A Limit-Cycle Self-Organizing Map Architecture for Stable Arm Control

Di-Wei Huang*,[a], Rodolphe J. Gentili[b,c,d], Garrett E. Katz[a], and James A. Reggia[a,c,d,e]

[a] *Department of Computer Science,*
[b] *Department of Kinesiology,*
[c] *Neuroscience and Cognitive Science Program,*
[d] *Maryland Robotics Center,*
[e] *University of Maryland Institute for Advanced Computer Studies,*
University of Maryland, College Park, MD 20742, United States

October 20, 2016

Abstract

Inspired by the oscillatory nature of cerebral cortex activity, we recently proposed and studied self-organizing maps (SOMs) based on limit cycle neural activity in an attempt to improve the information efficiency and robustness of conventional single-node, single-pattern representations. Here we explore for the first time the use of limit cycle SOMs to build a neural architecture that controls a robotic arm by solving inverse kinematics in reach-and-hold tasks. This multi-map architecture integrates open-loop and closed-loop controls that learn to self-organize oscillatory neural representations and to harness non-fixed-point neural activity even for fixed-point arm reaching tasks. We show through computer simulations that our architecture generalizes well, achieves accurate, fast, and smooth arm movements, and is robust in the face of arm perturbations, map damage, and variations of internal timing parameters controlling the flow of activity. A robotic implementation is evaluated successfully without further training, demonstrating for the first time that limit cycle maps can control a physical robot arm. We conclude that architectures based on limit cycle maps can be organized to function effectively as neural controllers.

Keywords: Self-Organizing Map, Neural Architecture, Limit Cycle Attractor, Robotic Arm Control, Neural Oscillation

*Corresponding author.

Email addresses: `dwh@cs.umd.edu` (Di-Wei Huang), `rodolphe@umd.edu` (Rodolphe Gentili), `gkatz12@umd.edu` (Garrett Katz), `reggia@cs.umd.edu` (James Reggia)

1 Introduction

A *neural architecture* is a system of neural networks composed of multiple interacting neural components, modules, or “regions”, that often loosely correspond to cortical or other brain regions. Traditionally, a basic module in a neural architecture is usually an unstructured layer of nodes. While these nodes may be connected to common upstream and downstream neural components, they are often not connected to one another and thus do not influence each other directly. An alternative is to build a neural module using a self-organizing map (SOM), whose nodes have explicit or implied lateral connections that lead to topographic map formation [29, 33]. A SOM learns to map high-dimensional inputs onto its map nodes that form a 2D lattice. This mapping preserves input topology non-linearly, meaning that nearby output nodes in a trained map are typically sensitive to similar input patterns. SOMs not only are widely used for unsupervised clustering and visualization in a wide range of computational applications [18, 25, 29, 38, 49], but are also reminiscent of many aspects of observed phenomena in biological cortical regions [3, 8, 37, 44, 45, 47].

Somewhat surprisingly, SOMs have only played a limited role in large-scale neural architectures. One fundamental barrier to adopting conventional SOMs in neural architectures is how they represent information. Typically, each input pattern is represented by a *single* map node (i.e., a “winning” node) selected by a winner-takes-all process. This significantly limits the number of all possible representations a SOM can express, leading to inefficient information encoding. Single-node representations are also more susceptible to node damage and activity noise than distributed population coding, resulting in less robust systems. Additionally, the activation of a winning node is typically transient, meaning that it lasts only while an input pattern is being provided. This imposes a strong restriction in that a neural architecture would have to complete all processing within the duration of input. Finally, conventional SOMs do not naturally represent temporal sequence inputs, although past work has explored extensions of SOMs that support temporal sequence processing.

To address these issues, we have recently begun to explore the possibility of designing neural architectures using SOMs based on a limit cycle representation [19–21]. With these *limit cycle SOMs*, each input pattern is encoded by a short periodic sequence of multi-winner activity patterns that collectively form a limit cycle attractor. This method utilizes distributed sparse coding as well as sustained neural dynamics that support ongoing processing after the termination of inputs. This initial work showed that limit cycle attractors have desirable representation properties such as stability/robustness, uniqueness, and distance correlation, in relation to other types of dynamics and conventional transient/static representations. In other words, limit cycle SOMs represent information in a more efficient and robust way, and they suggest the possibility of generalization to new inputs. We have also shown that limit cycle representations work with both static and temporal sequence inputs, and that topographic map formation is maintained reliably.

Given that a single SOM can encode input patterns using limit cycles, it is currently unknown whether such attractor activity can be harnessed to drive a neural architecture containing multiple SOMs, and whether such an architecture can perform useful neural computations. Further, while it seems plausible that an architecture based on oscillatory activity could be trained to generate oscillatory output patterns, it is much less clear whether such oscillatory activity can be used for more general non-oscillatory computations, such as holding a robot's arm in a fixed position, and whether an architecture based on limit cycle SOMs can generalize effectively to new situations. Our past work with limit cycle SOMs did not generate outputs at all and did not deal with continuous 3D space or with maintaining a constant output in the context of oscillations in the model's activation.

Traditional non-oscillatory SOMs have been used in solving inverse kinematics arm control problems, which is to find appropriate joint angles for an arm to reach a target spatial location [2]. In approaches using a single conventional SOM, the SOM is usually trained using concatenated vectors of joint angles and Cartesian coordinates [32, 43]. Similar strategies can be found in [1, 48], but these discrete SOM nodes are interpolated to sample a continuous manifold. In [30, 35], each SOM node stores extra information (e.g., a matrix) for performing locally linear transformations from the Cartesian space to the joint angle space. In approaches based on neural architectures containing multiple non-limit cycle SOMs, it is typical that inputs in the joint angle space and those in the Cartesian space are processed separately by two unimodal SOMs first. These unimodal SOMs are then associated directly or through other SOMs, such that appropriate joint angles can eventually be retrieved by Cartesian coordinates [27, 32]. A more biologically-realistic architecture for visually-guided arm reaching is characterized by iterative competition among SOM nodes [36].

In this current study, our goal is to examine whether one can construct a limit cycle SOM-based neural architecture to represent and control a *static* non-oscillatory task in spite of ongoing

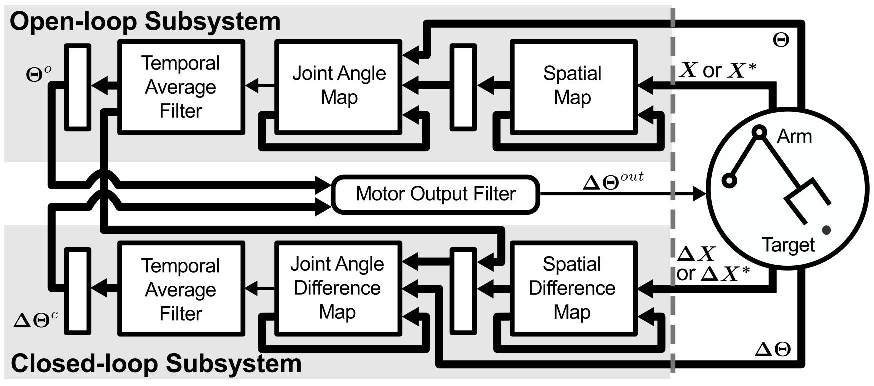

Figure 1: An overview of our neural architecture. To the left of the gray dashed line, our neural architecture is divided into open-loop and closed-loop subsystems that contain four SOMs with locally-recurrent feedback connections. Narrow rectangles without labels represent hidden layers. Thick arrow lines represent trainable synaptic connections. The input/output of the architecture is from/affects the state of the arm model (right of the gray dashed line).

oscillatory network activity, and whether this architecture can generalize to unseen inputs in 3D space. We choose to work on a robot arm control problem, where a manipulator (finger, gripper, etc) of an arm is to be moved to a *fixed* target location. The arm control problem is known to be ill-posed and non-linear, and is being studied intensively in robotics and cognitive science [5, 23]. The robotic arm in our work reported here is characterized by multiple rigid segments connected by rotatable joints, which, when rotated, change the location of the manipulator. Thus, the key question we are asking here is whether it is possible for a neural controller consisting of multiple interconnected limit cycle SOMs to arrive at a vector in the joint angle space (i.e., a set of rotation angles for all joints) that brings an arm manipulator to target spatial coordinates in 3D space, *and then holds it fixed there* in spite of the continuously changing activity in the limit cycle SOMs. To our knowledge, the work reported here is the first attempt to build a neural architecture that controls stable arm reaching movements based on limit cycle activity in SOMs.

2 Methods

Fig. 1 gives an overview of our system. To the left of the vertical dashed line is our neural architecture, including an open-loop and a closed-loop subsystem. To the right is an external environment, including an arm and a target location to be reached. During training, the arm joints are moved randomly in small steps, resembling motor babbling [5, 16]. The neural architecture reads, from the external environment, current arm joint angles $\boldsymbol{\Theta}$, current Cartesian coordinates of the manipulator $\boldsymbol{X}$, and their corresponding difference vectors $\boldsymbol{\Delta\Theta}$ and $\boldsymbol{\Delta X}$, the latter generated by subtracting the preceding joint angles and manipulator locations from the current ones during each motor babbling step. During training, a version of Hebbian learning is used to train the SOMs, and then gradient descent learning is used to train the associations between them and model output. After training, the neural architecture is given a target set of Cartesian coordinates $\boldsymbol{X}^*$ to reach for. The architecture runs iteratively to guide a reaching movement step-by-step, where each iteration corresponds to a small step in the arm trajectory. In addition to $\boldsymbol{X}^*$, the external environment also provides current spatial error $\boldsymbol{\Delta X}^* = (\boldsymbol{X}^* - \boldsymbol{X})/\|\boldsymbol{X}^* - \boldsymbol{X}\|$, a unit vector in Cartesian space pointing from the manipulator location towards the target location. Based on $\boldsymbol{X}^*$, $\boldsymbol{\Theta}$, and $\boldsymbol{\Delta X}^*$, the architecture generates a joint command $\boldsymbol{\Delta\Theta}^{out}$ at each iteration dictating joint rotations to be made by the arm, and eventually guiding the manipulator towards the target. While the manipulator location $\boldsymbol{X}$ can be derived using current joint angles $\boldsymbol{\Theta}$ (i.e., based on proprioception), determining target location $\boldsymbol{X}^*$ is a non-trivial visual image processing task in physical robotics. To simplify the problem, here we assume that a proper vision system is in place to determine Cartesian coordinates of both the target and the manipulator, and is also able to compute spatial difference vectors between them.

To limit the complexity of the problem, a 3-degree-of-freedom (3-DOF) arm operating in a 3D Cartesian space is used, having two shoulder and one elbow joint freely adjustable. This arm is modeled in a custom virtual environment for training and testing our neural architecture, before a real robotic arm is used [22].[*] Fig. 2 shows a schematic of the arm. We use both a simulated/virtual arm model and a physical robot in the experiments described later in this paper. For the virtual arm model, the current joint angles are represented by $\boldsymbol{\Theta} = (\theta_1, \theta_2, \theta_3)$, where each component is linearly scaled to be within $[0, 1]$. When given a joint command $\boldsymbol{\Delta\Theta}^{out} = (\Delta\theta_1, \Delta\theta_2, \Delta\theta_3)$, each joint angle θ_i is updated to be $\theta_i + dt\Delta\theta_i$ and then clamped within $[0, 1]$, where $dt = 1/60$ is the step size. Since the lengths of all arm segments are known, the manipulator location $\boldsymbol{X} = (x, y, z)$ can be determined exactly using the Denavit-Hartenberg method [17, p.435] based on current joint angles $\boldsymbol{\Theta}$. Each component of $\boldsymbol{X}$ is also linearly scaled to $[0, 1]$.

[*]Software available at: https://github.com/dwhuang/SMILE

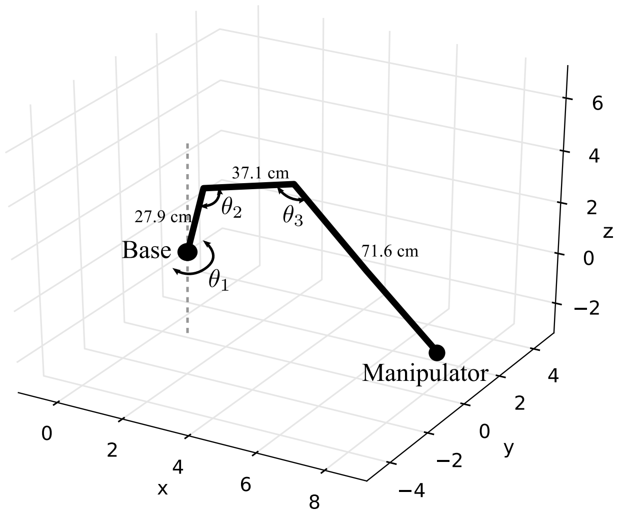

Figure 2: Schematic diagram of the arm model. The arm segment lengths shown are based on the actual size of a physical Baxter robot's arm that we use in experiments described later.

2.1 Neural Architecture

The output of our neural architecture is in the form of a joint difference vector $\mathbf{\Delta\Theta}^{out}$, which serves as a joint command that rotates the arm joints for a short period of time (i.e., ideally 1/60 second). The result is a small-step contribution to arm movement. To generate a complete arm trajectory for a reaching movement, the architecture is run multiple times, each of which is referred to as an *iteration*. In each iteration, the architecture undergoes multiple *time steps*, each of which is a discrete time unit when neural activity in the architecture is updated. This allows neural activity to dynamically change within an iteration.

Existing methods for robotic arm control utilize either open-loop control, closed-loop control, or a combination of both. *Open-loop methods* perform one-step computations to transform target spatial coordinates to a motor plan. There is no feedback control that refines the motor plan based on the outcome of motor execution. This approach avoids continuously monitoring the manipulator location and computing motor plan refinement at the cost of lower accuracy. In contrast, *closed-loop methods* continuously compare the current manipulator location with the target location, and use the differences as feedback to generate and refine motor plans. Close-loop methods typically use cameras to obtain visual feedback (i.e., visual servoing [7, 23]). The accuracy of reaching is much better in closed-loop approaches, although instability sometimes occurs as a result of latency between sensing and acting, feedback bandwidth issues, inaccurate estimation of 3D locations, etc. Some past work combines open-loop and closed-loop methods to better control reaching movements [13, 40], and our architecture is based on this latter approach.

