Input Prediction Using Consensus Driven SOMs

Noémie Gonnier, Yann Boniface, Hervé Frezza-Buet

Input Prediction Using Consensus Driven SOMs

1st Noémie Gonnier
Université de Lorraine, CentraleSupélec
CNRS, LORIA
F-57000 Metz, France
noemie.gonnier@loria.fr

2nd Yann Boniface
Université de Lorraine
CNRS, LORIA
F-54000 Nancy, France
yann.boniface@loria.fr

3rd Hervé Frezza-Buet
Université de Lorraine, CentraleSupélec
CNRS, LORIA
F-57000 Metz, France
herve.frezza-buet@centralesupelec.fr

***Abstract*—The motivation of our work is the instantiation of a computational view of the cerebral cortex. Kohonen's early definition of self-organizing maps was inspired by the cortical substrate on a local scale and is now a widely used learning algorithm. Following the same path, from biology to computation, the cortex can be interpreted as an architecture made of similar self-organizing modules connected together. To our knowledge, there are no such algorithmic derivation of large architectures of self-organizing modules. This paper presents the behavior of several maps connected one to another as a step towards wider networks of self-organizing maps and shows that this architecture learns a model of inputs and generates predictions in a map without using an additional algorithm. This prediction ability is applied to the control of a quadcopter flying in a corridor.**

***Index Terms*—Self-Organizing Maps, Modular Architecture, Prediction**

I. Introduction

Self-organization is a core feature of nervous systems. Many descriptions of the whole cerebral cortex, as in [1], support the idea that the cortex is an architecture made of self-organizing parts. These modules communicate around the sensory information collected by the organism; this communication is performed abstractly, linking sensory and memory information through internal connections. An early definition of self-organizing maps by Kohonen [2] refers to the dynamics of the cortical substrate, considered locally. As opposed to the local self-organization observed in the cortex from which the SOM algorithm has been derived, there is no, as far as we know, algorithmic derivation of large architectures of self-organizing modules. Our work investigates this issue. The creation of a general framework to combine maps in a network, aggregating usual and sequence processing maps and allowing them to handle any input, is, therefore, a relevant challenge in the field of self-organizing maps and unsupervised learning. A suitable choice to achieve such a network seems to use internal connections that depend neither on the dimension of the inputs processed by a map nor on their temporal aspect. Moreover, connections can be bidirectional. The algorithm has to handle this retroactive information transmission. On a computational aspect, the information transmitted between maps should be small to ease computation in larger architectures. Finally, adding modules and inputs should be straightforward. To address all these issues, this paper uses and extends the CxSOM model introduced in [3]. The model is designed as a general framework combining self-organizing maps into architectures

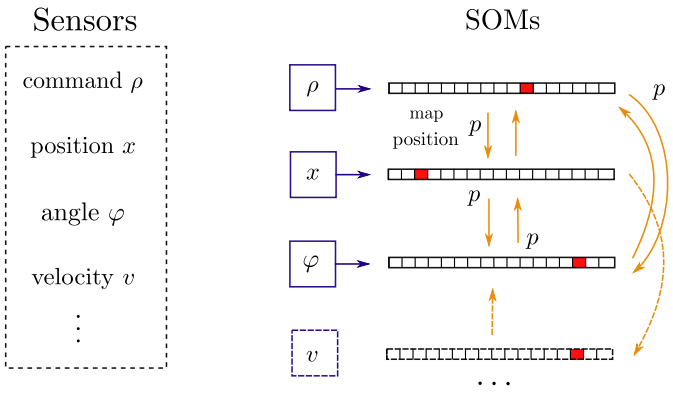

Fig. 1. Example of an architecture of maps on a multimodal input space. Data from sensors can be seen as modalities of different dimensions. Instead of stacking all the modalities into a global vector, each map takes as input one of these modalities. The communication in the architecture only relies on the best matching unit position, i.e a position p in each map. It is then straightforward to add additional maps to the architecture or to modify the connections. Each SOM learns from positions in the other maps, regardless to the dimension of the modality they handle.

where each map learns on different inputs. The communication in the architecture relies on transmitting the best matching unit position, i.e. a 1D position and uses a relaxation process to handle retroactions in the architecture. By these mechanisms, any architecture can be built, like in figure I. The motivation to search for such a modular and generic framework and the choice of the information to be transmitted is motivated by previous works.

