A Self-Organizing Map Architecture for Arm Reaching Based on Limit Cycle Attractors

Di-Wei Huang
Dept. of Computer Science
dwh@cs.umd.edu

Rodolphe J. Gentili
Dept. of Kinesiology, NACS
rodolphe@umd.edu

James A. Reggia
Dept. of Computer Science, UMIACS
reggia@cs.umd.edu

University of Maryland, College Park, MD 20742, United States

ABSTRACT

Creating and studying neurocognitive architectures is an active and increasing focus of research efforts. Based on our recent research that uses neural activity limit cycles in self-organizing maps (SOMs) to represent external stimuli, this study explores the use of such limit cycle attractors in a neurocognitive architecture for an open-loop arm reaching task. The goal is to learn to produce a static motor command for arm joints that moves the manipulator to a target spatial location, while the internal neural activity remains oscillatory. Unlike with static SOMs, stabilizing output based on changing neural activity becomes an important issue. Our architecture is also characterized by simple and forgiving timing requirements, meaning that the time of gating among neural components can be set relatively arbitrarily due to the repetitiveness of limit cycle activity. The results indicate that our architecture generalizes to unseen data, and that the overall performance is insensitive to exact gate timing.

Categories and Subject Descriptors

I.2.0 [**Artificial Intelligence**]: General—*cognitive simulation*; I.2.6 [**Artificial Intelligence**]: Learning—*connectionism and neural nets*; I.2.9 [**Artificial Intelligence**]: Robotics—*kinematics and dynamics*

Keywords

Self-organizing maps, Limit cycle attractors, Multi-SOM architecture, Neural oscillation, Open-loop motor control, Arm movements

1. INTRODUCTION

Interest in building artificially intelligent agents based on neurocognitive architectures has been accelerating during recent years. Often such systems try to implement cognitive functions by creating neuro-anatomically grounded simulations of multiple cortical regions using brain-inspired neural networks, among which self-organizing maps (SOMs) are known to be a promising model for cortical regions [19, 21, 8].

A SOM is a two-layer neural network inspired by cortical maps found in biological neural systems [11, 22]. It learns to map high-dimensional inputs onto its output nodes that form a low-dimensional (usually 2D) lattice. This mapping is non-linear and topology-preserving, meaning that nearby output nodes in a trained map are typically sensitive to similar input patterns, although sudden jumps may also occur. Due to this property, SOMs can effectively cluster and visualize complex data, and have gained much attention across several disciplines [11]. At the same time, SOMs have successfully captured many biological phenomena observed in cortical regions, including the topographical self-organization of somatosensory, visual, and auditory cortices [16, 20], the alignment of multiple feature maps [4], the formation of mirror-symmetric maps [21], and related cognitive phenomena [2].

To date SOMs have only played a limited role in large-scale neurocognitive architectures. A fundamental barrier to adopting SOMs in neurocognitive architectures is that most past SOMs have used a *static representation* of information, meaning that each input is encoded by a single fixed activity pattern in a SOM. Moreover, each activity pattern typically contains only a single maximally activated node (i.e., a "winning" node), which is similar to a one-hot encoding scheme. Our previous work addresses this issue by proposing SOMs based on a limit cycle representation [8, 9]. That is, instead of a fixed single-winner activity pattern, each input is encoded by a temporal sequence of multi-winner activity patterns that forms a limit cycle attractor. The limit cycles are learned through self-organization rather than manually specified. We have recently shown that such a representation can be used to encode both static vectors [8] and temporal sequences [9]. A major distinction of our approach is that it takes into account the ongoing temporal transformation of a SOM's activity, rather than only a specific activity pattern occurring at a certain time. An immediate benefit of using limit cycles is increased robustness of a SOM's representation, since attractors by definition can recover from reasonable amounts of perturbation. Additionally, limit cycle attractors can persist indefinitely in a SOM after their corresponding inputs are removed, allowing a less restrictive time period for a downstream component to access the SOM's activity. As a comparison, a static representation in a conventional SOM typically exists only as long as a corresponding input is presented, unless the SOM's activity is artificially frozen. Finally, using limit cycles captures the

.

oscillatory and rhythmic nature of cortical activity, which is clearly a prominent feature of biological neural systems [3, 17]. There is also substantial evidence that cognitive functions such as memory are strongly related to ongoing rhythmic oscillations of neural activity [6].

However, given that a SOM can encode inputs using limit cycles, it is still unknown if such attractor activity can be harnessed to drive a neurocognitive architecture containing multiple SOMs. Although our previous work has provided some positive preliminary results, it is limited in that the architecture has not been used to generate outputs and the data size is limited (50 pairs of phoneme sequences and images) [9]. More importantly, the ability to generalize to new and unseen data, a critical indicator of a successful neurocognitive architecture, is not clear. In this study, we aim to address these issues in the context of an arm reaching task.