The output of our architecture at each iteration is the result of integrating individual outputs of the open-loop and the closed-loop subsystems (see Fig. 1). The open-loop subsystem is weighted more heavily in the early iterations, and the closed-loop subsystem is weighted more heavily in later iterations. The open-loop subsystem takes a target spatial location $\boldsymbol{X}^*$ and generates a target joint angle output $\boldsymbol{\Theta}^o$. Since the target $\boldsymbol{X}^*$ remains the same throughout the reaching movement, in practice the open-loop subsystem needs to be run only once. However, the accuracy of a solely open-loop approach tends to be low because it does not take any feedback about how well the arm performs.[†] On the other hand, during each iteration the closed-loop subsystem takes a spatial difference unit vector $\boldsymbol{\Delta X}^*$ indicating the direction of spatial error from the manipulator to the target, as well as the current joint angles $\boldsymbol{\Theta}$ (indirectly via the joint angle map), and generates a joint angle difference output $\boldsymbol{\Delta\Theta}^c$. This output determines how the arm joints are to be rotated in the current iteration, based on feedback information about spatial error between the manipulator and the target. Therefore, the closed-loop subsystem can achieve higher accuracy, although it has to recompute output at each small movement step. In our architecture, the output of the open-loop subsystem has a critical role in the early phase of the reaching movement, to bring the manipulator towards the target without peripheral feedback processing. The output of the closed-loop subsystem is then gradually weighted more later in the movement, which is crucial to fine-tune the manipulator location to approach the target with relatively higher accuracy. Intuitively, this roughly corresponds to motor controls that progressively switch from a feedforward mode to a feedback mode [26].

2.1.1 Open-loop Subsystem

The open-loop subsystem corresponds to spatial-to-motor memory that makes a one-step decision about the joint angle that is associated with the given target location. That is, given a target

[†]We use the term "open-loop" here only in the limited sense that the open-loop subsystem considered in isolation (gray shading in Fig. 1) produces an output $\boldsymbol{\Theta}^o$ without any feedback from the environment, in contrast to the closed-loop subsystem considered in isolation. However, the open-loop subsystem is not truly "open-loop" in the sense that $\boldsymbol{\Theta}^o$ is modified downstream by the motor output filter by the current joint angle $\boldsymbol{\Theta}$ in generating the overall system's control signal $\boldsymbol{\Delta\Theta}^{out}$ (see Eq. 13).

location $\boldsymbol{X}^*$, the open-loop subsystem generates absolute joint angles $\boldsymbol{\Theta}^o$ to reach for the target. It does not rely on visual feedback and has relatively low accuracy. Since the given target location is assumed to remain unchanged throughout arm reaching, the open-loop subsystem is required to run once only per reaching movement. Our architecture uses an open-loop subsystem in the early stages of an arm trajectory to quickly guide the manipulator to the vicinity of the target.

The open-loop subsystem contains two SOMs, as illustrated in Fig. 1 (also see Fig. 5a for the active components involved in open-loop execution). A *spatial map* encodes Cartesian coordinates $\boldsymbol{X} = (x, y, z)$, and a *joint angle map* encodes joint angles $\boldsymbol{\Theta} = (\theta_1, \theta_2, \theta_3)$. The two SOMs are similar to those used in [19, 21], each having both afferent and recurrent connections. The afferent connections fully connect their respective inputs, i.e., $\boldsymbol{X}$ and $\boldsymbol{\Theta}$, while recurrent connections exist topographically between neighboring map nodes that are within a topological box distance of 2. The input coming from the recurrent connections has a unit time step delay. A set of associative connections fully connect the spatial map to the joint angle map through a hidden layer, allowing retrieval of corresponding limit-cycle neural activity in the joint angle map based on that in the spatial map. This forms an additional set of incoming connections for the joint angle map. The relative strengths for inputs coming from different sets of connections are expressed as adjustable gating parameters α. The net inputs h_i for each node i in the two maps at time step t, given afferent inputs $\boldsymbol{X}$ (or $\boldsymbol{X}^*$ after training) and $\boldsymbol{\Theta}$, is defined by:

$$h_i^S(t) = -\,\alpha_{\mathit{aff}}^S(t)\left\|\boldsymbol{X} - \boldsymbol{w}_i^S\right\|^2 + \alpha_{rec}^S(t)\boldsymbol{a}^S(t-1)\cdot\boldsymbol{u}_i^S, \tag{1}$$

$$\begin{aligned} h_i^J(t) = &-\,\alpha_{\mathit{aff}}^J(t)\left\|\boldsymbol{\Theta} - \boldsymbol{w}_i^J\right\|^2 + \alpha_{rec}^J(t)\boldsymbol{a}^J(t-1)\cdot\boldsymbol{u}_i^J \\ &+ \alpha_{assoc}^J(t) f_{assoc}^o(\boldsymbol{a}^S(t)), \end{aligned} \tag{2}$$

where superscripts S and J correspond to the spatial and the joint angle maps, respectively. Each $\boldsymbol{a}(t)$ is the activity pattern of a SOM at time t, and f_{assoc}^o represents the fully-connected, two-layer feedforward net between the two maps (superscript o denotes open-loop). The activation of the hidden nodes is based on the standard logistic function. Trainable parameters $\boldsymbol{w}$'s and $\boldsymbol{u}$'s denote the afferent and recurrent connection weights, respectively. Parameters α, taking on values in $[0, 1]$, are relative strengths that gate inputs from different sources, and can be different at different t. Subscripts *aff*, *rec*, and *assoc* correspond to afferent, recurrent, and associative inputs (if any), respectively. The first (afferent) terms follow standard distance-based SOM activation, where the negative signs assign a higher h_i when the afferent weights $\boldsymbol{w}_i$ are closer to the inputs. Given the values h_i, the activity output of the maps at time t can be determined using multi-winners-take-all dynamics that perform local competitions over h_i to assign peaks of activation centered at winning nodes:

$$a_i(t) = \min\left(1, \sum_{k \in (\{i\} \cup \mathbb{N}_i) \cap \mathbb{W}(t)} \gamma^{d(i,k)}\right), \tag{3}$$

$$\mathbb{N}_i = \{k \mid k \neq i \text{ and } d(i,k) \leq 2\}, \tag{4}$$

$$\mathbb{W}(t) = \{k \mid h_k(t) > h_l(t), \forall l \in \mathbb{N}_k\}, \tag{5}$$

$$d(i,k) = \max(|row_i - row_k|, |column_i - column_k|), \tag{6}$$

where $0 \leq \gamma < 1$ is a parameter controlling the degree of activation for nodes surrounding a winning node. $\mathbb{N}_i$ defines the competition neighborhood around node i to be of radius 2 (incidentally, but not required to be, identical to the radius of recurrent connections), where the distance d between nodes in a SOM is calculated using box distance. $\mathbb{W}(t)$ denotes the set of winning nodes. Eq. 3 states that the activation level of each node is determined by how close the node is to all winner nodes in its local neighborhood, and that it is upper-bounded by 1. Winner nodes are always fully activated (take on value 1), since $d(i,k)$ in Eq. 3 is 0. Since each activity pattern $\boldsymbol{a}(t)$ depends on $\boldsymbol{a}(t-1)$, the map activity changes with time and forms a dynamical system. In recent work, we found that limit cycles are a prominent class of attractors in a system like this, and that they are learned via self-organization [19, 21]. Each activity limit cycle in the spatial map (joint angle map) is said to encode the spatial location (joint angles) that triggers it.

Given the continuously changing activation patterns in the open-loop subsystem, the real challenge here is to get the arm manipulator to its target location *and* maintain it there in a steady fashion in spite of the continuously changing neural activity patterns. In order to generate steady joint outputs, the oscillatory activity in the joint angle map needs to be "smoothed out". For this purpose, a temporal average filter is added downstream of the joint angle map (Fig. 1). The filter contains the same number of nodes as the joint angle map, and each filter node connects one-to-one to the nodes in the map. The activity of each node i in the filter is a temporally moving average of the corresponding map node's activity in the last t^{filter} time steps (a parameter):

$$a_i^{F,o}(t) = \frac{1}{t^{filter}} \sum_{t'=0}^{t^{filter}-1} a_i^J(t-t'), \tag{7}$$

where superscript F stands for filter and o for open-loop. Finally the open-loop output

$$\mathbf{\Theta}^o = f_{out}^o(\boldsymbol{a}^{F,o}(t^{out})), \tag{8}$$

is generated, where f_{out}^o, like f_{assoc}^o above, represents the fully-connected, two-layer feedforward net between the temporal average filter and the motor output filter. Parameter t^{out} represents a fixed time step at which the output is taken. Although here the output is taken at a specific time step, we have shown in a recent study that continuous outputs with an open system alone yield stable joint angles with very small oscillations (e.g., around 5×10^{-5} in a normalized Cartesian space) [20]. This means that t^{out} can be chosen quite arbitrarily, but it must be late enough such that the values of $\boldsymbol{a}^{F,o}(t)$ become stabilized.

2.1.2 Closed-loop Subsystem

The closed-loop subsystem iteratively transforms spatial errors to joint displacements, guiding the manipulator towards the target in small steps, each of which is called an iteration. Keeping track of spatial errors requires visual and proprioceptive feedback to compare the location of the manipulator with that of the target. Our architecture uses the closed-loop subsystem in later iterations of an arm trajectory to fine-tune the location of the manipulator, and thereby achieves higher accuracy. Specifically, in each iteration, the closed-loop subsystem runs for multiple time steps starting from $t = 0$ and eventually generates an output vector. This output vector represents a desired direction for rotating arm joints (a unit vector in the joint angle space) during the current iteration, based on the currently observed spatial error direction (a unit vector in the Cartesian space) as well as the current joint angles. In the next iteration, t is reset to 0 and the process is repeated.

Like the open-loop subsystem, the closed-loop subsystem contains two SOMs (see Fig. 1), a *spatial difference map* and a *joint angle difference map*, which encode unit difference vectors in Cartesian and joint angle spaces ($\mathbf{\Delta X}$ and $\mathbf{\Delta\Theta}$), respectively (see also Fig. 5b for active components involved in closed-loop execution). During training, these difference vectors are generated by adding small random perturbations to current joint angles (motor babbling), and the resulting differences of manipulator locations are recorded [5]. These joint angle and spatial difference vectors are then normalized so that they specify the directions but not the magnitudes of the spatial or joint angle differences. This reduces the necessary number of training samples and network sizes needed because they do not need to account for different magnitudes in the same difference direction. Therefore, the two SOMs in the closed-loop subsystem contain fewer nodes than those in the open-loop subsystem, although they are otherwise structurally identical to those open-loop SOMs. There is also a set of fully connected associative connections via a hidden layer between the two SOMs. However, a distinction in the closed-loop subsystem is that there is an additional set of incoming connections from the open-loop subsystem's joint angle map via its downstream temporal average filter (see Fig. 1), which conveys neural activity encoding current joint angles to the closed-loop subsystem joint angle difference map. As such, the net inputs h_i for each node i in the two closed-loop maps at time step t, given unit vectors $\boldsymbol{\Delta X}$ (or $\boldsymbol{\Delta X}^*$ after training) and $\boldsymbol{\Delta\Theta}$ as afferent inputs, are similar to Eqs. 1 and 2:

$$\begin{aligned} h_i^{SD}(t) = &- \alpha_{aff}^{SD}(t) \left\| \boldsymbol{\Delta X} - \boldsymbol{w}_i^{SD} \right\|^2 + \alpha_{rec}^{SD}(t) \boldsymbol{a}^{SD}(t-1) \cdot \boldsymbol{u}_i^{SD}, \quad (9) \\ h_i^{JD}(t) = &- \alpha_{aff}^{JD}(t) \left\| \boldsymbol{\Delta\Theta} - \boldsymbol{w}_i^{JD} \right\|^2 + \alpha_{rec}^{JD}(t) \boldsymbol{a}^{JD}(t-1) \cdot \boldsymbol{u}_i^{JD} \\ &+ \alpha_{assoc}^{JD}(t) f_{assoc}^{c}(\boldsymbol{a}^{SD}(t), \boldsymbol{a}^{F,o}(t)), \quad (10) \end{aligned}$$

where superscripts SD and JD correspond to the spatial difference map and the joint angle difference map, respectively, and f_{assoc}^c represents the fully connected, two-layer feedforward net between the two maps (superscript c represents closed-loop). Notice that f_{assoc}^c takes an additional input vector $\boldsymbol{a}^{F,o}$ (Eq. 7), the activity of the temporal average filter downstream of the joint angle map. The output activation of the two maps follows Eqs. 3–6.

Similarly to Eqs. 7–8, oscillatory activity in the joint angle difference map needs to be stabilized to near-static outputs, so it is passed through a temporal average filter and then a two-layer

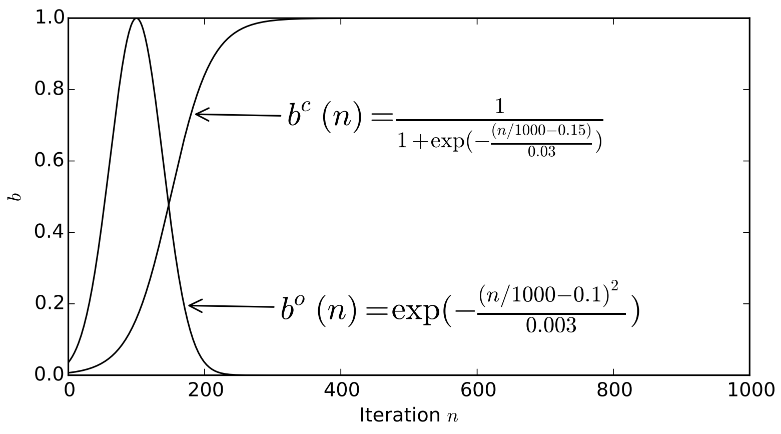

Figure 3: Weighting functions that modulate the influences of the open-loop (b^o) and closed-loop (b^c) outputs on the final output of the architecture.

feedforward net. The output of the closed-loop subsystem for each iteration is

$$a_i^{F,c}(t) = \frac{1}{t^{filter}} \sum_{t'=0}^{t^{filter}-1} a_i^{JD}(t-t'), \tag{11}$$

$$\boldsymbol{\Delta\Theta}^c = f_{out}^c(\boldsymbol{a}^{F,c}(t^{out})), \tag{12}$$

where t^{filter} is the same parameter as in Eq. (7), and f_{out}^c represents the fully-connected, two-layer feedforward net connecting the temporal average filter to the motor output filter. The activation of f_{out}^c is designed to generate unit vectors, and the learning rules are derived accordingly. Therefore, $\boldsymbol{\Delta\Theta}^c$ is a unit vector informing the direction in the joint angle space in which the arm joints are to be rotated. Without considering the magnitude, the network is focused on the direction aspect of joint angle differences, and is expected to generalize better in that regard. The parameter t^{out}, as in Eq. 8, is selected such that it is late enough that the values of $\boldsymbol{a}^{F,c}(t)$ become stabilized.

