Supplemental Materials

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

Di-Wei Huang, Rodolphe J. Gentili, Garrett E. Katz, and James A. Reggia

S1 Example of input pattern encoded as a limit cycle

To see how an input is encoded by a limit cycle after training, we take the open-loop spatial map as an example. Limit cycles on the other maps are qualitatively similar. The spatial map is provided with a spatial location $\boldsymbol{X}^*$ for 2 time steps as afferent input. 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 (α_{aff} disabled for $t \geq 2$), the activity of the map goes through a brief period of irregular aperiodic 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 the spatial location $\boldsymbol{X}^*$. As indicated in Huang, Gentili, and Reggia (2014), similar inputs result in similar limit cycles. Fig. S1 shows a sample learned limit cycle of length 6 in the spatial map, where each activation pattern is sparsely coded.

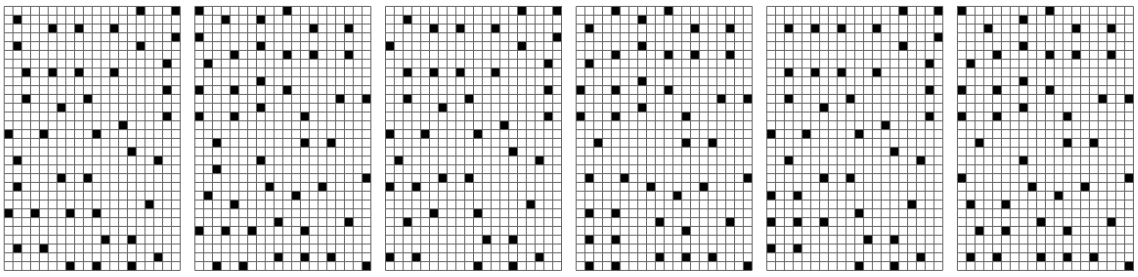

Figure S1: An example of a learned limit cycle of length 6. The patterns from left to right repeatedly occur at consecutive time steps. Each cell in each activity pattern represents a SOM node. Black cells represent "winner nodes". This particular limit cycle first appears at $t = 9$, meaning the left-most pattern appears at $t = 9, 15, 21, \cdots$, the second pattern at $t = 10, 16, 22, \cdots$, etc.

S2 Details of the learning rules in training state 3

To generate training data, a number of joint angles $\boldsymbol{\Theta}$ are randomly sampled, which leads to corresponding spatial locations $\boldsymbol{X}$. For each $\boldsymbol{\Theta}$, a number of small joint angle difference vectors $\boldsymbol{\Delta\Theta}$ are also generated, which result in corresponding spatial differences $\boldsymbol{\Delta X}$. Each $\boldsymbol{X}$ and $\boldsymbol{\Theta}$, as well as normalized $\boldsymbol{\Delta X}$ and $\boldsymbol{\Delta\Theta}$, are then presented to their respective maps via afferent connections for 2 times steps, after which activity of the four maps continues being updated. Starting from a pre-specified time step t^{assoc}, the sequence of activity patterns in the next K (a fixed parameter) time steps in those maps are stored as ordered lists A^S, A^J, A^{SD}, and A^{JD}, i.e., $A^{\cdot} = [\boldsymbol{a}^{\cdot}(t^{assoc}), \boldsymbol{a}^{\cdot}(t^{assoc}+1), \cdots, \boldsymbol{a}^{\cdot}(t^{assoc}+K-1)]$. Since the activity of the maps are limit cycles, the exact values of t^{assoc} and K are not critical; they only need to be sufficiently late and long enough to cover at least a large portion of a limit cycle.* As with training stage 2, it is possible for multiple different activity sequences in the joint angle (difference) map to correspond to the same activity sequences in the spatial (difference) map, due to discretization of SOMs. Whereas for the open-loop subsystem, only the most relaxed joint angles among them is selected as a training target, for the closed-loop subsystem an arbitrary one is chosen.