Arm reaching, an inverse kinematics problem, is a control task where the manipulator (hand, gripper, etc.) of an arm is to be driven to a target location. This problem is known to be ill-posed and non-linear, and is being studied intensively in robotics and neurosciences. The arm in this context is characterized by multiple rigid segments connected by rotatable joints, which, when rotated, will change the location of the free end of the arm (i.e., the manipulator). Thus the goal of the problem is to find a vector in the joint angle space (i.e., a set of rotation angles for all joints) that brings the manipulator of the arm to the target spatial coordinates in Cartesian space. Traditional non-oscillatory SOMs has been widely used in solving this problem [1]. In [18, 13], a SOM is trained using samples in both the joint angle space and the Cartesian space. Later when the SOM is given only Cartesian inputs, the weights for joint angles in each node serve as an associative memory for patter completion that yields joint angle outputs. Similar strategies can be found in [23, 5], but discrete SOM nodes are smoothed out as a continuous manifold by interpolation. In [14, 12], each SOM node stores extra information (e.g., a matrix) for performing locally linear transformation from the Cartesian space to the joint angle space. In architectures that contain multiple SOMs, it is typical that inputs in each modality are processed separately by a SOM first. These unimodal SOMs are then associated directly or through other SOMs. In [10], inputs from the joint angle space and the Cartesian space are processed separately by two unimodal SOMs, and then the association between winning nodes in the two SOMs are learned using Hebb's rule. In [13], a separate multimodal SOM is used for associating unimodal SOMs, although winning node indexes rather than activity patterns are used as inputs to the multimodal SOM. A more realistic multi-SOM architecture can be found in [15], which is characterized by iterative competition among nodes and topographic connections between SOMs.

Our goal is thus to build a SOM-based architecture for arm reaching that, unlike the past work summarized above, operates on limit cycle activity. In this study, we focus on an open-loop version of the problem, or coarse arm reaching, meaning that the manipulator location during a reaching movement is not provided as feedback to the architecture. This task is analogous to reaching for something with one's eyes closed. To our knowledge, this is the first attempt to build a neurocognitive architecture that generates fixed outputs based on oscillatory neural activity. Our focus is on learning associations between SOMs that generalize, meaning: The architecture needs to be able to invoke a proper oscillatory activity sequence, and eventually a proper output, for a new input never seen during training. Learning such associations for limit cycle activity is a much more challenging task than with static single-winner activity, because each limit cycle contains multiple activity patterns and each activity pattern contains multiple winning nodes. Another equally important objective is to make timing control simple and forgiving, because an architecture based on dynamic neural activity can be much less robust if its operation relies on highly specific timing. Specifically, each component's activity needs to be readable by a downstream component at a relatively arbitrary time without overall performance being substantially compromised. Further, the operation timing also needs to be independent of the exact timing of individual limit cycles (i.e., the start time and the length), so that the architecture does not need to explicitly detect onset of limit cycles.

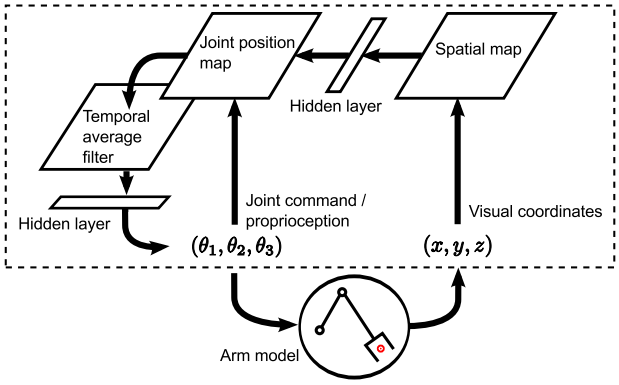

Figure 1: An overview of our architecture. The architecture contains two SOMs, the spatial map and the joint position map, which are connected through a hidden layer. The temporal average filter computes average of the activity in the joint position map, and is connected to the output layer through a hidden layer. An arm model is also created for converting joint positions to spatial locations.

2. METHODS

An overview of the neurocognitive architecture is shown in Fig. 1. This architecture contains two SOMs, the spatial map and the joint position map, each taking input from spatial coordinates of the manipulator and joint proprioception of the arm, respectively. Connections between the two maps are used to associate the two corresponding modalities. The joint position map also drives joint command output, through temporal averaging of its activity. Finally, an arm model, which is not a part of the architecture, is used to convert joint positions to manipulator coordinates. The following sections describe the activation and training of these components.

2.1 Arm Model

The arm being used here is modeled after that of Baxter robot by Rethink Robotics. To manage the complexity of the problem, four of the seven joints are fixed at 0, while two

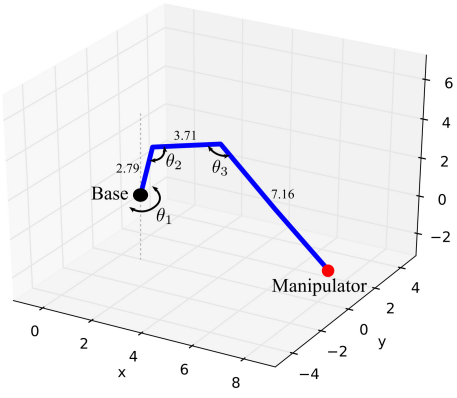

Figure 2: Schematic diagram of the arm model.

shoulder and one elbow joints are freely adjustable, forming a 3-DOF arm operating in a 3D cartesian space. Figure 2 shows the schematic of the arm model. When given a set of joint angles $(\theta_1, \theta_2, \theta_3)$ the arm model determines the spatial location (x, y, z) of the manipulator using the Denavit-Hartenberg method [7, p.435]. The value range of each dimension in both (x, y, z) and $(\theta_1, \theta_2, \theta_3)$ is normalized to fit in the range between 0 and 1, before being fed into the architecture.