2.1.3 Motor Output Filter

At time step t^{out} of each iteration, the motor output filter (see Fig. 1) aggregates the outputs from both the open-loop and the closed-loop subsystem and generates a motor output. A motor output at each iteration n is computed by weighting both subsystems based on predefined weighting functions b:

$$\boldsymbol{\Delta\Theta}^{out}(n) = b^o(n)\left(\boldsymbol{\Theta}^o - \boldsymbol{\Theta}\right) + b^c(n)\boldsymbol{\Delta\Theta}^c(n)\left(\sigma \left\|\boldsymbol{X}^* - \boldsymbol{X}\right\|\right), \tag{13}$$

where $\boldsymbol{\Theta}^o$ is the output of the open-loop subsystem and $\boldsymbol{\Delta\Theta}^c(n)$ is that of the closed-loop subsystem at iteration n, and $\boldsymbol{\Theta}$ is the current joint angle. The value of $\boldsymbol{\Theta}^o$ is independent of n and remains fixed throughout the whole arm trajectory because its input, the target location, remains fixed. In practice the open-loop subsystem is run once at the beginning of an arm reaching event, where its output $\boldsymbol{\Theta}^o$ is cached in the motor output filter. Functions $b^o(n)$ and $b^c(n)$ serve as multiplicative weights/gains that modulate the influences of the open-loop and closed-loop subsystems on the final motor output for each iteration n. Their values are given in Fig. 3. Similar modulatory weights b have been used in past neural models to control the degree in which a motor plan is converted to actions, the mechanism of which is believed to be related to the basal ganglia or the prefrontal cortex [15]. In early iterations, influences of both subsystems increase, with the open-loop subsystem rising more rapidly and contributing more. In later iterations, the situation reverses, with the closed-loop subsystem gradually dominating the motor output. Such weighting functions are inspired by human motor control, in which early stages (up to 50–100 ms) of a movement are in a feedforward mode, i.e., primarily without sensory feedback, while later stages (beyond to 50–100 ms) are in a feedback mode [31]. Also notice that since the closed-loop output $\boldsymbol{\Delta\Theta}^c(n)$ is a unit vector informing direction but not magnitude, the value of $\sigma \left\|\boldsymbol{X}^* - \boldsymbol{X}\right\|$ serves as the magnitude of joint rotations, where $\sigma = 10$ is a constant scaling factor determined empirically.

2.2 Training

The organizational similarity of the open-loop and the closed-loop subsystems allow for similar training processes. Training of the architecture is divided into three stages. The four maps are first trained individually using unsupervised learning. The two subsystems are then trained separately to generate outputs based on the joint angle map and the joint angle difference map. Finally the inter-modality associations are learned between the spatial map and the joint angle map, as well as between the spatial difference map and the joint angle difference map. These associations between the joint angle space and the Euclidean space correspond to the proprioceptive and the visual

Table 1: Summary of parameter values for training and execution.*

Variable	Description	Default values Training	Execution
α_{aff}	Gating parameters in Eqs. (1), (2), (9), and (10). They take on default values when enabled, and 0 when disabled.	0.64	0.64
α_{rec}		0.36	0.36
α_{assoc}		n/a	0.4
t^{out}	Timing parameters described in Sect. 2.2 and 2.3, and in Eqs. (7), (8), (11), and (12).	100	130
t^{filter}		30	30
t^{assoc}		100	50
μ_1	Learning rates in Eqs. (14) and (15).	$0.44/(1+\exp((\phi-0.4)/0.0001))$	n/a
μ_2		$0.62/(1+\exp((\phi-0.8)/0.04))$	n/a
γ	Peak parameter in Eq. (3).	$0.37/(1+\exp((\phi-0.2)/0.16))$	0

*The variable ϕ represents the proportion of completed training epochs.

modalities, respectively. The end result is that when given a spatial location input, the open-loop subsystem can generate a proper joint angle output, and that when given a spatial difference input, the closed-loop subsystem can generate a proper joint angle difference output. Parameter values used during training and execution (next subsection) are summarized in Table 1.

2.2.1 Stage 1: Individual map training

In this first stage of learning, the four maps in the architecture are trained individually using unsupervised learning, to obtain limit cycle representations for their respective afferent input domains (see Fig. 4a). Each map independently forms internal representations for external stimuli in different modalities: spatial location, spatial difference, joint angle, and joint angle difference. The training data are sampled randomly in each modality, where the spatial difference and joint angle difference data are then normalized to unit lengths. Each data sample is presented to each map for 2 time steps (i.e., α_{aff} enabled for $0 \leq t < 2$), after which the map continues to run and adapt for another 4 time steps (i.e., α_{rec} enabled for $0 \leq t < 6$). The latter period is referred to as the *continuation time*. Gate α_{assoc} (if any) remains disabled (i.e., takes on value 0). When a gate α is enabled, it takes on a positive value (see Table 1) to allow the corresponding input activity to affect the SOM's own activity. Our choices of enabled α values are determined empirically so that each source of inputs has approximately equal contribution to the net input. While our preliminary results indicated that a wide range of α values led to similar limit cycle properties, the values used here are kept consistent with our past studies [20, 21].

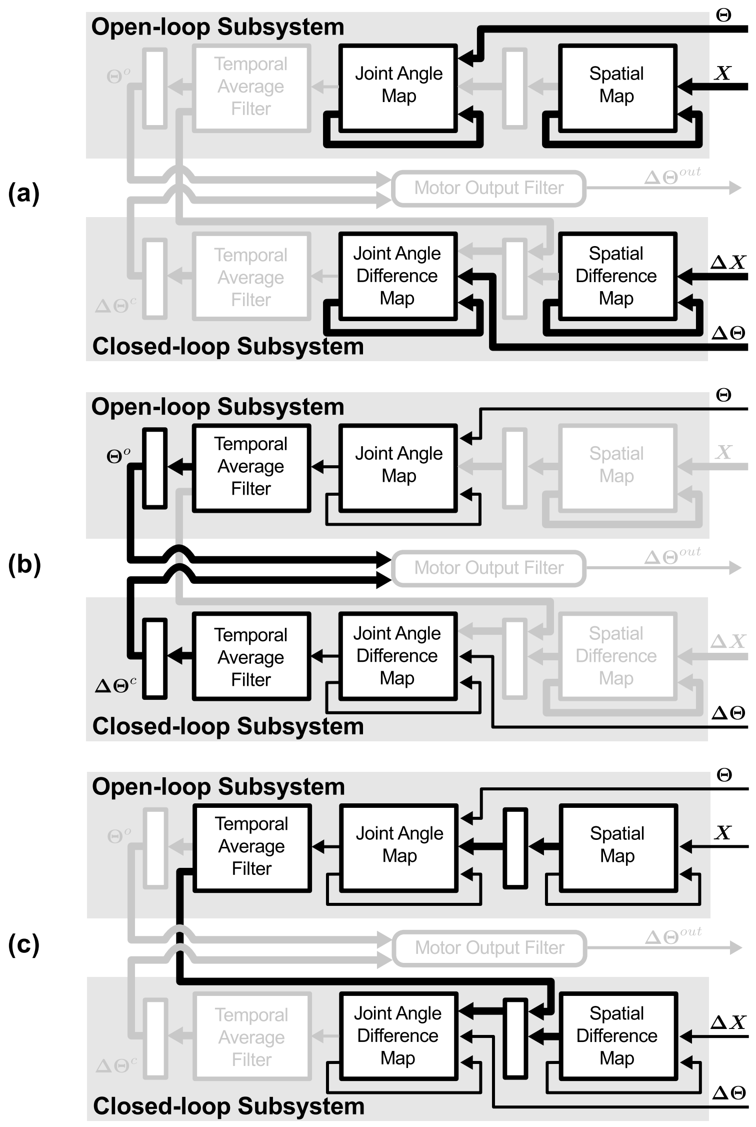

Figure 4: Schematic illustration of the three training stages. Temporarily disabled components and connections are grayed out. Thick arrow lines indicate the connections that are being adapted. Thin arrow lines indicate the connections that are used to spread neural activity but are not being adapted. (a) Stage 1: individual map training. (b) Stage 2: learning joint command outputs. (c) Stage 3: learning inter-modality associations.

At each time step, afferent weights $\boldsymbol{w}$ are updated based on a typical unsupervised SOM learning rule, and recurrent weights $\boldsymbol{u}$ are updated based on temporally asymmetric Hebbian learning [44]:

$$\boldsymbol{w}_i(t+1) = \boldsymbol{w}_i(t) + \mu_1 a_i(t)(\boldsymbol{a}^{Input} - \boldsymbol{w}_i(t)), \tag{14}$$

$$\hat{u_{ik}}(t+1) = u_{ik}(t) + \mu_2 a_k(t-1)\max(0, a_i(t) - a_i(t-1)), \tag{15}$$

$$u_{ik}(t+1) = \hat{u_{ik}}(t+1)/\sum_l \hat{u_{il}}(t+1), \tag{16}$$

where μ_1 and μ_2 are learning rates, and $\boldsymbol{a}^{Input}$ denotes an afferent input vector. The learning rates μ_1 and μ_2, as well as the activation parameter γ in Eq. 3, are decreased nonlinearly throughout the training process (see Table 1). Upon completion of this stage, limit cycle representation is expected to occur in all four maps when the maps are allowed to run for a longer period of time, e.g., by keeping α_{rec} enabled for $0 \leq t \leq 500$. An example of how an input pattern is encoded by a length six limit cycle is given in Sect. S1 of the online Supplementary Material.

2.2.2 Stage 2: Joint command output

In this stage, the two subsystems learn to generate outputs that match given joint proprioceptive inputs (see Fig. 4b). Specifically, for the open-loop subsystem, when a joint proprioception pattern $\boldsymbol{\Theta}$ is presented to the joint angle map, the goal is to eventually generate $\boldsymbol{\Theta}^o$ such that $\boldsymbol{\Theta}^o \approx \boldsymbol{\Theta}$. Similarly, the closed-loop subsystem is to learn to generate $\boldsymbol{\Delta\Theta}^c \approx \boldsymbol{\Delta\Theta}$, where $\boldsymbol{\Delta\Theta}$ is a proprioceptive input to the joint angle difference map. The training samples are again generated randomly. Each input ($\boldsymbol{\Theta}$ or $\boldsymbol{\Delta\Theta}$) results in a limit cycle in its corresponding map, whose activity is then passed to the downstream temporal average filter that generates average activity for the most recent t^{filter} time steps, where t^{filter} is a parameter (see also Eq. 7). The value of t^{filter} can be set rather arbitrarily, as long as it covers the lengths of most limit cycles. At a pre-specified time step t^{out}, outputs $\boldsymbol{\Theta}^o$ and $\boldsymbol{\Delta\Theta}^c$ are generated by passing the activity of the temporal average filter through the two-layer feedforward nets f^o_{out} and f^c_{out} (see Eqs. 8 and 12). Again, t^{out} can be set rather arbitrarily, as long as it is late enough such that the activity dynamics of the joint angle map and the joint angle difference map have entered a limit cycle, such that the activity of the temporal average filters becomes stabilized. This is because activity of a limit cycle is regular, and thus the results of temporal averages at different time steps are quite similar. Finally, f^o_{out} and f^c_{out} are trained using a resilient error-backpropagation method [24]. Notice that it is possible for different joint angles to be mapped to the same limit cycle due to SOM's discretization effect, and thus an input pattern for f^o_{out} can potentially correspond to multiple target patterns in the training data. In this case, only the "most relaxed" joint position among them, i.e., $\arg\min_{\boldsymbol{\theta}} \|\boldsymbol{\theta} - (.5, .5, .5)\|$, is selected as the target output for training. For multiple joint angle differences corresponding to the same activity limit cycle in the joint angle difference map, an arbitrary one is picked as the target output for training.

2.2.3 Stage 3: Inter-modality associations

In this final stage, the associative connections between the spatial map and the joint angle map (open-loop), as well as those between the spatial difference map and the joint angle difference map (closed-loop), are trained (see Fig. 4c). The goal of this stage is to establish associations between the spatial modalities (i.e., spatial locations and differences) and the joint modalities (i.e., joint angles and differences), and to eventually be able to transform spatial inputs to joint outputs. To this end, limit cycle activity in the spatial and spatial difference maps needs to be correlated to that in the corresponding joint angle and joint angle difference maps through the feedforward nets f^o_{assoc} and f^c_{assoc}. This is a much harder problem than associating patterns for two conventional single-winner SOMs with static representations, because each limit cycle representation contains multiple activity patterns where each activity pattern contains distributed winning nodes.