Given training sequences A^S, A^J, A^{SD}, and A^{JD} described above, each containing K ordered patterns, the goal is to map A^S to A^J, and $(A^{SD}, \overline{A^J})$ to A^{JD}, through feedforward networks f^o_{assoc} and f^c_{assoc}, respectively. Our approach is based on error backpropagation. However, generic error backpropagation methods do not account for patterns resulting from multi-winners-take-all, and they do not naturally deal with sequence-to-sequence mapping with each sequence sampled from a limit cycle. Here we describe how our learning rules address these two issues.

The standard error function calculates the squared distance between a target pattern $\boldsymbol{T}$ and an output pattern $\boldsymbol{O}$ as $E(\boldsymbol{T}, \boldsymbol{O}) = \sum_i (T_i - O_i)^2$. In our case, the target pattern $\boldsymbol{T}$ is a binary vector (elements $\in \{0, 1\}$) since $\gamma = 0$ (see Eq. 3 in the text). However, this error function does not account for the fact that the downstream component of the feedforward net is a SOM, which performs multi-winners-take-all for node activations, and is therefore too restrictive. The reason is that the generic error function unnecessarily 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

*On the other hand, a large value for K slows down training significantly. Therefore, we set $K = 10$ empirically.

needs to be smaller than the greatest value in its local neighborhood, i.e., $O_i < \max_{k\in\mathbb{N}_i} O_k$. This is realized by defining the following alternative error function to guide learning:

$$E(\boldsymbol{T},\boldsymbol{O}) = \sum_i (T'_i - O_i)^2,$$
$$T'_i = \begin{cases} 1; & \text{if } T_i = 1, \\ \xi \max_{k\in\mathbb{N}_i} O_k; & \text{if } T_i = 0, \end{cases} \tag{S1}$$

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

Another adjustment is needed to address the fact that the goal of training is to retrieve a target sequence output, by feeding each consecutive pattern in an input sequence to the feedforward net to be trained. A naive approach would be to pair each pattern in the input sequence with the pattern of the same index in the target sequence for training. However, since both the target and input sequences are sampled from limit cycle dynamics, they both contain periodically repeating or at least partially repeating patterns, and therefore it is possible to train based on a cyclically rotated target (or input) sequence in order to generalize better.

Suppose that the input sequence A^I (i.e., A^S or $(A^{SD}, \overline{A^J})$) and the target sequence A^T (i.e., A^J or A^{JD}) each contains K patterns, $A^{\cdot} = (A^{\cdot}_1, A^{\cdot}_2, \cdots, A^{\cdot}_K)$. Let $0 \le \delta < K$ be an alignment parameter such that the j-th pattern in A^I is paired with the $((j+\delta) \bmod K)$-th pattern in A^T, $j = 1, 2, \cdots, K$. There are K possible values for δ and thus K possible cyclic alignments between A^I and A^T. We found that training with the alignment δ^* minimizing the sum of errors across all pattern pairs can eventually generalize better. That is, we used

$$\delta^* = \arg\min_{\delta} \sum_{j=0}^{K-1} E(A^T_{[(j+\delta) \bmod K]}, f^{\cdot}_{assoc}(A^I_j)), \tag{S2}$$

where the error function E is defined in Eq. S1. The value of δ^* is updated for each epoch of training, and the resulting input-target pairs, $\langle A^I_j, A^T_{[j+\delta^* \bmod K]}\rangle, j = 1, 2, \cdots, K$, are used to adapt weights of $f^{\cdot}_{assoc}$ (either f^o_{assoc} or f^c_{assoc}). We showed in a recent study that the value of δ^* converges during training, and that the resulting $f^{\cdot}_{assoc}$ generalizes well (Huang, Gentili, and Reggia, 2015a).

S3 Lengths of limit cycles

Fig. S2 summarizes the lengths of the limit cycles formed in each map, given 1000 random inputs following Stage 1 of training. On average, about 60% of the random inputs results in a limit cycle of length 2 in three of the maps, although there is high variation among different simulations, as indicated by the error bars (standard deviations). Other common lengths include 4, 6, 10, 12, etc. The lengths of limit cycles tend to be multiples of 2, 3, and 5, where smaller factors tend to appear more frequently. Before the dynamics settle in a limit cycle, there are on average 6.94 (SD=2.52) time steps of irregular/aperiodic activity. Note that our architecture does not need to detect the lengths nor the onset time of limit cycles. They are shown here for illustration purposes only. Moreover, 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 (Huang, Gentili, and Reggia, 2015b).