2.2 Neural Architecture

The spatial map and the joint position map are SOMs similar to those used in [8], each having two sets of incoming connections, afferent connections and recurrent connections. The afferent connections connect their respective afferent input, i.e., (x, y, z) and $(\theta_1, \theta_2, \theta_3)$, while recurrent connections exist between neighboring map nodes that are within a box distance of 2. The input coming from the recurrent connections has a unit time step delay. The associative connections from the spatial map to the joint position map (top of Fig. 1) give the latter an additional set of incoming connections. The relative strengths for inputs coming from different sets of connections are expressed as adjustable gating parameters. The net inputs for each node i in the two maps at time t, given afferent inputs $\boldsymbol{x} = (x, y, z)$ and $\boldsymbol{\theta} = (\theta_1, \theta_2, \theta_3)$, is defined by:

$$h_i^S(t) = -\alpha_1^S(t) \left\| \boldsymbol{x} - \boldsymbol{w}_i^S \right\|^2 + \alpha_2^S(t) \boldsymbol{a}^S(t-1) \cdot \boldsymbol{u}_i^S, \quad (1)$$

$$h_i^J(t) = -\alpha_1^J(t) \left\| \boldsymbol{\theta} - \boldsymbol{w}_i^J \right\|^2 + \alpha_2^J(t) \boldsymbol{a}^J(t-1) \cdot \boldsymbol{u}_i^J + \alpha_3^J(t) f_{assoc}(\boldsymbol{a}^S(t)), \quad (2)$$

where superscripts S and J correspond to the spatial and the joint position maps, respectively. Each $\boldsymbol{a}(t)$ is the activity pattern of a SOM at time t, and f_{assoc} represents the fully-connected, two-layer feedforward net between the two maps. Parameters α are relative strengths that gate inputs from different sources, and can be different at different t. Trainable parameters $\boldsymbol{w}$'s and $\boldsymbol{u}$'s denote the afferent and recurrent connection weights, respectively. Given h's, the activity output of the maps at time t can be determined using multi-winners-take-all:

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

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

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

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

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 the local neighborhood, upper-bounded by 1. Winner nodes are always fully activated (take on value 1), since $d(i,k)$ in Eq. 3 is 0. Notice that since each activity pattern $\boldsymbol{a}(t)$ depends on the last activity pattern $\boldsymbol{a}(t-1)$, the activity pattern of a map changes with time and forms a dynamical system. Specifically, we have found that limit cycles are a prominent class of attractors that are learned via self-organization [8, 9].

In order to generate steady joint command output, the oscillatory activity in the joint position map needs to be "smoothed out". For this purpose, a temporal average filter is added downstream of the joint position map. The filter contains the same number of nodes as the joint position map, and each node connects one-to-one to the nodes in the map. The activity of each node in the filter is a temporally moving average of the corresponding map node's activity in the last T^F (a parameter) time steps, which can be expressed as:

$$a_i^F(t) = \alpha^A(t) \cdot \frac{1}{T^F} \sum_{t'=0}^{T^F-1} a_i^J(t-t'), \quad (7)$$

where $\alpha^A(t)$ is the output gate. Finally the joint command output

$$\boldsymbol{\theta}^{out}(t) = f_{out}(\boldsymbol{a}^F(t)) \quad (8)$$

is generated, where f_{out}, like f_{assoc} above, represents the fully-connected, two-layer feedforward net between the temporal average filter and the output nodes.

2.3 Training

Training of the architecture is divided into three stages. The two maps are first trained separately. Then the the architecture learns to generate joint command output based on joint proprioception. Finally the inter-modality association is learned between the spatial and the joint position maps.

2.3.1 Stage 1: Individual map training

In this first stage, the two maps are trained separately to obtain limit cycle representations for their respective afferent inputs. The training data for joint proprioception inputs are generated randomly, and are run through the arm model to obtain data for spatial coordinates input. This is analogous to motor babbling in development stages of biological neural systems. Each data sample is presented to a map for 2 time steps from $t = 0$, after which the map continues to

run and adapt for another 4 time steps. The latter is referred to as the continuation time. This is done by specifying the gating parameters:

$$\alpha_1(t) = \begin{cases} 0.64 & \text{if } 0 \le t \le 1 \\ 0 & \text{otherwise} \end{cases}; \ \alpha_2(t) = \begin{cases} 0.36 & \text{if } 0 \le t \le 3 \\ 0 & \text{otherwise} \end{cases};$$
$$\alpha_3^J(t) = 0, \forall t. \quad (9)$$

At each time step, the weights are updated as:

$$\boldsymbol{w}_i(t+1) = \boldsymbol{w}_i(t) + \mu_1 a_i(t)(I - \boldsymbol{w}_i(t)), \quad (10)$$
$$\hat{u_{ik}}(t+1) = u_{ik}(t) + \mu_2 a_k(t-1)\max(0, a_i(t) - a_i(t-1)), \quad (11)$$
$$u_{ik}(t+1) = \hat{u_{ik}}(t+1) / \sum_l \hat{u_{il}}(t+1), \quad (12)$$

where μ_1 and μ_2 are learning rates, and I denotes the afferent input, either $\boldsymbol{x}$ (spatial map) or $\boldsymbol{\theta}$ (joint position map). Eq. 10 specifies a typical unsupervised SOM learning rule, while Eqs. 11 and 12 perform temporally asymmetric Hebbian learning [19]. 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. Readers are referred to [9] for more details. Upon completion of this stage, limit cycle representation is expected to occur in both maps when the maps are allowed to run for a longer period of time, e.g., by setting $\alpha_2(t) = 0.36$ for $0 \le t \le 500$ in Eq. 9.