Specifically, in the open-loop subsystem, consider an activity sequence A^J in the joint angle map triggered by an arbitrary afferent input $\boldsymbol{\Theta}$, and the activity sequence A^S in the spatial map triggered by the afferent input $\boldsymbol{X}$ corresponding to $\boldsymbol{\Theta}$. Since A^J and A^S contain the limit cycles representing $\boldsymbol{\Theta}$ and $\boldsymbol{X}$, the goal is to train f^o_{assoc} to learn the associations between activity sequences in the two maps. That is, when the joint angle map's afferent input becomes unavailable, an activity sequence similar to A^J is to be retrieved in the joint angle map by the associative input sequence $f^o_{assoc}(A^S)$ alone (e.g., $\alpha^J_{aff} = 0$ and $\alpha^J_{assoc} > 0$). Here $f^o_{assoc}(A^S)$ represents the activity sequence resulting from passing each activity pattern in A^S through f^o_{assoc}. Similarly in the closed-loop subsystem, consider a small joint angle change $\boldsymbol{\Delta\Theta}$ from joint position $\boldsymbol{\Theta}$ that causes the manipulator's location to move $\boldsymbol{\Delta X}$, and that they as afferent inputs trigger activity sequences A^{SD}, A^J, and A^{JD} in the spatial difference map, the joint angle map, and the joint angle difference map, respectively. The goal is to train f^c_{assoc} to learn to retrieve an activity sequence similar to A^{JD} in the joint angle difference map solely by the associative inputs $f^c_{assoc}(A^{SD}, \overline{A^J})$, where $\overline{A^J}$ is a result of temporally averaging A^J. Compared with f^o_{assoc}, f^c_{assoc} contains an additional set of incoming connections from the open-loop temporal average filter ($\overline{A^J}$) that needs to be trained. Training inputs are obtained by concatenating patterns in A^{SD} and $\overline{A^J}$.

Training data are generated by randomly sampling $\boldsymbol{\Theta}$ and $\boldsymbol{\Delta\Theta}$ (and thus $\boldsymbol{X}$ and $\boldsymbol{\Delta X}$). Fixed-length short sequences of activity in the four maps, recorded at a fixed time step, are used to train f^o_{assoc} and f^c_{assoc} using supervised learning. Since the training sequences are potentially periodic, each target output sequence can be aligned differently against its corresponding input sequence. Our learning rule aims to determine the alignments that facilitate generalization throughout the training process. Further, our learning rule accounts for the fact that the "output layer" of $f^{\cdot}_{assoc}$ is based on multi-winner-takes-all activation. Detailed descriptions of these learning methods are given in Sect. S2 of the online Supplementary Material.

2.3 Experimental methods

After training, a set of 3D spatial locations are selected as targets for arm reaching for validation and evaluation purposes. Each target location $\boldsymbol{X}^*$ for testing is generated by manually rotating the arm joints to each of a $10 \times 10 \times 10$ grid of points in the joint angle space (i.e., each $\theta_i = \{.05, .15, .25, \cdots, .95\}$), and then recording the manipulator's locations. This ensures that the targets are reachable and include some extreme locations that are hard to reach. Each $\boldsymbol{X}^*$ may be reachable by multiple joint angles, and therefore the joint angles generated by the trained architecture may be different from those at the $10 \times 10 \times 10$ grid points.

The neural architecture is run for multiple iterations until the spatial error, measured as the Cartesian distance between the manipulator and the target location $\|\boldsymbol{X}^* - \boldsymbol{X}\|$, stays below a predetermined threshold ϵ for consecutive 20 iterations, where ϵ represents a target accuracy. We use $\epsilon = 0.001$ in the following. This stopping criteria require that the manipulator not only is positioned close to the target, but also maintains that position for a period of time, which implies stability of the arm. If the stopping criteria is never met, the architecture stops at the 1000th iteration. In either case, the final spatial error and the number of iterations used are reported as performance metrics of the architecture.

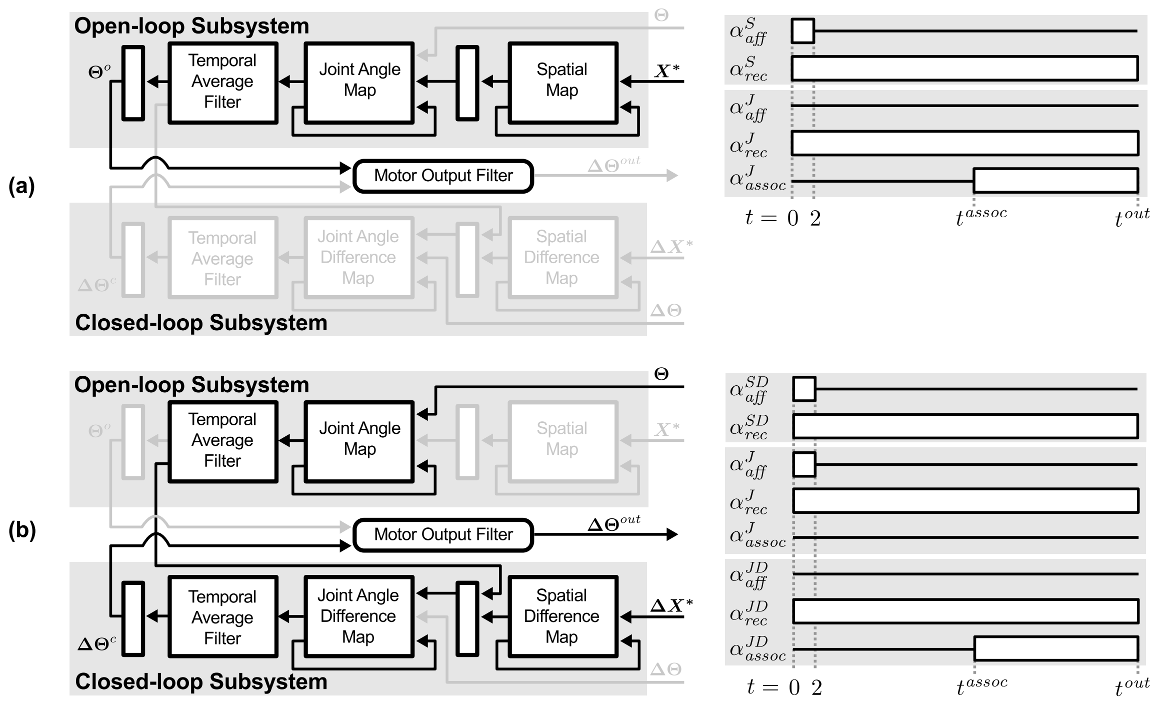

Figure 5: Schematic illustration of the execution of (a) the open-loop and (b) the closed-loop subsystems. The left column shows neural components and connections involved, and the right column shows the gate timing of the enabling (thick bar) and the disabling (thin line) of connections. See Eqs. 1, 2, 9, and 10 for the roles of α, and Table 1 for the values used.

For each target, the open-loop subsystem is run with input $\boldsymbol{X}^*$ and generates output $\boldsymbol{\Theta}^o$. Since the input to the open-loop subsystem is never changed for the same spatial target, practically it is run once only in the first iteration and the output $\boldsymbol{\Theta}^o$ is cached in the motor output filter. On the other hand, the closed-loop subsystem is run once per iteration, taking inputs $\boldsymbol{\Delta X}^*$ (spatial error direction to the target) and $\boldsymbol{\Theta}$ (current joint angles) and generating outputs $\boldsymbol{\Delta\Theta}^c$. At the motor output filter, the outputs from both subsystems are integrated according to Eq. 13 and a final motor output $\boldsymbol{\Delta\Theta}^{out}$ is generated for each iteration. This output causes the arm to move a small step. The new $\boldsymbol{\Theta}$ and $\boldsymbol{\Delta X}^*$ in the environment are fed back for the next iteration.

Gating and timing for running the open-loop subsystem are shown in Fig. 5a. The spatial map receives as afferent input the target location $\boldsymbol{X}^*$ in the first two time steps, while the afferent input for the joint angle map remains disabled (see α_{aff} rows). The recurrent connections for both maps remain open (α_{rec} rows). From $t = 2$, the activity of the spatial map starts a brief irregular dynamics and then settles in a limit cycle attractor. The associative connections f^o_{assoc} between the two maps, initially closed, are opened at a fixed time $t = t^{assoc}$ (see α^J_{assoc}). From this point on, the changing activity of the spatial map starts to drive the activity of the joint angle map, which is initially silent. In addition to the input from the spatial map, the activity of the joint angle map is also affected by itself through its own recurrent connections. Finally, at a fixed time $t = t^{out}$, the output $\mathbf{\Theta}^o$ is generated by passing the activity of the temporal average filter, which maintains an average of the most recent t^{filter} activity patterns in the joint angle map, through the output feedforward net f^o_{out}.

A similar process is performed for the closed-loop subsystem (see Fig. 5b), except that the joint angle map also participates in computing output $\mathbf{\Delta\Theta}^c$. That is, the current joint angle $\mathbf{\Theta}$ is fed to the joint angle map at the first two time steps (α^J rows). Then, starting from $t = t^{assoc}$, averaged joint angle map activity, together with limit cycle activity in the spatial difference map, jointly triggers limit cycle activity in the joint angle difference map that eventually generates $\mathbf{\Delta\Theta}^c$.

As with training, the values of t^{assoc} and t^{out} can be set rather arbitrarily anytime after the activity of the maps has entered limit cycle attractors. The value of t^{filter} can also be fixed rather arbitrarily. Importantly, they do not depend on the exact timing of individual limit cycles (i.e., the exact start time and the length), and thus the boundaries of limit cycles do not need to be detected by the architecture.

The results reported below are obtained using an architecture with $40 \times 30 = 1200$ nodes in each of the spatial and joint angle maps, $30 \times 20 = 600$ nodes in each of the spatial difference and joint angle difference maps, 200 nodes in each of the hidden layers of the open-loop feedforward nets f^o_{assoc} and f^o_{out}, and 150 nodes in each of the hidden layers of the closed-loop feedforward nets f^c_{assoc} and f^c_{out}. A total of 10 independent simulations are performed in each computational experiment with different initial random weights each time. The average results are reported below. See Table 1 for the parameter values used during execution.

3 Results

This section reports the results obtained using computer simulations, while the next section describes additional results with a physical robot.

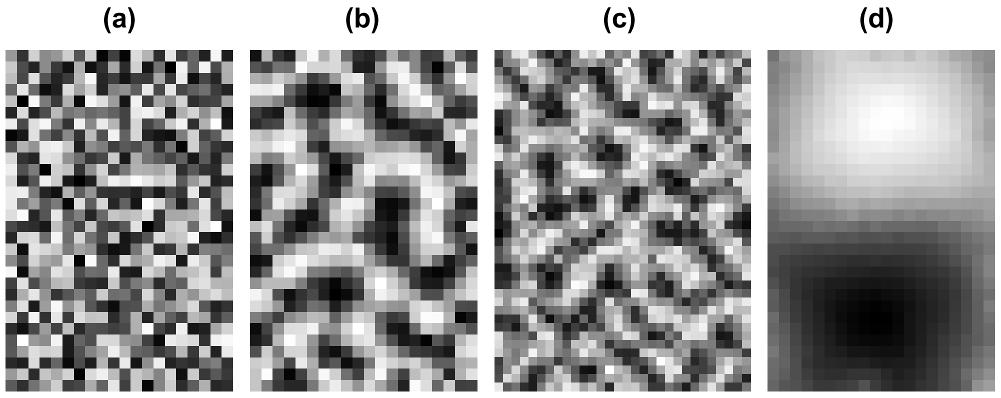

Figure 6: Examples of map formation. Each cell represents the value of the same selected afferent weight for each node. Brighter shades indicate higher values. (a) The random pre-training weight values corresponding to input Δy in the spatial difference map. Coordinate y is defined in Fig. 2. (b) The post-training weight values for (a). (c) The post-training weight values corresponding to y in the spatial map, which contains more nodes than the spatial difference map. (d) For comparison purposes, the results of training a conventional SOM is also plotted. This SOM is trained with exactly the same data as in (b), but using single-winner activation and without continuation time during training.

3.1 Map and limit cycle formation

In the first stage of training, all four maps in the architecture are trained separately using unsupervised learning with random inputs. Fig. 6 shows examples of how the maps become self-organized after training. In Fig. 6a, the weights in the spatial difference map that correspond to input Δy (y is a Cartesian coordinate defined in Fig. 2) are initially random. After training (Fig. 6b), the weights become self-organized into quasi-repetitive patterns of high and low value clusters, forming dark and light interleaved stripes. In Fig. 6c, the weights after training in the spatial map that correspond to input Cartesian coordinate y shows a similar organization as in Fig. 6b, despite a different map size and that the input coordinates are not normalized as in the spatial difference map. Other weight organizations not shown are qualitatively similar. Although this result is less smooth than many conventional SOMs (e.g., Fig. 6d), because multi-winner activation is used and the maps are trained for limit cycles, its appearances are quite similar to some cortical maps in biological neural systems, such as maps in cats' visual cortex (for example, see [46, Fig. 1]). Limit cycles varied in length from 2 to 12 or higher; see Sect. S3 of the online Supplementary Material. Although it is possible to learn longer limit cycle attractors, fixed-points attractors, or even chaotic dynamics using different parameter values, their properties were found to be less desirable [21].

Table 2: Summary of performance results. Numbers in parentheses indicate standard deviations.

	All test data (1000 targets)			Typical workspace (800 targets)		
	spatial error	error in cm	#iterations	spatial error	error in cm	#iterations
pre-training	.551 (.0807)	117.1 (17.96)	1000 (0)	.566 (.0835)	119.9 (18.44)	1000 (0)
post-training	.011 (.0024)	2.2 (.49)	554 (21.6)	.003 (.0007)	.7 (.14)	505 (18.8)
open-loop only	.083 (.0031)	16.7 (.67)	1000 (.2)	.084 (.0036)	16.9 (.76)	1000 (.3)
closed-loop only	.025 (.0022)	5.2 (.52)	570 (22.5)	.009 (.0034)	1.9 (.74)	493 (25.4)
conventional SOMs	.194 (.0254)	39.1 (5.84)	972 (7.0)	.175 (.0281)	35.4 (6.35)	966 (8.6)

3.2 Arm performance and trajectory

As described in Sect. 2.3, the 1000 spatial targets used for testing are generated by sampling a grid in the 3D joint angle space. The performance against the test data can serve to indicate how well the architecture generalizes to new inputs, since test inputs are very unlikely to be in the training data, which were generated randomly. The arm is initialized using the same joint angles for all test targets (illustrated using thick line segments in Fig. 7). For each test target, the architecture is run for a maximum of 1000 iterations to drive the arm toward the target. Whenever the spatial error $\|\boldsymbol{X}^* - \boldsymbol{X}\|$ has been maintained below a predetermined threshold $\epsilon = 0.001$ for 20 consecutive iterations, the arm is considered to have reached the target. In this case the execution is terminated early. At the end of reaching each target, the spatial error and the number of iterations required to reach termination are recorded to measure the accuracy and speed of a reaching movement. The whole workspace of the arm is linearly transformed into a unit cube, i.e., x, y, z coordinates are each within $[0, 1]$. This allows measuring spatial error relative to workspace size and independent of actual dimensions of the arm. For size reference, the extent that a Baxter robot's arm can reach, inferred from a 3D model, are roughly of length 170 cm, 235 cm, and 218 cm in the x, y, and z directions, respectively. A fully extended arm is about 137 cm long (see Fig. 2).