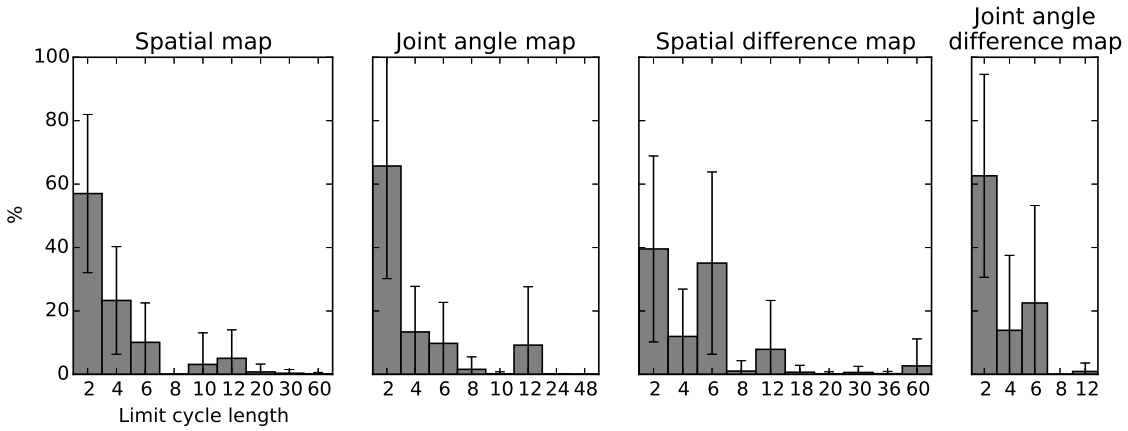

Figure S2: Distribution of the lengths of limit cycles representing 1000 random inputs after training. Each error bar indicates one standard deviation over the 10 independent simulations.

S4 Sensitivity to timing parameters

Internal timing of a neural architecture refers here to the times at which its neural components or connections are enabled or disabled. In other words, internal timing controls and coordinates the sequences of activations/deactivations of the components of the architecture, and thus controls

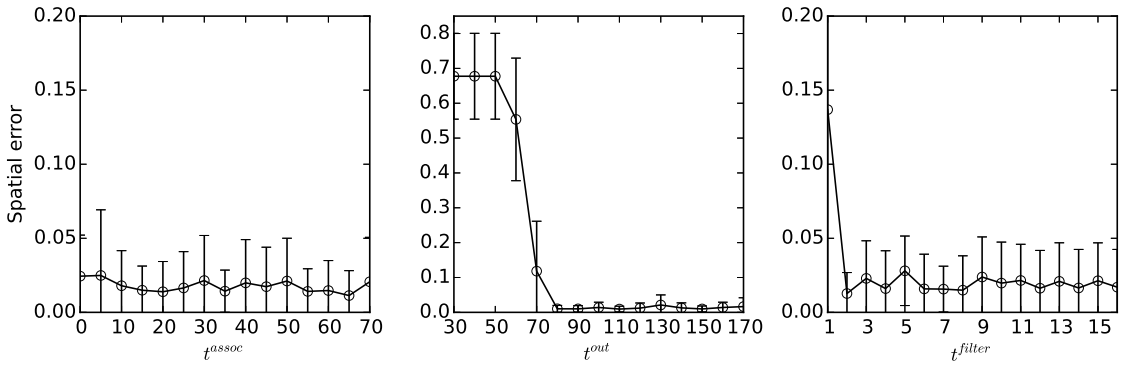

Figure S3: Effects of timing parameter values t^{assoc}, t^{out}, and t^{filter} on spatial error. Each error bar indicates one standard deviation over the 10 independent simulations. Note the different scales on the vertical axes. The t^{filter} range shown in the right-most figure is up to 16 (instead of 30) because further increasing t^{filter} does not lead to substantial changes of spatial error.

the flow of neural activity to achieve certain computations. The complexity of computing feasible timing, as well as the precision requirements for executing the timing, directly contributes to the control overhead incurred by a timing control mechanism (not neurally modeled in this study). It is therefore preferable that internal timing of an architecture be made simple and forgiving, which means that the architecture can tolerate inaccurate timing.

