48.

Thomas Geijtenbeek and Nicolas Pronost. 2012. Interactive character animation using simulated physics: A state-of-the-art review. In Computer graphics forum, Vol. 31. Wiley Online Library, 2492–2515.

Félix G Harvey, Mike Yurick, Derek Nowrouzezahrai, and Christopher Pal. 2020. Robust motion in-betweening. ACM Transactions on Graphics (TOG) 39, 4 (2020), 60–1.

Erik Herrmann, Han Du, Noshaba Cheema, Janis Sprenger, Somayeh Hosseini, Klaus Fischer, and Philipp Slusallek. 2019. Adaptive gaussian mixture trajectory model for physical model control using motion capture data. In Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games. 1–8.

Néstor J Jarque-Bou, Manfredo Atzori, and Henning Müller. 2020. A large calibrated database of hand movements and grasps kinematics. Scientific data 7, 1 (2020), 1–10.

Manfred Lau and James J Kuffner. 2005. Behavior planning for character animation. In Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation. 271–280.

Jehee Lee, Jinxiang Chai, Paul SA Reitsma, Jessica K Hodgins, and Nancy S Pollard. 2002. Interactive control of avatars animated with human motion data. In Proceedings of the 29th annual conference on Computer graphics and interactive techniques. 491–500.

Yan Li, Tianshu Wang, and Heung-Yeung Shum. 2002. Motion texture: a two-level statistical model for character motion synthesis. In Proceedings of the 29th annual conference on Computer graphics and interactive techniques. 465–472.

Guodong Liu and Leonard McMillan. 2006. Segment-based human motion compression. In Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation. 127–135.

Libin Liu and Jessica Hodgins. 2018. Learning basketball dribbling skills using trajectory optimization and deep reinforcement learning. ACM Transactions on Graphics (TOG) 37, 4 (2018), 1–14.

Libin Liu, KangKang Yin, and Baining Guo. 2015. Improving sampling-based motion control. In Computer Graphics Forum, Vol. 34. Wiley Online Library, 415–423.

Libin Liu, KangKang Yin, Michiel Van de Panne, Tianjia Shao, and Weiwei Xu. 2010. Sampling-based contact-rich motion control. In ACM SIGGRAPH 2010 papers. 1–10.

Anthony R Luff. 1998. Age-associated changes in the innervation of muscle fibers and changes in the mechanical properties of motor units. Annals of the New York Academy of Sciences 854, 1 (1998), 92–101.

Jianyuan Min and Jinxiang Chai. 2012. Motion graphs++ a compact generative model for semantic motion analysis and synthesis. ACM Transactions on Graphics (TOG) 31, 6 (2012), 1–12.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. nature 518, 7540 (2015), 529–533.

Jihun Park and Donald S Fussell. 1997. Forward dynamics based realistic animation of rigid bodies. Computers & Graphics 21, 4 (1997), 483–496.

Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel Van de Panne. 2018. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. ACM Trans. on Graph. 37, 4 (2018), 1–14.

Ioannis Polykretis, Lazar Supic, and Andreea Danielescu. 2023. Bioinspired smooth neuromorphic control for robotic arms. Neuromorphic Computing and Engineering (2023). http://iopscience.iop.org/article/10.1088/2634-4386/acc204

Zoran Popović and Andrew Witkin. 1999. Physically based motion transformation. In Proceedings of the 26th annual conference on Computer graphics and interactive techniques. 11–20.

Katherine Pullen and Christoph Bregler. 2002. Motion capture assisted animation: Texturing and synthesis. In Proceedings of the 29th annual conference on Computer graphics and interactive techniques. 501–508.

Joose Rajamäki and Perttu Hämäläinen. 2017. Augmenting sampling based controllers with machine learning. In Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation. 1–9.

Balajee Ramakrishnananda and Kok Cheong Wong. 1999. Animating bird flight using aerodynamics. The Visual Computer 15, 10 (1999), 494–508.

Liu Ren, Gregory Shakhnarovich, Jessica K Hodgins, Hanspeter Pfister, and Paul Viola. 2005. Learning silhouette features for control of human motion. ACM Transactions on Graphics (ToG) 24, 4 (2005), 1303–1331.

Charles Rose, Brian Guenter, Bobby Bodenheimer, and Michael F Cohen. 1996. Efficient generation of motion transitions using spacetime constraints. In Proceedings of the 23rd annual conference on Computer graphics and interactive techniques. 147–154.

Samuel Schmidgall, Catherine Schuman, and Maryam Parsa. 2022. Biological connectomes as a representation for the architecture of artificial neural networks. bioRxiv (2022).

Kazuhiko Seki, Steve I Perlmutter, and Eberhard E Fetz. 2003. Sensory input to primate spinal cord is presynaptically inhibited during voluntary movement. Nature neuroscience 6, 12 (2003), 1309–1316.

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. 2014. Deterministic policy gradient algorithms. In International conference on machine learning. PMLR, 387–395.

David J Sturman. 1994. A brief history of motion capture for computer character animation. SIGGRAPH94, Course9 (1994).

Guy Tevet, Sigal Raab, Brian Gordon, Yonatan Shafir, Daniel Cohen-Or, and Amit H Bermano. 2022. Human motion diffusion model. arXiv preprint arXiv:2209.14916 (2022).

Misha Tsodyks, Klaus Pawelzik, and Henry Markram. 1998. Neural networks with dynamic synapses. Neural computation 10, 4 (1998), 821–835.

Hado Van Hasselt and Marco A Wiering. 2007. Reinforcement learning in continuous action spaces. In 2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning. IEEE, 272–279.

Jacob M Wilson, Jeremy P Loenneke, Edward Jo, Gabriel J Wilson, Michael C Zourdos, and Jeong-Su Kim. 2012. The effects of endurance, strength, and power training on muscle fiber type shifting. The Journal of Strength & Conditioning Research 26, 6 (2012), 1724–1729.

Andrew Witkin and Michael Kass. 1988. Spacetime constraints. ACM Siggraph Computer Graphics 22, 4 (1988), 159–168.

Jia-chi Wu and Zoran Popović. 2003. Realistic modeling of bird flight animations. ACM Transactions on Graphics (TOG) 22, 3 (2003), 888–895.

Katsu Yamane, James J Kuffner, and Jessica K Hodgins. 2004. Synthesizing animations of human manipulation tasks. In ACM SIGGRAPH 2004 Papers. 532–539.