The average spatial error after training is 0.011 (SD=0.0024), decreasing from 0.551 (SD=0.0807) before training (see Table 2). This corresponds to 1.1% of the length of each workspace axis, or 2.2 cm in a Baxter robot's physical space. The average number of iterations required is decreased to 554 (SD=21.6) from the maximum 1000 before training. For comparison purposes, a control experiment was conducted where all limit cycle SOMs in the architecture were replaced by conventional SOMs while other conditions were kept the same. The results in Table 2 indicate higher spatial errors and greater numbers of iterations. Fig. 7 plots spatial errors for all test targets after

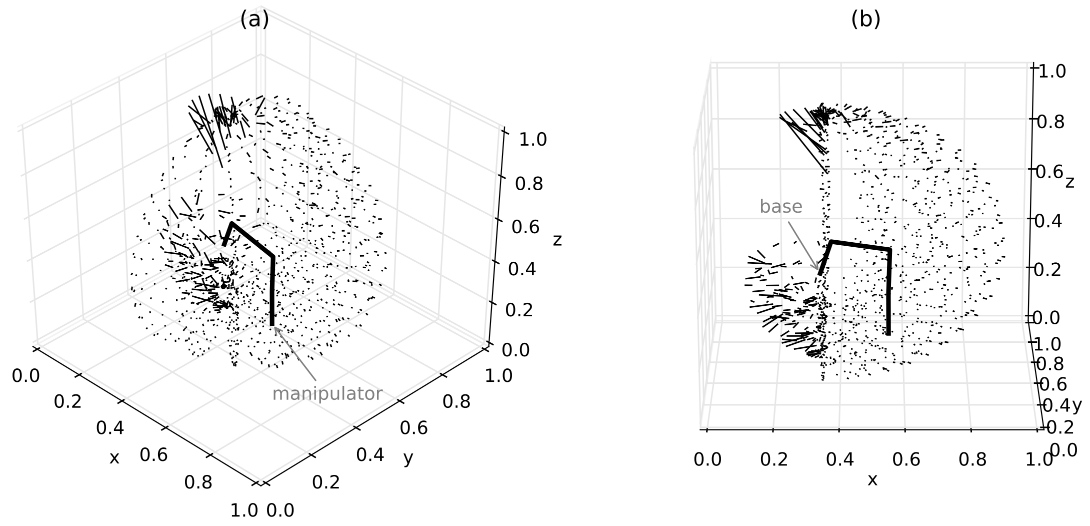

Figure 7: Distribution of spatial errors seen from two different view angles. Each line segment connects a target location and a corresponding final manipulator location. Longer line segments indicate greater spatial errors. The thick line segments illustrate the starting position of the arm.

training of the current model. Each line segment connects a spatial target and the corresponding final location of the manipulator. Therefore, shorter line segments indicate more accurate reaching. There are clearly two 3D regions where the errors are larger. A sideways view (see Fig. 7b) reveals that these two regions are behind and below the base, and near the top, which are relatively difficult to reach. Targets in these regions are likely to require difficult arm positions, i.e., they lie around "singularities", which require joint positions that make moving in certain spatial directions difficult if not impossible [16]. However, these regions are not a usual workspace in a typical robotic application. Unlike with the simulations done here that attempt to include all possible positions, the physical space behind and below the arm base is typically occupied by a robot's torso and thus not accessible in any event. Objects to be acted on are typically placed in front of a robot at a moderate height, and therefore excluding these inaccessible region better reflects more practical situations. Specifically, if we exclude 200 targets whose x coordinates are less than that of the arm base (i.e., behind the arm base), as well as those with $z > 0.96$, the average post-training spatial error for the remaining 800 targets in a typical workspace drops to 0.003 (SD=0.012), which corresponds just 0.7 cm in the Baxter robot's scale.

Fig. 8 shows a typical arm trajectory when reaching for one of the test targets. The spatial trajectory of the manipulator is smooth without notable jerks (Fig. 8a). Fig. 8b indicates that the spatial error decreases drastically from 0.7 to below 0.03 in early iterations (e.g., first 150 iterations), during which the open-loop subsystem dominates (refer to Fig. 3). From there on, the closed-loop outputs start to out-weigh the open-loop ones to perform fine-tuning for time until the target spatial error $\epsilon = 0.001$ is met. Fig. 8c–d show the joint angles and joint velocities during reaching. The joint angle trajectory appears to be overall smooth and sigmoid-shaped, and the velocity profiles are single-peaked and near bell-shaped. This is comparable to and consistent with the results in past studies focusing on human arm movements, which suggests that joint trajectories are highly stereotyped and contain invariant spatio-temporal features including sigmoid joint displacement and bell-shaped velocity profiles [14, 15].

In a neural architecture based on changing activity, control timing may greatly affect system performance and therefore undermines the robustness of the architecture. While the above results are obtained using certain values for the timing parameters t^{assoc}, t^{filter}, and t^{out} (Table 1), systematic experiments demonstrated that our architecture is quite tolerate of substantial changes in the timing parameters as long as $t^{out} \geq t^{assoc} + t^{filter}$ and $t^{filter} > 1$; see Sect. S4 of the online Supplementary Material for details.

3.3 Standalone open-loop and closed-loop subsystems

To better understand the individual contributions of the open-loop and the closed-loop subsystems after training, each of the subsystems is disabled in turn, forming an open-loop-only[‡] system and a closed-loop-only system. Specifically, the open-loop-only system is obtained by setting the modulatory weight functions $b^c_{open}(n) = 0$ and $b^o_{open}(n) = b^o(n) + b^c(n)$. The latter is for maintaining the same total modulatory weight as that of the full architecture, so that the results are fairly comparable. Similarly for the closed-loop-only system, $b^o_{closed}(n) = 0$ and $b^c_{closed}(n) = b^o(n) + b^c(n)$. The same test data are used to compare the open-loop-only and closed-loop-only systems against the full architecture.

Fig. 9 shows the arm trajectory driven by the open-loop-only system, where the initial arm

[‡]As we noted earlier, the open-loop-only system is not a purely open-loop system by definition, since the current joint angles $\boldsymbol{\Theta}$ are used in the motor output filter (Eq. 13). Our use of "open-loop-only" in this context is only intended to be a convenient label, to be consistent with the terminology of our earlier paper [20] and to signify the fact that the manipulator's position $\boldsymbol{X}$ is never fed back, which is more relevant in inverse kinematics problems.

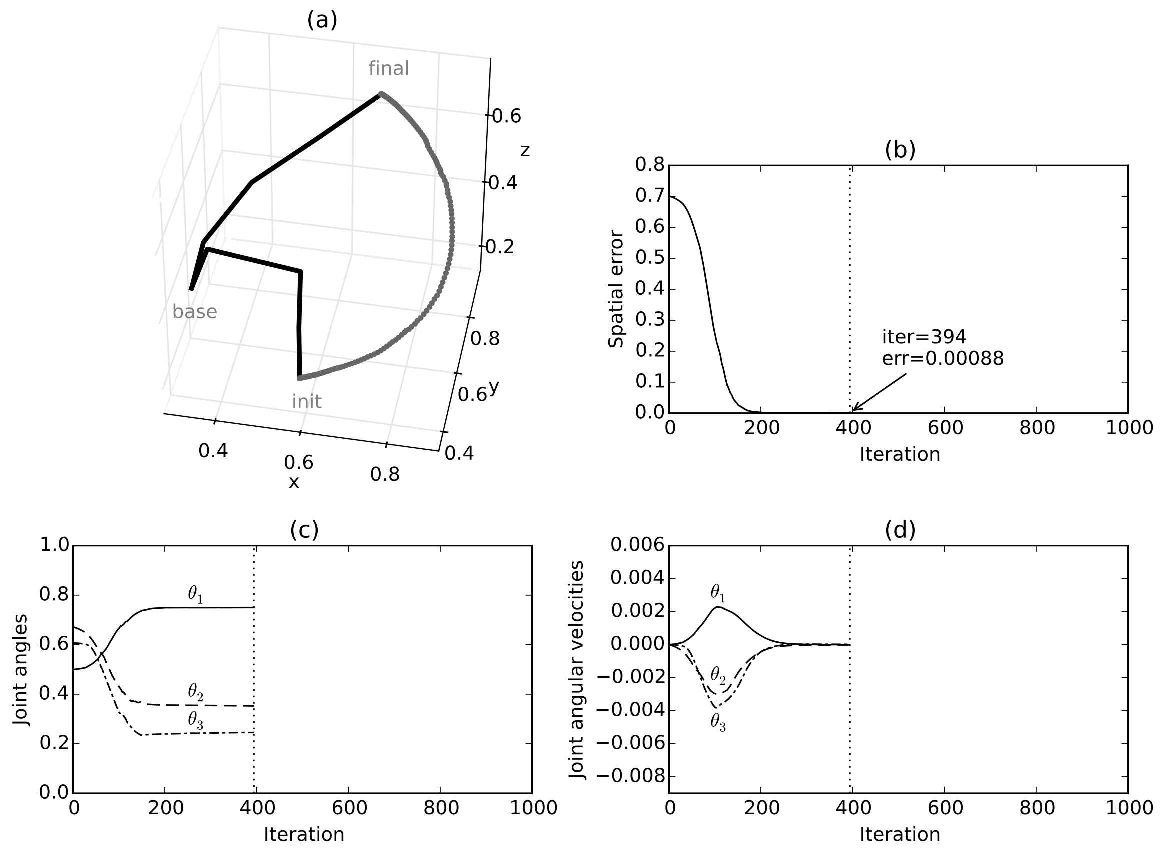

Figure 8: Arm trajectory for a typical reaching movement. (a) The spatial representation of the trajectory, where the initial and final arm positions are drawn using line segments. The label "base" indicates the fixed base of the arm. The labels "init" and "final" mark the initial and final manipulator locations. The manipulator location at each iteration is plotted using a dot, forming a curve connecting "init" and "final". The 3D coordinates are the result of linearly transforming each axis of the arm-reachable Cartesian space to $[0, 1]$. (b) The spatial error at each iteration. The final iteration and spatial error are labeled at the bottom right. (c) The joint angles of the arm at each iteration, where the three joints θ_1, θ_2, and θ_3 correspond to those in Fig. 2. The joint angles are represented by linearly transforming the whole angular range of each joint to $[0, 1]$. (d) The angular velocities of the joints at each iteration, low-pass filtered. The vertical lines in (b)–(d) indicate the terminating iteration.

position and the spatial target are the same as those used in Fig. 8. The arm trajectory is similar to that of the full architecture (Fig. 8), except that the trajectory is smoother and that the final spatial error 0.073 is substantially larger. The smoothness is quantified using jerk cost, where a smaller value indicates a smoother trajectory [12]. The values are listed in Table 3 for comparison. The smoother trajectory produced by the open-loop-only system is the result of lacking movement regulations that are normally generated by the closed-loop subsystem. The low accuracy is ex-

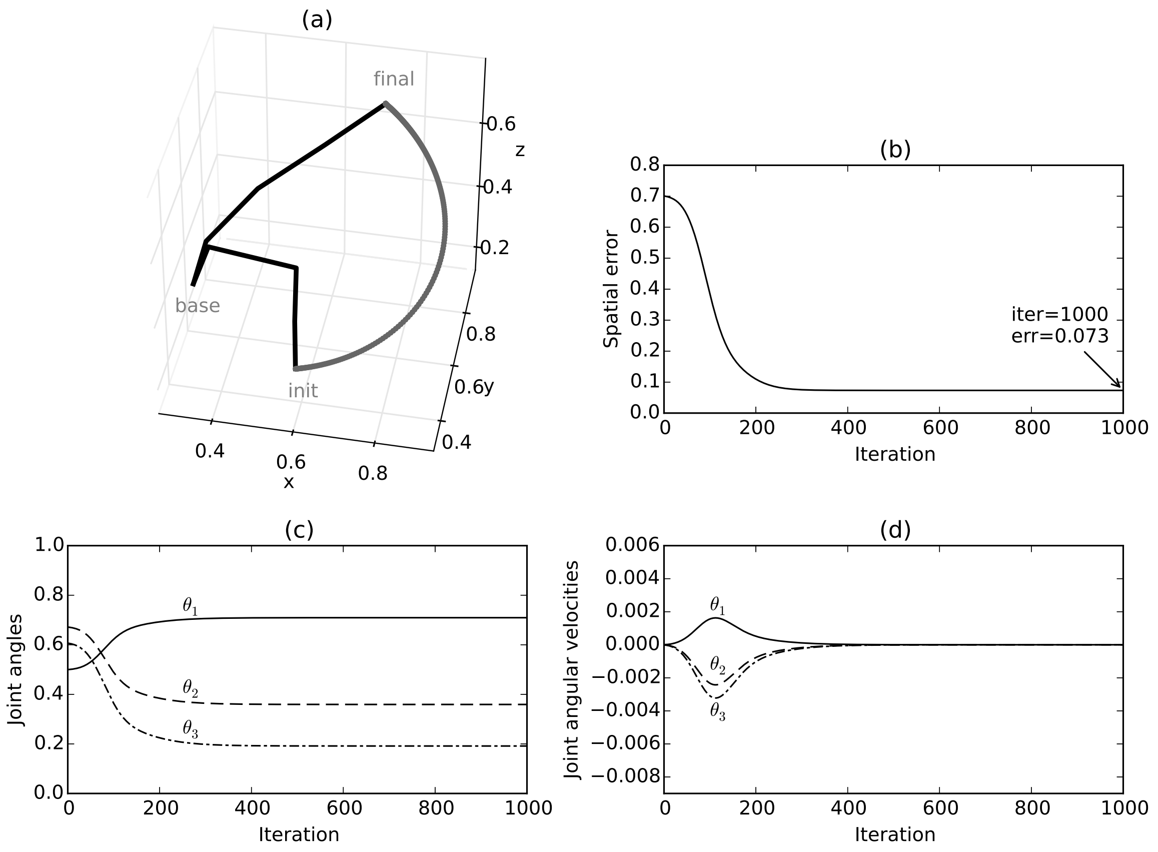

Figure 9: Arm trajectory using the open-loop subsystem alone. Same arrangement as in Fig. 8. While this trajectory is marginally smoother than in Fig. 8, the final spatial error is much worse.