The internal timing of our architecture is fairly simple, containing only three timing parameters: t^{assoc}, t^{out}, and t^{filter}. The value of t^{assoc} defines the time when the associative connections between the spatial map and the joint angle map, as well as those between the spatial difference map and the joint angle difference map, are enabled. The value of t^{out} defines the time when the outputs ($\mathbf{\Theta}^o$ and $\mathbf{\Delta\Theta}^c$) are generated. The value of t^{filter} defines the length of the time window for the temporal average filters. This subsection studies how sensitive the spatial error is to the values of the timing parameters. Forgiving internal timing should accept a relatively wide range of timing parameter values without the spatial error significantly increasing.

In this experiment, the values of t^{assoc}, t^{out}, and t^{filter} are varied while the initial arm position and the spatial target are kept the same as used in Fig. 8 of the main text. The baseline values are: $t^{assoc} = 50$, $t^{out} = 130$, and $t^{filter} = 30$. The effects of varying the three parameters are shown in Fig. S3, where each data point is the average of 10 independent simulations with randomly initialized architectures. The spatial error fluctuates but stays essentially at the same level while

changing t^{assoc} over a rather wide range. The value of t^{out} also has little effect on the spatial error when it is greater than or equal to 80, which is t^{filter} time steps after t^{assoc}. Generating output earlier than this time of course results in poor accuracy either because the neural activity has not been propagated to the joint angle map and the joint angle difference map, or because the temporal average filters have not seen t^{filter} activity patterns. The effects of t^{filter} indicate that as long as the temporal average filter takes an average over more than one activity pattern, the spatial error becomes steady. In summary, our architecture is quite tolerant of changes in control parameter values. The only restrictions about the internal timing of the architecture are: (1) $t^{out} \geq t^{assoc} + t^{filter}$ and (2) $t^{filter} > 1$, which are quite forgiving. More importantly, these timing parameters do not depend on the length or the onset of individual limit cycles, and therefore online limit cycle detection is not necessary.

S5 Per-iteration performance of the closed-loop subsystem

The zig-zag path generated by the closed-loop-only system indicates that the learned mapping from spatial differences to joint angle differences is not perfect. To further understand this, we evaluate the closed-loop outputs on a per-iteration basis. A set of 1000 random spatial difference vectors, paired with random joint angles, were generated to serve as test inputs to the closed-loop subsystem. The resulting joint angle difference outputs are then converted to spatial vectors using the model arm. The angles between the resulting spatial vectors and the input spatial vectors indicate per-iteration directional errors of the closed-loop subsystem. Fig. S4 plots the distribution of the directional error in degrees before and after training the closed-loop subsystem, averaged over 10 independent trials with different random initial weights. Before training, the angular errors are evenly distributed across the 0–180 degree range, with an average of 90.3 degrees (SD=1.2) and a median of 90.2 degrees (SD=2.6). After training, the errors are distributed mostly among small angles, with an average of 30.9 degrees (SD=1.4) and a median of 18.9 degrees (SD=0.8), substantially lower than the pre-training case but clearly not perfect. One possible source of error is that the architecture is operating on a finite number of limit cycles, while space coordinates and joint angles are continuous. However, the trained closed-loop subsystem can still reach a given target eventually with high accuracy, because to do so, the only requirement is that the manipulator