To test how an input is encoded by a limit cycle after training, spatial coordinates (x, y, z) are presented to the spatial map at $t = 0$ and 1. The activation parameter γ is fixed at 0 after training, meaning each non-winner has an activation value of 0 (inactive) while each winner has 1 (maximally activated). After the input is removed, the activity of the map goes through a brief period of irregular dynamics and eventually settles into a limit cycle attractor, a cyclically repeating sequence of activity patterns. This limit cycle is used as a representation of the corresponding afferent input, which, in this case, is spatial coordinates. As indicated in [8], similar inputs result in similar limit cycles. Fig. 3 shows a sample limit cycle of length 2, where each activation pattern is sparsely-coded.

2.3.2 Stage 2: Joint command output

In this stage, the architecture learns to generate joint command outputs that match given joint proprioceptive inputs. Specifically, when a joint proprioception pattern $\boldsymbol{\theta}$ is presented to the joint position map, the goal is to eventually generate $\boldsymbol{\theta}^{out}$ such that $\boldsymbol{\theta}^{out} \approx \boldsymbol{\theta}$. Again, the training samples are generated randomly. Each joint position input results in a limit cycle in the joint position map, whose activity is then passed to the temporal average filter that generates average activity for the most recent T^F time steps, where T^F is a parameter. The value of T^F can be set rather arbitrarily, as long as it covers the lengths of most limit cycles. At a pre-specified output time t^{out}, the output gate of the temporal average filter is opened ($\alpha^F(t^{out}) = 1$, Eq. 7), passing its activity through the two-layer feedforward net, represented by f_{out}, to generate an output joint command, $\boldsymbol{\theta}^{out} = f_{out}(\boldsymbol{a}^F(t^{out}))$. Again, t^{out} can be set rather arbitrarily, as long as it is late enough such that the activity dynamics of the joint position map has entered a limit cycle. This is because activity of a limit cycle is regular, and thus the results of temporal averages at different times are quite similar. Finally, f_{out} is trained using a resilient error-backpropagation (RPROP) method. Notice that it is possible for different joint positions to be mapped to the same limit cycle due to SOM's discretization effect, and thus a pattern $\boldsymbol{a}^F$ can correspond to multiple $\boldsymbol{\theta}^{out}$ in 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 used to train f_{out}.

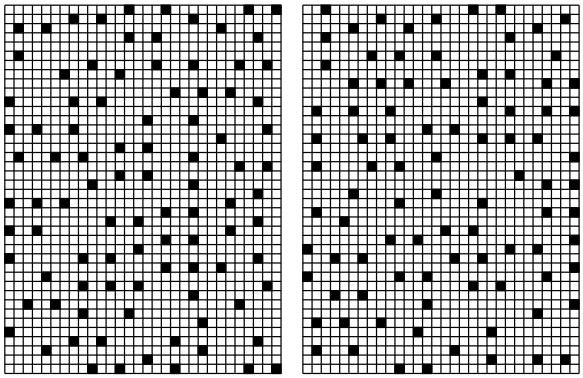

Figure 3: A sample limit cycle of length 2. Each cell represents a SOM node. Black cells represent winner nodes. This particular limit cycle first appears at $t = 7$, meaning the pattern on the left appears at $t = 7, 9, 11, \cdots$, while the pattern on the right appears at $t = 8, 10, 12, \cdots$.

2.3.3 Stage 3: Inter-modality associations

To establish associations between the spatial map and the joint position map, and to eventually be able to transform spatial coordinates to joint commands, limit cycle activity in the spatial map needs to be correlated to that in the joint position map through the feedforward network between the two maps represented by f_{assoc}. More generally, let $A^S(\boldsymbol{x})$ and $A^J(\boldsymbol{\theta})$ each denotes a sequence of activity patterns in the spatial and the joint position maps, when given the manipulator coordinates $\boldsymbol{x}$ and the corresponding joint positions $\boldsymbol{\theta}$ as afferent inputs. The goal of this stage is to train f_{assoc} such that the activity dynamics in the joint position map, when driven solely by $A^S(\boldsymbol{x})$ through f_{assoc} (i.e., without afferent input θ), can become similar to $A^J(\boldsymbol{\theta})$. Note that this is a much harder problem than associating patterns from two single-winner SOMs with static representations, because each representation now contains multiple activity patterns and each activity pattern contains distributed winning nodes.

To generate training data, a number of $\boldsymbol{\theta}$'s is randomly sampled, which leads to corresponding $\boldsymbol{x}$'s. Each $\boldsymbol{\theta}$ and $\boldsymbol{x}$ are then presented to their respective maps for 2 times steps, after which activity of the two maps continues being updated. Starting from a pre-specified time step t^{assoc}, the sequence of activity patterns in the next T^A (a fixed parameter) time steps in both maps are stored as ordered lists A^S and A^J, i.e., $A^{\cdot} = \left[\boldsymbol{a}^{\cdot}(t^{assoc}), \boldsymbol{a}^{\cdot}(t^{assoc}+1), \cdots, \boldsymbol{a}^{\cdot}(t^{assoc}+T^A-1)\right]$. Since the activity of the maps are limit cycles, the exact values of t^{assoc} and T^A are again not critical; they only need to be reasonably late and long enough to cover at least a large portion of a limit cycle.* As with the previous stage, notice

*On the other hand, large T^A slows down training signifi-

that it is possible for multiple different activity sequences in the joint position map to correspond to the same activity sequences in the spatial map, due to discretization of SOMs and the nature of a redundant manipulator. Again, only the most relaxed joint position among them is selected as training data in this case.