pected since the open-loop subsystem here does not have a perfect internal representation, and thus the movement is likely to be erroneous without being regulated by error feedback along the trajectory [20, 26]. On the other hand, Fig. 10 shows the arm trajectory driven by the closed-loop-only system, where the initial arm position and the spatial target remain the same. Although $\epsilon = 0.001$ is not met, and thus the maximum number of iterations is used, the final spatial error is relatively low. However, the spatial trajectory is much jerkier (see Table 3). The joint angle plots (Fig. 10c–d) also contrast with Fig. 8, showing apparent signs of non-monotonic movement towards the target. This can be observed by the peaks in the joint angle plot, as well as the velocity profiles having multiple (positive and negative) peaks. This jerky motion is likely caused by the closed-loop subsystem's constantly regulating movement to compensate for the lack of an initial "push" that is normally provided by the open-loop subsystem [26]. Further analysis of this trajectory instability is given in Sect. S5 of the online Supplementary Material. The superiority of the full system over

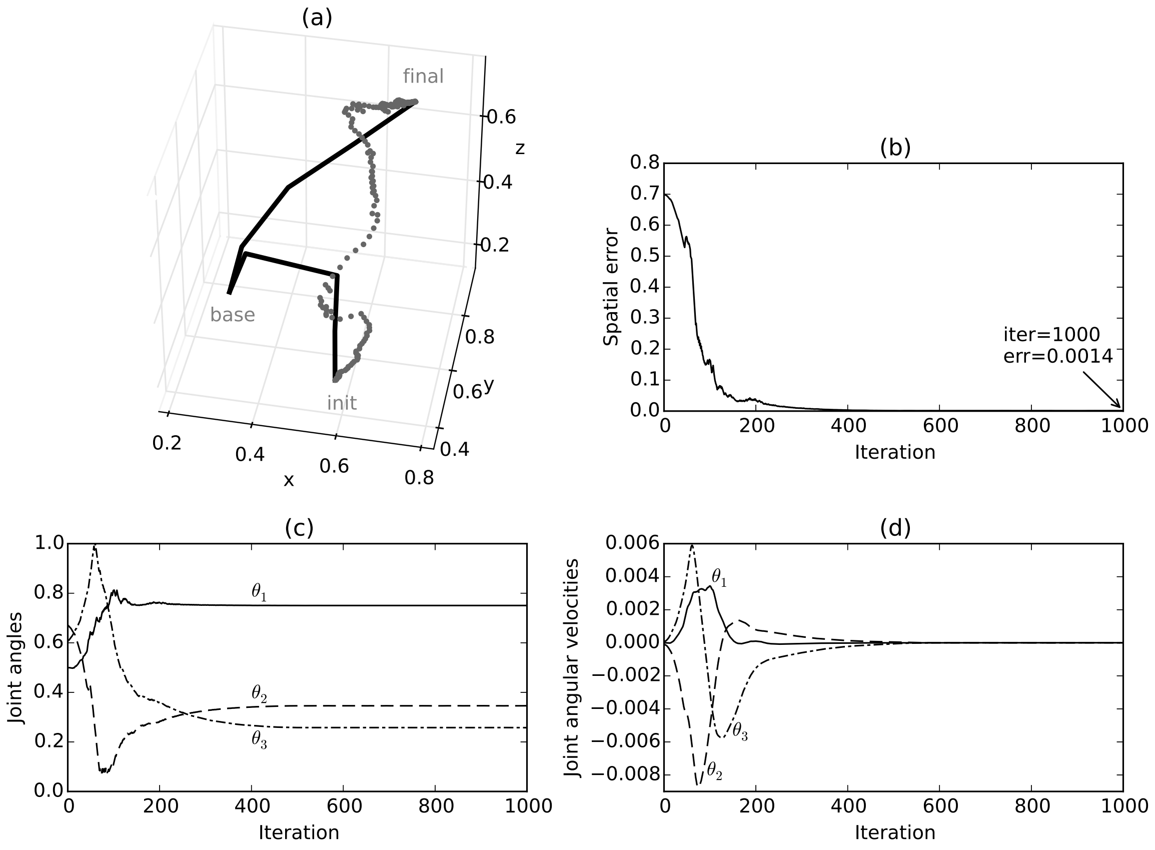

Figure 10: Arm trajectory using the closed-loop subsystem alone. Same arrangement as in Fig. 8. While this trajectory has a final spatial error comparable to that of Fig. 8, the smoothness here is much worse.

both the open-loop-only and closed-loop-only versions of our architecture persisted over a broad range of values of the target spatial error ϵ (see Sect. S6, online Supplementary Material).

3.4 Performance in the presence of noise

Real-world control systems routinely operate under many sources of noise. Inspired by that fact, and that our architecture is based on limit cycle attractor states, which by definition can resist certain degrees of perturbation, this section studies how our neural architecture based on limit cycle attractors responds to internal and external interference. To this end, three types of internal and external disruption are introduced: transient perturbation to activity in each of the four maps, permanently disabling/"damaging" map nodes, and sudden perturbations of arm positions that simulate unexpected external forces.

Table 3: Jerk costs of arm trajectories. Smaller values indicate smoother trajectories.

Description	Figure	Jerk Cost
Normal architecture	Fig. 8a	1.0×10^{-4}
Open-loop-only	Fig. 9a	5.2×10^{-9}
Closed-loop-only	Fig. 10a	7.2×10^{-3}
Damaged spatial map	Fig. 12a	8.0×10^{-3}
Damaged spatial difference map	Fig. 12b	7.3×10^{-2}
Physical robot*	Fig. 15a	2.2×10^{-4}

*The raw data are low-pass filtered.

First, we study the effects of transient activity noise in our architecture by temporarily perturbing activity in each map (spatial, joint angle, spatial difference, joint angle difference maps), and observing how perturbation to each of these maps affects spatial error. When a map activity pattern is being perturbed, the activity of each node is given a probability to flip from 0 to 1 or 1 to 0 (i.e., making active nodes inactive and vice versa), where the probability of node activity flipping is a parameter. Perturbation to a map is temporary, meaning that for each iteration, perturbation occurs only at a predetermined time step. For the spatial and spatial difference maps, the perturbation time is set to be at $t = 30$ for two reasons. One is that since these maps start receiving inputs at $t = 0$, by $t = 30$ their activity is very likely to have entered a limit cycle. The other reason is that by $t = t^{assoc} = 50$, their downstream maps start to read from them, so this allows them 20 time steps to recover from noise. For similar reasons, the perturbation time for the joint angle and the joint angle difference maps is set to be $t = 80$, which is 30 time steps after they start receiving inputs.

Fig. 11a shows the spatial errors as a result of perturbing different maps with different activity flipping probabilities, where the initial arm position and the spatial target are the same as those used in Fig. 8. Each data point in the figure is the average of 100 independent simulations with different random seeds for perturbation. Polynomial curves are fitted to the data points to highlight the trends. Overall, activity noise in the spatial and spatial difference maps causes much greater impact to the performance, compared with noise in the joint angle and joint angle difference maps. A reason for this is that the latter receive inputs continuously from their upstream maps, which helps mitigate their own activity noise, while the former receive inputs only briefly. For all maps, the spatial error tends to be greater as the activity flipping probability is in the mid-range, when

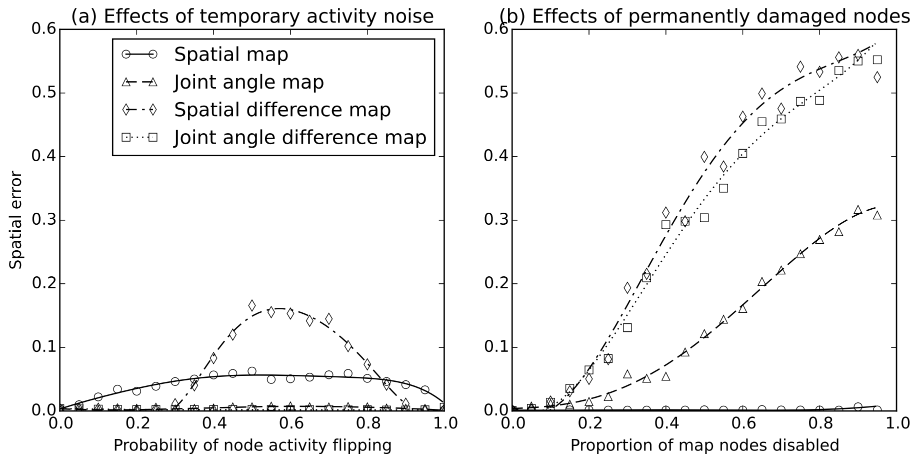

Figure 11: Effects of internal interference. (a) Spatial error as a result of perturbing activity in each map. The horizontal axis represents the probability that node activity is flipped from 0 to 1 or 1 to 0 during perturbation. (b) Spatial error as a result of permanently disabling map nodes, using the same notations as in (a). In both figures, each data point indicates the average of 100 simulations of random activity perturbation or random selections of disabled nodes. Curves are polynomially fitted to the data points to aid interpretation.

the uncertainty about a node's activity is the highest. Note that very high probability of activity flipping also decreases spatial error. This can be understood as follows. Consider an extreme case where the activity flipping probability is 1, and where all winner nodes take on value 0 and non-winner nodes value 1. Since topographical recurrent connections of each node covers its local neighborhood except for itself, in the next time step after activity flipping, a previous winner node is guaranteed to receive all 1 recurrent inputs, while a previous non-winner node may receive some 0 recurrent inputs for the presence of winner nodes in its neighborhood (which is very likely). As a result, a winner node in an inverted activity pattern will almost always win in the next time step because all recurrent weights are positive. This cancels the perturbation. Another observation from Fig. 11a is that even a small activity flipping probability in the spatial map is enough to cause the spatial error to rise. The maximum spatial error it causes is limited at around 0.05. On the contrary, spatial error is rather insensitive to small activity flipping probability in the spatial difference map, but once the probability exceeds around 0.3, spatial error rises more rapidly and may reach above 0.15, which is poor but not devastating compared with pre-training error around 0.55 (Table 2). Perturbation to the joint angle map and the joint angle difference map have much less effects on spatial error. A reason is that they receive inputs continuously from their upstream maps, which help mitigate their activity noise, while the other two maps receive inputs only briefly.

Next, consider the impact of permanently disabling map nodes. A disabled map node always has a zero activity level, and does not compete with its neighbor nodes during the multi-winners-take-all process. Compared with activity perturbation, which is a transient effect, disabled map nodes remain so in all time steps throughout a reaching movement, and thus cause a much larger disruption to the architecture. In this experiment, the initial arm position as well as the target are again the same as used in Fig. 8. Fig. 11b summarizes the results of disabling different proportions of map nodes, where each data point is the average of 100 simulations, each having different random map nodes disabled. Disabling nodes in the spatial difference map and the joint angle difference map is the most detrimental, causing the spatial error to rise rapidly above 0.1 (a borderline acceptable value) when 30% of the nodes or more are disabled. Both of these maps belong to the closed-loop subsystem that is more essential to spatial accuracy. Note that since the joint angle map also participates in generating closed-loop outputs, disabling nodes in the map also cause the spatial error to rise, although less rapidly. The error rises above 0.1 only when 50% of the joint angle map nodes or more are disabled. Disabling nodes in the spatial map has little effect on spatial errors, since the spatial map is related to open-loop outputs only, and as discussed in Sect. 3.3 (see also Table 2), the architecture can still achieve high accuracy without open-loop outputs, although this also leads to jerky trajectories.

Arm trajectories as a result of an impaired open-loop or closed-loop subsystem can be illustrated by permanently damaging 30% of the spatial map nodes or 30% of the spatial difference map nodes, respectively. The resulting arm trajectories are shown in Fig. 12, where the initial arm position and the spatial target are kept the same as used in Fig. 8. When the open-loop subsystem alone is impaired, the spatial trajectory appears to be somewhat S-shaped, indicating signs of overshooting. The spatial error decreases relatively slowly in the first 100 iterations, during which the movement is primarily guided by the open-loop subsystem. The intact closed-loop subsystem then takes over, and the error eventually drops to a relatively low value. On the other hand, when the closed-loop subsystem alone is impaired, the spatial trajectory is smooth in early iterations, and then the manipulator starts to exhibit a "tremor" towards the end of the trajectory. The spatial error in the first 100 iterations decreases quickly (at a similar rate to Fig. 8) since the manipulator is guided primarily by the intact open-loop subsystem. Then the error starts to oscillate and remains at

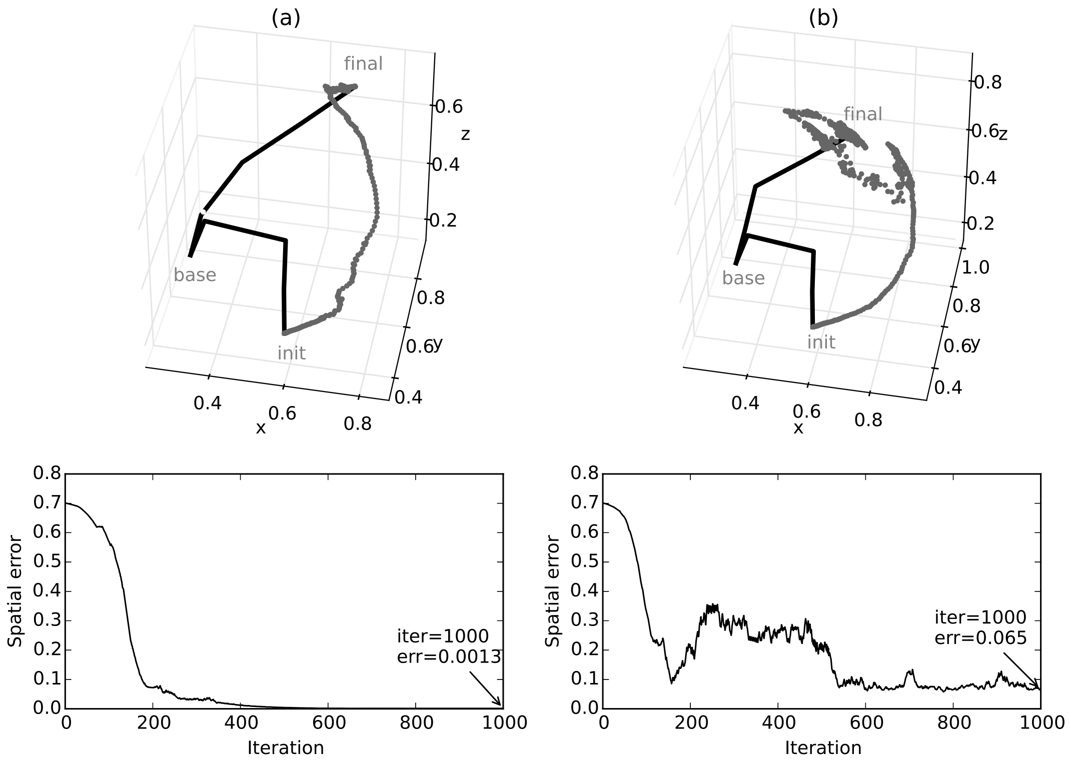

Figure 12: Arm trajectories when 30% of random nodes are disabled in the (a) spatial or (b) spatial difference map. They each indicate how the neural architecture performs with an impaired open-loop or closed-loop subsystem, respectively. The top row shows the spatial trajectories of the manipulator, and the bottom row shows the spatial error at each iteration.

relatively high values.