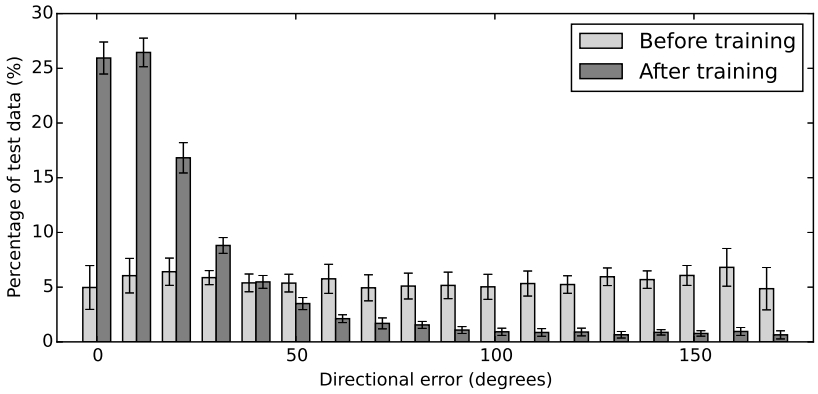

Figure S4: Distribution of per-iteration angular error for the closed-loop subsystem. The angular error is the angle between the spatial direction from the manipulator location towards the target location, and the spatial direction as a result of the joint command generated by the closed-loop subsystem. Each error bar indicates one standard deviation over 10 independent simulations.

moves closer to the target in each iteration, which corresponds to per-iteration directional errors less than 90 degrees. The results show that per-iteration errors less than 90 degrees account for 92.4% (SD=0.9) of the test data, meaning that there is good chance that the manipulator moves closer to the target in each iteration.

S6 Effects of target spatial error ϵ

The value of ϵ determines the spatial accuracy that the arm must meet before a reaching movement is considered completed. Specifically, the spatial distance between the manipulator and the spatial target must be maintained under ϵ for a number of consecutive iterations. Therefore, the value of ϵ affects the overall performance, including the final spatial error and the number of iterations required. A wide range of ϵ values (i.e., 0.001–0.1) was explored to determine relative performance of the full architecture, the standalone open-loop subsystem, and the standalone closed-loop subsystem.

Fig. S5 shows the performance comparisons in terms of spatial error and number of iterations with respect to varying target errors ϵ. The full architecture always outperforms the other two systems in terms of spatial error, indicating a clear benefit of integrating open-loop and closed-loop subsystems. In terms of number of iterations, or speed of a reaching movement, the full architecture

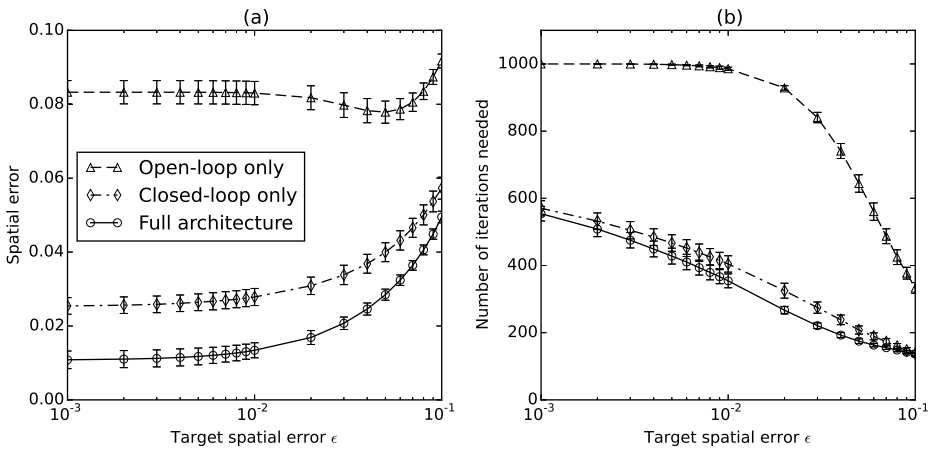

Figure S5: Performance comparisons among an open-loop-only system, a closed-loop-only system, and the full architecture. (a) Average actual spatial error with respect to target spatial error ϵ. (b) Average number of iterations required, upper-bounded by 1000, to reach the target spatial error. The lines are labeled in (a). Error bars indicate standard deviations.

performs significantly better than the open-loop-only system, which mostly takes the maximum number of iterations for $\epsilon < 0.01$, implying that the target errors are never met. Compared to the closed-loop-only system, the full architecture requires slightly fewer iterations, where the differences become significant starting from $\epsilon = 0.01$. However, when ϵ exceeds 0.06, the target error becomes so large that both the full and closed-loop-only systems require very few (i.e., < 100) iterations, and the differences become insignificant. Under moderate target errors (e.g., $0.01 \leq \epsilon \leq 0.06$, roughly 2–12 cm), the full architecture acquires targets faster than the closed-loop-only system with statistical significance.