To correlate A^S and A^J, our approach is to train the feedforward net between the two maps, f_{assoc}, using error backpropagation (RPROP). Since A^S and A^J contain T^A (a fixed parameter) patterns each, and since each pattern is a result of multi-winner-takes-all processes, conventional error backpropagation does not directly apply and thus needs the following modifications:

1. Error function. The standard error function calculates the squared distance between a target pattern $\boldsymbol{T}$ and an output pattern $\boldsymbol{O}$ as $E(\boldsymbol{T},\boldsymbol{O}) = \sum_i (T_i - O_i)^2$. The target pattern $\boldsymbol{T}$ is a binary vector ($\in \{0,1\}$) since $\gamma = 0$ (see Eq. 3 and Fig. 3). However, this error function does not account for the fact that the downstream component of the net is a SOM, which performs multi-winners-take-all dynamics for node activation. This is too restrictive since the error function drives O_i of a non-winning node i (i.e., $T_i = 0$) toward 0, while in fact, all that is needed is that i is not selected as a winner. To achieve this, O_i only needs to be smaller than the greatest value in its competition neighborhood, i.e., $O_i < \max_{k\in\mathbb{N}_i} O_k$. This is realized by defining the following alternative error function:

$$E(\boldsymbol{T},\boldsymbol{O}) = \sum_i (T'_i - O_i)^2, \qquad T'_i = \begin{cases} 1; & \text{if } T_i = 1, \\ \xi \max_{k\in\mathbb{N}_i} O_k; & \text{if } T_i = 0, \end{cases} \tag{13}$$

   where $\xi = 0.7$ is a discount parameter whose value is determined empirically.

2. Alignment. While f_{assoc} takes one map pattern as input and generate one as output at a time, A^S and A^J each contain T^A patterns. To train f_{assoc}, patterns in A^S and A^J need to be properly aligned to form T^A training samples (i.e., input-target pairs). There are T^A possible cyclic alignment between the two sequences. Namely, let $0 \le \delta < T^A$ be an alignment parameter such that the k-th pattern in A^S is paired with the $(k+\delta \mod T^A)$-th pattern in A^J, $k = 0, 1, \cdots, T^A$. Here we choose the alignment δ^* that minimizes the sum of errors for all pattern pairs. That is,

$$\delta^* = \arg\min_{\delta} \sum_{k=0}^{T^A-1} E(A^J_{(k+\delta \mod T^A)}, f_{assoc}(A^S_k)), \tag{14}$$

   where the error function E is defined in Eq. 13. The value of δ^* is updated for each epoch of training, and the resulting input-target pairs, $\langle A^S_k, A^J_{(k+\delta^* \mod T^A)}\rangle$, are used to adapt weights of f_{assoc}. We hypothesize that, by using such δ^* that minimizes the sum of errors, weight adaptation of f_{assoc} will eventually converge and generalize reasonably well.

cantly. Therefore, we set $T^A = 10$ empirically.

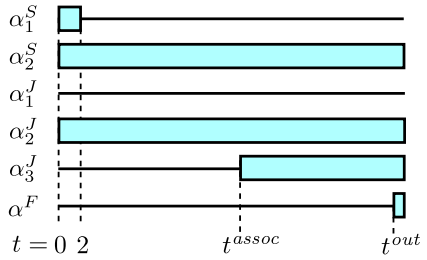

Figure 4: A summary of gate timing. The horizontal axis represents time. A rectangular region indicates that the gate is open (connections enabled), while a horizontal lines indicates that the gate is closed. Values $\alpha^\cdot_1$ and $\alpha^\cdot_2$ denote the gates for the afferent and the recurrent connections of the SOMs, respectively. Value α^J_3 denotes the gate for the incoming associative connections of the joint position map, while α^F denotes the gate for the connections between the temporal average filter and the output layer.

2.4 Testing

After training, new spatial coordinates $\boldsymbol{x}^{targ}$ are provided as input to the architecture, specifying the target location to be reached by the arm. The output joint command of the architecture $\boldsymbol{\theta}^{out}$ is passed through the arm model of Fig. 2 to obtain resulting manipulator coordinates $\boldsymbol{x}^{out}$. By comparing the two spatial locations $\boldsymbol{x}^{targ}$ and $\boldsymbol{x}^{out}$, the spatial error $D = \|\boldsymbol{x}^{targ} - \boldsymbol{x}^{out}\|$ indicates the overall accuracy of the architecture. The target coordinates in testing data are generated by recording the manipulator locations corresponding to a 10-by-10-by-10 grid in the joint position space, i.e., $\theta_1, \theta_2, \theta_3 = \{.05, .15, .25, \cdots, .95\}$. These targets contain some extreme locations that are hard to reach.