Finally, we study the effects of applying abrupt external forces to the arm, which is simulated by adding a random perturbation vector to the arm joints at a given iteration. With the angular range of each joint being normalized to be in $[0, 1]$, the length of each random vector is fixed at 0.1, or 10% of each axis length in the joint angle space. The question being asked here is whether the arm can resume reaching while maintaining a low spatial error in the presence of such an external perturbation. The initial arm position and the target are again made the same as in Fig. 8, such that the results shown in Fig. 8 can serve as a control for comparison. Fig. 13 shows the results of applying external perturbations at different iterations during a reaching movement: (a) at iteration 100 when the open-loop subsystem dominates, (b) at iteration 150 when both subsystems are weighed about equally, and (c) at iteration 350 when the closed-loop subsystem dominates. From

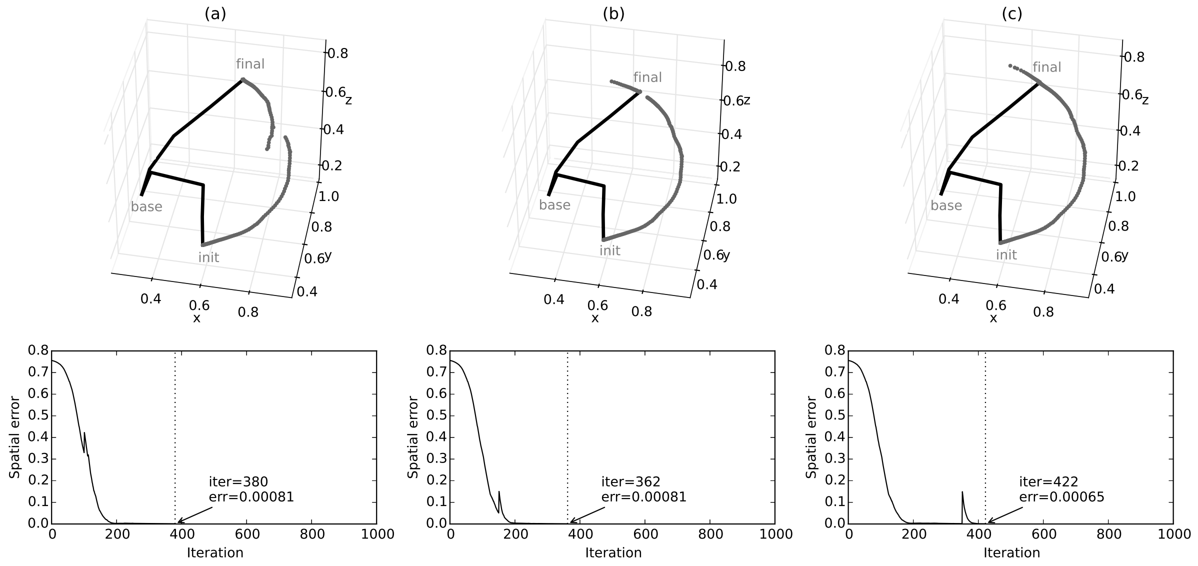

Figure 13: Arm trajectories for abrupt arm joint perturbation during a reaching movement. Columns (a), (b), and (c) show the results of perturbing at iterations 100, 150, and 350, respectively. The top row plots the spatial trajectories of the manipulator. The discontinuities of the trajectories indicate where the perturbations occur. The bottom row shows the spatial error in each iteration.

the spatial trajectory (top row), the exerted perturbations do not greatly affect the smoothness of the trajectory. From the spatial error plots (bottom row), as soon as a perturbation occurs, the spatial error can be quickly reduced to a normal level within a few tens of iterations. The final spatial error and the number of iterations spent are not substantially affected, either. This result suggests robustness of the neural architecture against unexpected external perturbations, regardless of when they occur.

4 Physical robot implementation results

For validation purposes and for assessing the applicability of our neural architecture, we also studied its use in controlling the movements of a Baxter robot's arm. Our goal here is very limited: just to demonstrate that the neural architecture we described and trained using a simulated robot as above, could be ported to and run effectively on a physical robot *without any further training.* Two shoulder joints, S0 (roll) and S1 (pitch), and an elbow joint, E1 (pitch), of the robot's left arm are respectively mapped to $\theta_1, \theta_2, \theta_3$ of the neural architecture, while other joints are fixed at 0, forming a 3-DOF arm as schematized in Fig. 2 and studied in the simulations above. A commodity workstation is used to host both the neural architecture and Robot Operating System (ROS) [41], which serves as a communication medium between our neural control architecture and the robot. Through ROS, the neural architecture obtains joint angles and issues joint velocity commands at a target polling rate of 60 Hz. While the robot is equipped with cameras, since our interest in this study is not machine vision, these cameras are not used to determine spatial locations. Instead, the target location is specified and input manually, and the manipulator location is computed based on joint angles (reported by the robot's sensors) using the Denavit-Hartenberg method [17, p.435]. The neural architecture is trained using data generated by the ideal arm model (Sect. 2), and is then used to control the robotic arm without further training on the robot itself. Unlike the arm model, the mechanical components and sensors of the robotic arm may introduce errors and latencies in measuring the joint angles and in executing joint velocity commands.

Fig. 14 shows results of the robot arm performing three reaching movements. The initial arm postures and the spatial targets of the movements are distinct for each movement. The first two movements are center-out reaching, where the arm starts at positions close to the torso and reaches out to an above-shoulder target and a shoulder-height target. The third movement starts from one side of the torso and ends in the front. The trajectories of manipulator locations show smooth curved paths from the starting points to the targets. The final spatial errors for the three movements are 0.008 (1.3 cm), 0.002 (0.3 cm), and 0.001 (0.1 cm).[§] Since the first two movements do not satisfy the stopping criteria $\epsilon = 0.001$, they both take the maximum of 1000 iterations. The third movement takes 899 iterations. While this performance is not as good as running on the simulated ideal arm model, it is sufficient for typical Baxter robot applications (i.e., considering that Baxter robots are not specialized in high mechanical accuracy), and could potentially be improved with further training on the physical robot, e.g., further Stage 2 training with smaller learning rate to compensate for the motor imprecision.

Fig. 15 shows the detailed joint angle and velocity trace of the first arm movement, which uses the same initial position and the same spatial target as the experiments shown in Fig. 8. The trajectory is qualitatively similar to Fig. 8, showing a sigmoid-shaped joint angle trace and a

[§]Since these spatial errors are computed based on the joint angles, their accuracy depends on the accuracy of the joint sensors.

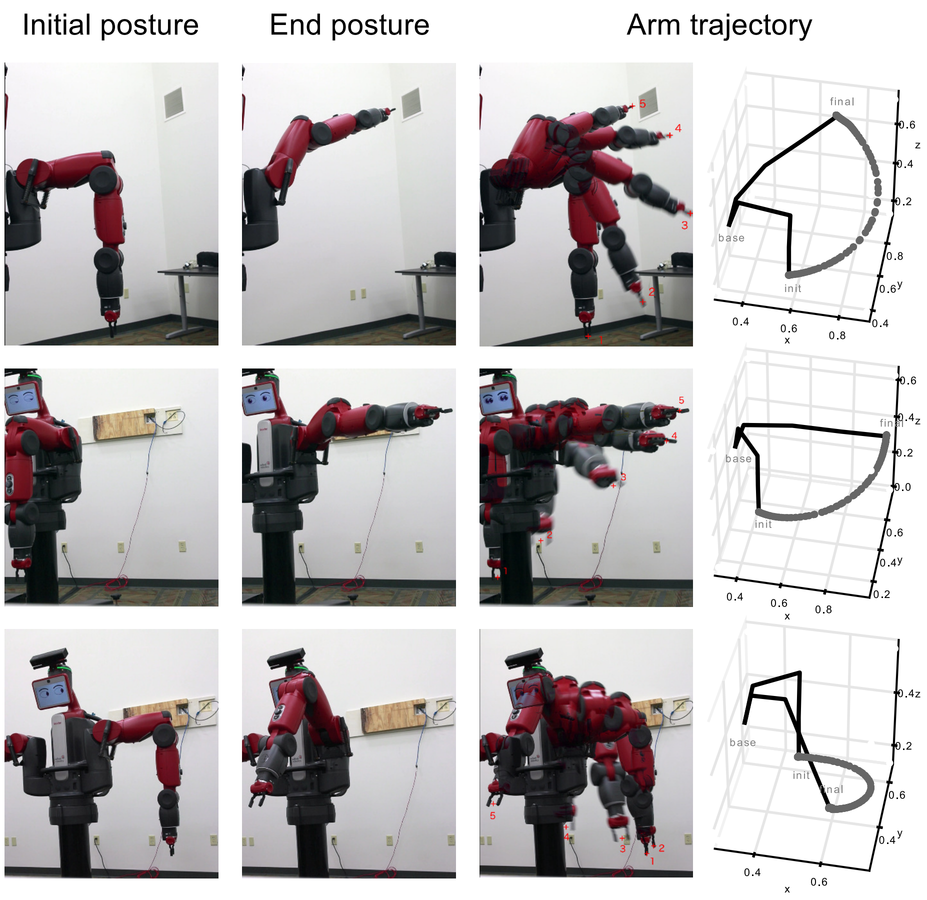

Figure 14: Three trajectories of a physical robotic arm. Each row shows the results of a different pair of initial arm position and spatial target. The first two columns show the initial and final arm positions of the robot. The third column shows superimposed images illustrating the trajectory of the robot arm. See online Supplementary Material for the video. The right-most column plots the manipulator locations using joint angle sensor history recorded from the robot in a fashion like the earlier figures in this paper.

single-peaked, close to bell-shaped, velocity profile, although slightly not as smooth (see Table 3). In summary, these results indicate that the neural architecture, *despite being trained with an ideal arm model*, can successfully operate a Baxter robot's arm without further training.

5 Discussion

Our goal in this study was to explore the possibility of using limit cycle representations to perform practical neural computations, given that limit cycle SOMs have been shown previously to have some clear advantages over conventional SOM representations in terms of information encod-

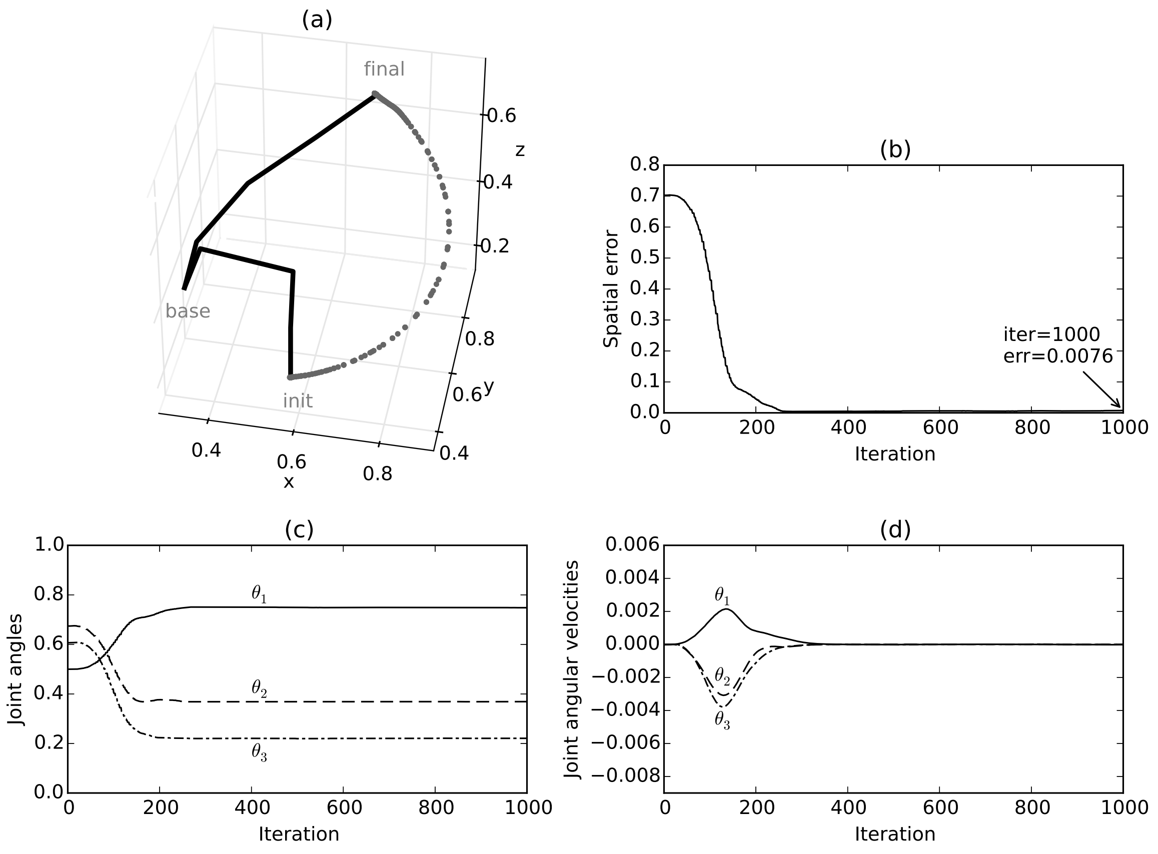

Figure 15: Arm trajectory of the robot. The initial arm position and the spatial target are the same as those used in Fig. 8 as well as in Fig. 14a. Figures (a)–(d) are again arranged in the same way as in Fig. 8. Data are recorded from the joint angle sensors of the robot.

ing [21]. While the intention was not to build a realistic model of biological motor control, our architecture was inspired by some neurophysiological findings, including self-organization of cortical maps, rhythmic neural oscillations, and different feedforward/feedback modes of human motor control.