The gate timing of the architecture is illustrated in Fig. 4. The afferent input of the spatial map receives fixed target coordinates during the first two time steps, while that of the joint position map remains shut. The latter ensures that the architecture receives only target spatial coordinates as input. The recurrent connections for both maps remain open. From $t = 2$, the activity of the spatial map starts to settle in a limit cycle attractor. The associative connections between the two maps, initially closed, are opened at a fixed time $t = t^{assoc}$, at which point on the changing activity of the spatial map starts to drive the activity of the joint position map, which is initially silent. In addition to the input from the spatial map, the activity of the joint position map is also affected by itself through its own recurrent connections. Finally, at a fixed time $t = t^{out}$, the output of the temporal average filter, which maintains an average of the most recent T^F activity patterns in the joint position map, is open, and then a joint command is generated. Note that, as with training, the values of t^{assoc}, t^{out}, and T^F can be set rather arbitrarily, only that they are sufficiently late such that the activity of the maps has entered a limit cycle attractor. More 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.

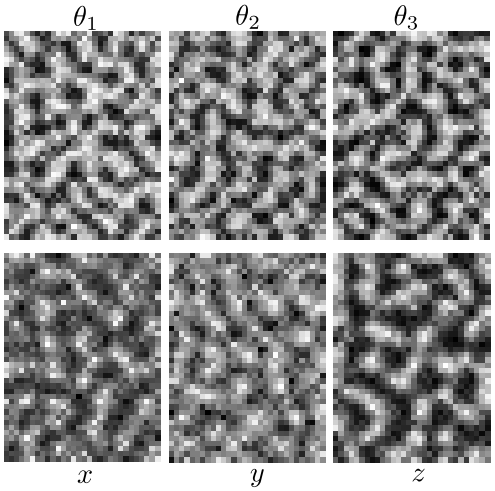

Figure 5: Map formation. Each subgraph plots a weight component of a map, namely θ_1, θ_2, and θ_3 of the joint position map (top row) and x, y, and z of the spatial map (bottom row). Each cell in each subgraph corresponds to a node in a map. Lighter shade indicate a higher value, while a darker shade indicate a lower value.

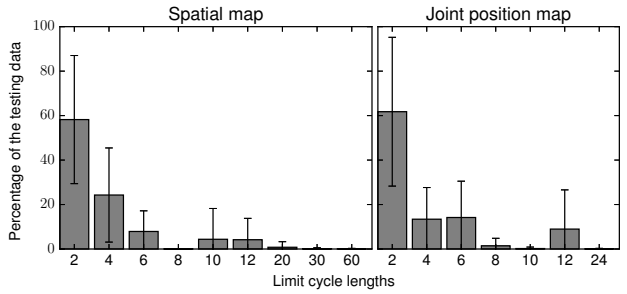

Figure 6: Lengths of limit cycles formed in both maps for the testing data.

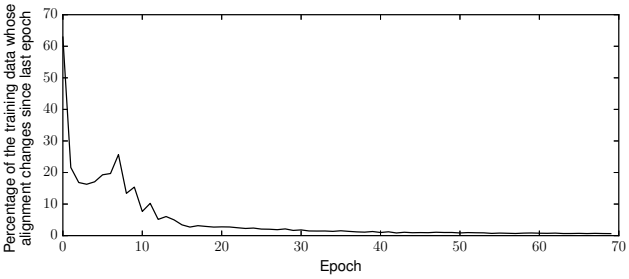

Figure 7: Convergence of limit cycle alignments in the course of training.

3. RESULTS

The results reported below are obtained using an architecture with 40×30 nodes in each map, and 200 nodes in each hidden layer. A total of 10 independent simulations are performed with different initial random weights. The average results are reported below. Unless otherwise noted, the timing parameters are: $t^{assoc} = 50$, $t^{out} = 130$, and $T^F = 30$.

3.1 Map and limit cycle formation

The two maps are separately trained during the first stage of training. Fig. 5 shows the individual afferent weights of both maps after training. The initially random weights become self-organized into quasi-repetitive patterns of high and low value clusters, forming dark and light interleaved stripes. Although this result is certainly less smooth than conventional SOMs, because multi-winner activation is used and the maps are trained for limit cycles, its appearances are qualitatively similar to some cortical maps in biological neural systems.

Fig. 6 summarizes the lengths of the limit cycles formed in each map. On average, about 60% of the testing data results in a limit cycle of length 2 in each map, although there is high variation among different simulations, as indicated by the error bars (standard deviations). Other common lengths include 4, 6, 10, and 12. The lengths of limit cycles tend to be multiples of 2, 3, and 5, where smaller factors appear more frequently. Note that the architecture does not need to detect the lengths of limit cycles. They are shown here for illustration purposes only.

3.2 Convergence of pattern alignment

During the third stage of training (Sect. 2.3.3), when the inter-modality associative connections are being trained, an alignment parameter δ^* is used to align the input and output activity sequences. Since the value of δ^* is updated every training epoch (Eq. 14), if its value keeps changing every epoch, the same input pattern will correspond to a different target pattern when adapting the weights, potentially causing the training process to diverge. To investigate this issue, the changes of alignment in a typical simulation are plotted in Fig. 7. In the first 7 epochs, alignment changes occur with about 20% or higher of the training data, and the percentage may even rise, e.g., from epoch 3 to 7. Later, the alignment changes eventually drop to a low percentage and become stabilized, about 2% at epoch 30 and about 1% at epoch 40.

Such convergence can be considered as indicating that a "consensus" alignment has been reached among the training data, which initially prefer different alignments. By "preferred" we mean the alignment that results in the least error (Eq. 14). Suppose that initially each activity sequence prefers to align differently, there will be an alignment that is preferred by slightly more sequences than other alignments, resulting in slightly more overall influences on weight adaptation. Then the adapted weights in turn encourage more sequences in the training data to prefer this alignment, forming a positive feedback process. This process continues until nearly all sequences prefer the same alignment.

3.3 Spatial error

Fig. 8 shows a comparison of overall spatial errors D, as measured as Euclidean distances (Sect. 2.4), before versus after training. Since the testing data set is never used during training, the results can be used to evaluate the degree of generalization. The value of spatial coordinates are normalized such that each of x, y, and z is within $[0, 1]$. Before

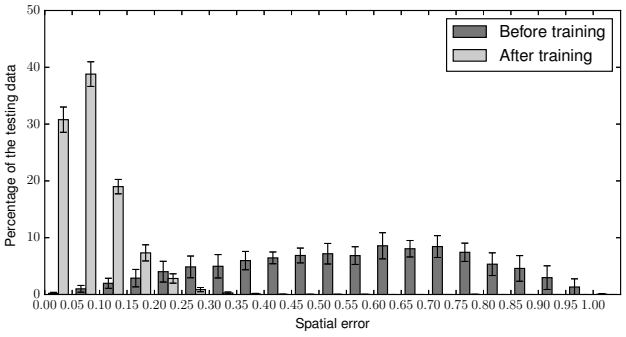

Figure 8: End-to-end spatial error $D = \|\boldsymbol{x}^{targ} - \boldsymbol{x}^{out}\|$ for the testing data before and after training.

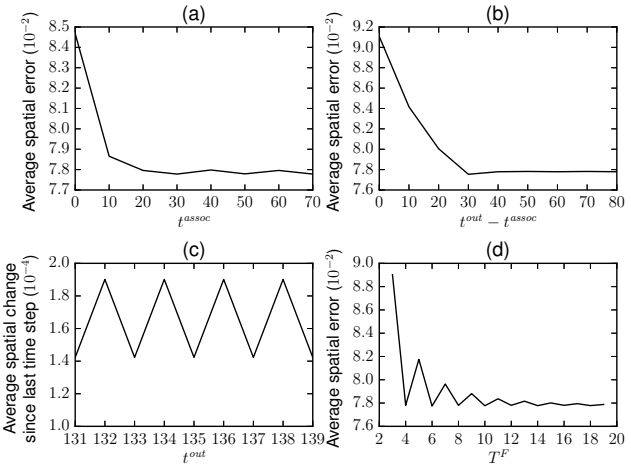

Figure 9: Effects of timing parameters. The baseline values are: $t^{assoc} = 50$, $t^{out} = 130$, and $T^F = 30$. (a–b) and (d) show the effects of t^{assoc}, $t^{out} - t^{assoc}$, and T^F on the spatial error, respectively. (c) shows how much the manipulator moves in each output time step, indicating how stable the output is.

training, the spatial errors of the testing data were widely distributed across the value range, while after training, they are clustered at low values. Errors less than 0.1 account for about 70% of the testing data. Numerically, the average spatial error before training is 0.5578 (SD=0.0522), while after training the value becomes 0.084 (SD=0.0031) quite consistently. The post-training error is about 15% of the pre-training error and 1/12 of the length of each axis. The architecture clearly generalizes to the testing data.

3.4 Sensitivity of timing

Three timing parameters have been introduced in the testing process (Sect. 2.4), namely t^{assoc}, the time from which the associative connections between the spatial map and the joint position map becomes active, t^{out}, the time at which a joint command output is generated, and T^F, the number of time steps on which the temporal average filter operates. In an architecture based on constantly changing activity, the overall performance could be very sensitive to operation timing. This would be undesirable because the system becomes unstable and requires highly accurate control mechanism.

To examine the sensitivity of timing, the three timing parameters were varied in a typical simulation. Fig. 9a shows that the overall performance is insensitive to t^{assoc} after 20 time steps, although there are slight fluctuations. This result indicates that the choice of t^{assoc} is quite unrestrictive. The only requirement is that it is late enough such that the dynamics in the spatial map have entered a limit cycle. Similarly, Fig. 9b indicates that t^{out} can be chosen quite arbitrarily, only that it is later than t^{assoc} by at least $T^F = 30$ time steps, such that the temporal average filter gathers all 30 patterns from the joint position map. Another important indicator is that, since the internal representation of the architecture is non-fixed point, if the architecture is allowed to continuously generate outputs, how stable these outputs are. Fig. 9c shows the changes of manipulator locations at each time step compared with the previous time step. The fluctuations are quite small, less than 2×10^{-4} of the length of each axis. Finally, Fig. 9d shows the effects of T^F on the spatial error. Small T^F results in generally worse performance, although even T^F values tend to perform better than odd ones. This is because the lengths of the limit cycles tend to be multiples of 2. As T^F increases, this effect diminishes and the value of T^F becomes insensitive to even or odd numbers. Therefore, like the other two timing parameters, the choice of T^F can be rather arbitrary, as long as it is large enough.