Our primary finding was that our architecture for inverse kinematics arm control produced well-formed topographic maps in the limit cycle SOMs during learning, and was able to learn associations between limit cycle SOMs that generalized to new spatial targets that were never seen during training. Learning such associations for limit cycle activity is a much more challenging task than with static single-winner activity. This is because each limit cycle contains multiple activity patterns and each activity pattern contains multiple winning nodes that have to be associated with a target spatial location. Further, each of those winning nodes may participate in encoding many other spatial targets.

Our work also demonstrated the robustness of a limit cycle architecture. We found that when a part of the neural system was disabled or disrupted in a variety of ways, the overall functionality was impaired but not completely destroyed. The architecture's operation was found to be independent of the exact timing of individual limit cycles (i.e., their start time and length), so that the overall neural architecture does not need to explicitly detect onset of limit cycles. We also found three other key results. First, in spite of the fact that the internal activity of the system is oscillatory, its outputs can be stabilized so as to drive the arm to a fixed target in a smooth trajectory without apparent oscillations and then maintain it at that position. Second, the concurrent open-loop and closed-loop methods, each having its own limitations in arm movement control when used in isolation, together collectively achieve a precise and smooth reaching movement. Finally, although our work is solely from an engineering perspective, it was interesting to discover that the results of disabling different parts of the architecture are reminiscent of symptoms seen with human patients. We also observed that when our architecture was ported to a physical robot, it was able to control the robot arm to reach targets fast, accurately, and smoothly without further training, in spite of the inevitable mechanical latency and imprecision this produced.

In our architecture, outputs of the open-loop and closed-loop subsystems are weighted such that relative influence is progressively shifted from the open-loop to the closed-loop subsystem during a movement. This was inspired by the human motor control system that combines two control modes: feedforward and feedback [26, 28, 42]. The feedforward mode is particularly critical in the early stages (up to ∼50–100 ms) of a movement since the sensory feedback from the periphery cannot be fully processed and thus employed by the supra-spinal control system to regulate the movement [31]. Conversely, the feedback mode is employed during the later stages of movement where the sensory feedback has time to be processed, and this allows for movement regulation by supra-spinal control centers for dealing with perturbations and/or accuracy constraints on the final position (beyond ∼50–100 ms) [31].

In addition to their computational advantages over conventional SOMs, limit cycle SOMs are, in a very limited sense, more reminiscent of biological cortical activity than conventional SOMs, in that they involve sparsely coded multi-focal activation and rhythmically oscillating dynamics [6, 9, 11]. Resemblance can also be drawn between the sustained limit cycle dynamics and working memory, both temporarily "remembering" transient stimuli in the form of sustained neural activity for subsequent processing. Further, in relation to the simple gradient map formed in a conventional SOM, the semi-repetitive topographic map formed with a limit cycle SOM is qualitatively similar to some biological visual cortex maps.

In this work we were not trying to create a veridical model of biological motor control. Accordingly, there are many aspects of our architecture that do not capture critical mechanisms present in biological motor control, such as internal prediction models and reward modulated sustained activity for sensory-motor contingencies (e.g. [10, 39]). Still, it is of interest to observe that the results of partially damaging the open-loop or closed-loop maps are somewhat reminiscent of human behaviors. For example, it has been argued that biases in biological "internal models", which purportedly guide movements without sensory feedback, result in overshooting or undershooting of the movements [4, 34]. Somewhat similar behaviors were observed with our architecture with the damaged open-loop subsystem. In addition, when the close-loop subsystem was damaged, a "tremor" appeared late during movements upon approaching the target location. This instability is interesting because it behaviorally resembles human intention (cerebellar) tremor seen at the end of visually guided movements, and because it was neither built in intentionally nor anticipated by the authors prior to computational experiments. Computationally, the late tremor produced by our model is due to the motor output filter placing more weight on the outputs of the damaged closed-loop subsystem in late stages of an arm movement. Of course, for this observation to become of any interest as an analog of human motor behavior would require a more thorough future study involving the use of biologically realistic neural models of the relevant brain structures.

Ackowledgement

This work was supported by ONR award N000141310597.

References

[1] Angulo, V., Torras, C. (2008). Learning inverse kinematics: Reduced sampling through decomposition into virtual robots. *IEEE Trans. Syst. Man, Cybern. B, Cybern.*, *38*(6), 1571–1577.

[2] Barreto, G., Araújo, A., & Ritter, H. (2003). Self-organizing feature maps for modeling and control of robotic manipulators. *J. Intelligent and Robotic Systems*, *36*(4), 407–450.

[3] Bednar, J. & Miikkulainen, R. (2000). Tilt aftereffects in a self-organizing model of the primary visual cortex. *Neural Computation*, *12*(7), 1721–1740.

[4] Bhanpuri, N., Okamura, A., & Bastian, A. (2014). Predicting and correcting ataxia using a model of cerebellar function. *Brain*, *137*(7), 1931–1944.

[5] Bullock, D., Grossberg, S., & Guenther, F. (1993). A self-organizing neural model of motor equivalent reaching and tool use by a multijoint arm. *J. Cognitive Neuroscience*, *5*(4), 408–435.

[6] Buzsaki, G. (2006). *Rhythms of the Brain.* Oxford University Press.

[7] Chaumette, F. & Hutchinson, S. (2006). Visual servo control. I. Basic approaches. *IEEE Robotics Automation Magazine*, *13*(4), 82–90.

[8] Chen, Y. & Reggia, J. (1996). Alignment of coexisting cortical maps in a motor control model. *Neural Computation*, *8*(4), 731–755.

[9] Churchland, M., Cunningham, J., Kaufman, M., Foster, J., Nuyujukian, P., Ryu, S., & Shenoy, K. (2012). Neural population dynamics during reaching. *Nature*, *487*(7405), 51–56.

[10] Duff, A., Fibla, M., & Verschure, P. (2011). A biologically based model for the integration of sensory–motor contingencies in rules and plans: A prefrontal cortex based extension of the Distributed Adaptive Control architecture. *Brain Research Bulletin*, *85*(5), 289–304.

[11] Fell, J. & Axmacher, N. (2011). The role of phase synchronization in memory processes. *Nature Reviews Neuroscience*, *12*(2), 105–118.

[12] Flash, T. & Hogan, N. (1985). The coordination of arm movements: an experimentally confirmed mathematical model. *J. Neuroscience*, *5*(7), 1688–1703.

[13] Gaskett, C. & Cheng, G. (2003). Online learning of a motor map for humanoid robot reaching. In *Int. Conf. CIRAS.*

[14] Gentili, R., Han, C., Schweighofer, N., & Papaxanthis, C. (2010). Motor learning without doing: Trial-by-trial improvement in motor performance during mental training. *J. Neurophysiology*, *104*(2), 774–783.

[15] Gentili, R., Oh, H., Huang, D.-W., Katz, G., Miller, R., & Reggia, J. (2015). A neural architecture for performing actual and mentally simulated movements during self-intended and observed bimanual arm reaching movements. *Int. J. Social Robotics*, *7*(3), 371–392.

[16] Guenther, F. & Barreca, D. (1997). Neural models for flexible control of redundant systems. In P. Morasso & V. Sanguineti (Eds.), *Self-organization, Computational Maps, and Motor Control* (pp. 383–421). North-Holland.

[17] Hartenberg, R. & Denavit, J. (1965). *Kinematic synthesis of linkages.* McGraw-Hill.

[18] Hua, H. (2016). Image and geometry processing with oriented and scalable map. *Neural Networks*, *77*, 1–6.

[19] Huang, D.-W., Gentili, R., & Reggia, J. (2014). Limit cycle representation of spatial locations using self-organizing maps. In *IEEE Symp. CCMB*, (pp. 79–84).

[20] Huang, D.-W., Gentili, R., & Reggia, J. (2015a). A self-organizing map architecture for arm reaching based on limit cycle attractors. In *EAI Int. Conf. BICT*.

[21] Huang, D.-W., Gentili, R., & Reggia, J. (2015b). Self-organizing maps based on limit cycle attractors. *Neural Networks*, *63*, 208–222.

[22] Huang, D.-W., Katz, G., Langsfeld, J., Gentili, R., & Reggia, J. (2015). A virtual demonstrator environment for robot imitation learning. In *IEEE Int. Conf. TePRA*.

[23] Hutchinson, S., Hager, G., & Corke, P. (1996). A tutorial on visual servo control. *IEEE Trans. Robot. Autom.*, *12*(5), 651–670.

[24] Igel, C. & Hüsken, M. (2003). Empirical evaluation of the improved Rprop learning algorithms. *Neurocomputing*, *50*, 105–123.

[25] Iwasaki, T. & Furukawa, T. (2016). Tensor SOM and tensor GTM: Nonlinear tensor analysis by topographic mappings. *Neural Networks*, *77*, 107–125.

[26] Jordan, M. & Wolpert, D. (1999). Computational motor control. In M. Gazzaniga (Ed.), *The Cognitive Neurosciences*. MIT Press.

[27] Kajić, I., Schillaci, G., Bodiroža, S., & Hafner, V. (2014). Learning hand-eye coordination for a humanoid robot using SOMs. In *ACM/IEEE Int. Conf. HRI*, (pp. 192–193).

[28] Kawato, M., Furukawa, K., & Suzuki, R. (1987). A hierarchical neural-network model for control and learning of voluntary movement. *Biological Cybernetics*, *57*(3), 169–185.

[29] Kohonen, T. (2013). Essentials of the self-organizing map. *Neural Networks*, *37*, 52–65.

[30] Kumar, P. & Behera, L. (2010). Visual servoing of redundant manipulator with Jacobian matrix estimation using self-organizing map. *Robot. and Auton. Syst.*, *58*(8), 978–990.

[31] Kurtzer, I. (2015). Long-latency reflexes account for limb biomechanics through several supraspinal pathways. *Frontiers in Integrative Neuroscience*, *8*, 99.

[32] Lallee, S. & Dominey, P. (2013). Multi-modal convergence maps: From body schema and self-representation to mental imagery. *Adaptive Behavior*, *21*(4), 274–285.

[33] Malsburg, C. (1973). Self-organization of orientation sensitive cells in the striate cortex. *Kybernetik*, *14*(2), 85–100.

[34] Manto, M. (2009). Mechanisms of human cerebellar dysmetria: experimental evidence and current conceptual bases. *J. NeuroEngineering and Rehabilitation*, *6*, 10.

[35] Martinetz, T., Ritter, H., & Schulten, K. (1990). Three-dimensional neural net for learning visuomotor coordination of a robot arm. *IEEE Trans. Neural Netw.*, *1*(1), 131–136.

[36] Ménard, O. & Frezza-Buet, H. (2005). Model of multi-modal cortical processing: Coherent learning in self-organizing modules. *Neural Networks*, *18*(5-6), 646–655.

[37] Miikkulainen, R., Bednar, J., Choe, Y., & Sirosh, J. (2005). *Computational maps in the visual cortex.* Springer.

[38] Mohebi, E. & Bagirov, A. (2014). A convolutional recursive modified self organizing map for handwritten digits recognition. *Neural Networks*, *60*, 104–118.

[39] Murata, S., Arie, H., Ogata, T., Sugano, S., & Tani, J. (2014). Learning to generate proactive and reactive behavior using a dynamic neural network model with time-varying variance prediction mechanism. *Advanced Robotics*, *28*(17), 1189–1203.

[40] Nori, F., Natale, L., Sandini, G., & Metta, G. (2007). Autonomous learning of 3D reaching in a humanoid robot. In *IEEE/RSJ Int. Conf. IROS*, (pp. 1142–1147).

[41] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., & Ng, A. (2009). ROS: An open-source robot operating system. In *ICRA Workshop Open Source Software*.

[42] Sainburg, R., Ghez, C., & Kalakanis, D. (1999). Intersegmental dynamics are controlled by sequential anticipatory, error correction, and postural mechanisms. *J. Neurophysiology*, *81*(3), 1045–1056.

[43] Saxon, J. & Mukerjee, A. (1990). Learning the motion map of a robot arm with neural networks. In *Int. Jt. Conf. Neural Networks*, volume 2, (pp. 777–782).

[44] Schulz, R. & Reggia, J. (2004). Temporally asymmetric learning supports sequence processing in multi-winner self-organizing maps. *Neural Computation*, *16*(3), 535–561.

[45] Sutton, G., Reggia, J., Armentrout, S., & D'Autrechy, C. (1994). Cortical map reorganization as a competitive process. *Neural Computation*, *6*(1), 1–13.

[46] Swindale, N., Shoham, D., Grinvald, A., Bonhoeffer, T., & Hubener, M. (2000). Visual cortex maps are optimized for uniform coverage. *Nature Neuroscience*, *3*(8), 822–826.

[47] Sylvester, J. & Reggia, J. (2009). Plasticity-induced symmetry relationships between adjacent self-organizing topographic maps. *Neural Computation*, *21*(12), 3429–3443.

[48] Walter, J. & Ritter, H. (1996). Rapid learning with parametrized self-organizing maps. *Neurocomputing*, *12*(2–3), 131–153.

[49] Yu, Z. & Lee, M. (2015). Real-time human action classification using a dynamic neural model. *Neural Networks*, *69*, 29–43.