4. DISCUSSION AND CONCLUSIONS

Although SOMs have been used widely for clustering and visualization purposes as standalone devices, they are certainly not prevalent in designs for large-scale neurocognitive architectures, something that is surprising given the increasing interest in "mapping the brain". In those architectures which do use SOMs, they are often limited by single-winner activation and static representations. In this paper, we have introduced a neurocognitive architecture for open-loop arm reaching based on SOMs using limit cycle activity, which captures cortex-like activity profiles in the sense that they involve sparsely-coded multi-peak activation and rhythmically oscillating dynamics. This type of activity, however, is much harder to harness, especially for tasks that require fixed-point outputs such as arm reaching and fixed position maintenance. We have provided a 3-stage training method where each stage trains a different part of the architecture. First, sensory maps for the spatial and the joint position modalities are formed using unsupervised learning. Then the activity of the joint position map is associated with proper motor outputs. Finally, the inter-modality association is established for relating limit cycles in the two maps.

A potential issue of using limit cycle activity representations is generalization, i.e., whether the architecture can invoke proper limit cycle activity, and thus generate a proper output, when seeing a new input it never saw during learning. To make it worse, there are 5 layers of nodes in the output generation path of the architecture, which results in many degrees of freedom. The simulation results indicate that our architecture does generalize reasonably well. It does so by self-organizing limit cycle representations in individual maps, by self-organizing alignments between limit cycles in the two maps during training, and by temporally

averaging limit cycles when generating constant persisting motor outputs. One possible source of error is the discretization error of a SOM. Although the representational capacity of multi-winner limit cycle activity is much greater than single-winner activity, there are still different inputs being grouped into the same limit cycle. Other possible sources of error come from generalization error of the associative connections between the two maps and between the temporal average filter and the output layer. To further reduce the error, spatial error feedback must be accounted for by the architecture, forming a closed-loop system.

While timing control may not be an important issue for fixed-point neural architectures, it becomes much more relevant in architectures based on dynamic activity patterns, since the activity patterns are constantly changing. In the latter case, if operation of the architecture relies on highly specific timing, not only does it require a highly accurate control mechanism that usually incurs a large overhead, but the system also becomes less robust, because missing a specific activity pattern can potentially cause the system to fail. In our architecture, timing control is relatively simple and forgiving, and the simulation results indicate that the overall performance is insensitive to timing. Specifically, time coordination between neural components, such as the times to enable connections, can be determined quite arbitrarily. The only restriction is that they need to be sufficiently late, after the attractor dynamics in a map are stabilized. Also, the timing parameters, both during and after training, do not depend on the lengths of limit cycles. In fact, they remain fixed for limit cycles of different lengths, and therefore it is not necessary for the architecture to detect or adjust timing for individual limit cycles.

Acknowledgement

This work was supported by ONR award N000141310597.

5. REFERENCES

[1] G. A. Barreto, A. F. R. Araújo, and S. C. Kremer. A taxonomy for spatiotemporal connectionist networks revisited: The unsupervised case. *Neural Computation*, 15(6):1255–1320, 2003.

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

[3] G. Buzsaki. *Rhythms of the Brain*. Oxford University Press, 2006.

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

[5] V. de Angulo and C. Torras. Learning inverse kinematics: Reduced sampling through decomposition into virtual robots. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 38(6):1571–1577, 2008.

[6] J. Fell and N. Axmacher. The role of phase synchronization in memory processes. *Nat Rev Neurosci*, 12(2):105–118, 2011.

[7] R. S. Hartenberg and J. Denavit. *Kinematic synthesis of linkages*. McGraw-Hill series in mechanical engineering. McGraw-Hill, New York, 1965.

[8] D.-W. Huang, R. Gentili, and J. Reggia. Limit cycle representation of spatial locations using self-organizing maps. In *IEEE Symposium on Computational Intelligence, Cognitive Algorithms, Mind, and Brain (CCMB'14)*, pages 79–84, 2014.

[9] D.-W. Huang, R. J. Gentili, and J. A. Reggia. Self-organizing maps based on limit cycle attractors. *Neural Networks*, 63:208–222, 2015.

[10] I. Kajic, G. Schillaci, S. Bodiroza, and V. V. Hafner. A biologically inspired model for coding sensorimotor experience leading to the development of pointing behaviour in a humanoid robot. In *Proceedings of the Workshop HRI: a bridge between Robotics and Neuroscience. 9th ACM/IEEE Int. Conf. on Human-Robot Interaction (HRI'14)*, 2014.

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

[12] P. P. Kumar and L. Behera. Visual servoing of redundant manipulator with Jacobian matrix estimation using self-organizing map. *Robotics and Autonomous Systems*, 58(8):978–990, 2010.

[13] S. Lallee and P. F. Dominey. Multi-modal convergence maps: From body schema and self-representation to mental imagery. *Adaptive Behavior - Animals, Animats, Software Agents, Robots, Adaptive Systems*, 21(4):274–285, 2013.

[14] T. Martinetz, H. Ritter, and K. Schulten. Three-dimensional neural net for learning visuomotor coordination of a robot arm. *IEEE Transactions on Neural Networks*, 1(1):131–136, 1990.

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

[16] R. Miikkulainen, J. A. Bednar, Y. Choe, and J. Sirosh. *Computational maps in the visual cortex*. Springer, 2005.

[17] E. Niedermeyer and F. da Silva. *Electroencephalography: Basic Principles, Clinical Applications, and Related Fields*. Lippincott Williams & Wilkins, 2005.

[18] J. Saxon and A. Mukerjee. Learning the motion map of a robot arm with neural networks. In *International Joint Conference on Neural Networks*, volume 2, pages 777–782, 1990.

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

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

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

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

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