Many works have indeed followed the idea that combining simple principles or algorithms leads to complex computational mechanisms. Let us mention deep learning as an aggregation of single-layer perceptrons or cellular automata that are able to perform computation through local interactions. Combining self-organizing maps in a search for more complex learning mechanisms has also been introduced early, like HSOM [4], evaluating the clustering properties of two stacked SOMs. The best matching unit index of the first map is used as an input to the second layer, like CxSOM, but the connections can only be hierarchical. Some works rather focus on the association of maps allowing retroactions, such as A-SOM [5], or [6]. Those models rely on transmitting information between two or three maps so that they learn jointly on different inputs. The information transmitted between the maps

in these works is a whole set of neurons activities. On the other hand, the modular aspect of the brain is closely linked with temporal data processing and memories. Concerning this topic, several models of self-organizing maps for sequence processing have been developed. They are, like the multi-map models, based on the reuse of some internal map information from a learning timestep in the next one [7]. This information is used either to compute the activity or as secondary input in the next timestep. In particular, SOMSD [8]–[10] uses the best matching unit index as this transmitted information. As a model linking temporal and modular aspects, let us cite [11], using a structure of two growing self-organizing networks for sequence processing and decision, inspired by episodic and semantic memories in the brain. All these models, i.e. the multi-map and the sequence processing ones, carry a similar idea of transmitting information between several maps or timesteps. It supports the idea of creating a modular framework.

The goal of the paper is to understand the mechanisms involved in the decentralized self-organization created by CxSOM. One of the computational applications of such a network is multimodal data processing. Most of the time, the data used for learning are high-dimensional, but this dimension can be decomposed in several *modalities*, as reviewed in [12]. For example, a robot has many sensors that acquire image and sound features, positions of robot parts, speed, etc. These are different modalities in the multimodal dataset used to learn and control the robot. An example of processing modalities with an architecture is given in figure I. In this paper, we use CxSOM on a prediction application. An architecture learns on several mutually dependent modalities. Once learning is realized, one of the modalities is not presented to the corresponding map. The other maps drive then the behavior of this map. We show that the output of this map is a good prediction of the missing modality, and therefore, the network of maps has learned multimodal connections. The paper is organized as follows: first, we describe the CxSOM model and how to use it for prediction. Then, we show the prediction ability on simulated inputs in an architecture of three maps. Finally, a four map architecture is used to control a quadcopter in a corridor, using real inputs.

II. Algorithm Description

The algorithm used in this paper is a modified version of the standard Kohonen self-organizing map algorithm, designed to create non-hierarchical architectures only composed of self-organizing maps. The model has been introduced in [3] with a two-map architecture, showing how the maps get organized and learn a relationship between multimodal inputs. The CxSOM model aims at simplifying joint self-organized biologically inspired processes [13], [14]. In this paper, we address modality prediction with architectures of maps with CxSOM. Each map takes its inputs in a modality. Once learning has been realized, tests are run with the same algorithm but with one of the maps not receiving its input. The output of this map is taken as a prediction of the input. We use architectures of three, then four maps, on simulated and real inputs.

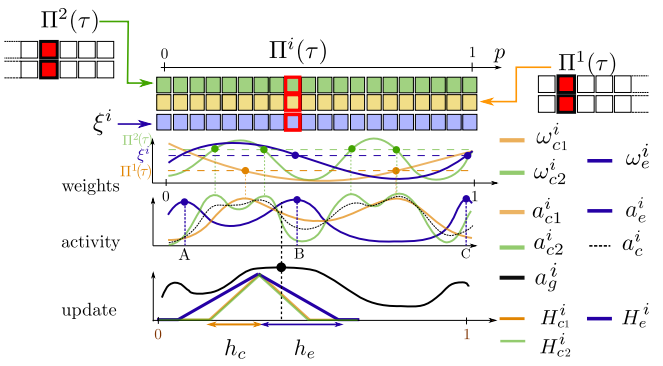

Fig. 2. Weights, activity, and neighborhood functions in a map being connected to two other maps in CxSOM. The external activity has three maxima A, B, and C. The global activity corresponds to the external one modulated by the contextual activity. It allows the map to select the best matching unit at a position close to B, as both the contextual activities are high only in this area.

A. Kohonen Self-Organizing Map

Let us present the standard SOM algorithm with the notations used in our model. A map is an array (1D) or a grid (2D) of computational units. The input feeding the map is taken in a space noted $\mathcal{D}$. To enable better visualization and plotting, we consider in the paper one-dimensional maps of N units where a position $p \in [0, 1]$ indexes each unit. To each of those units is associated a randomly initialized weight vector $\omega_e(p) \in \mathcal{D}$, also referred to as a *prototype*. A learning step, indexed by t is the following:

1) An input $\xi_t \in \mathcal{D}$ is presented to the map.
2) A *best matching unit (BMU)* is found as the position Π_t where a matching function between the input and the weights is maximum. The matching function is a gaussian activation:
$$a(\xi, p) = \exp(\frac{(\xi - w(p))^2}{2\sigma^2} \quad (1)$$
3) Learning is realized by shifting the weight of each unit towards ξ_t relatively to how close it is from the BMU position Π_t:
$$\forall p,\ \omega_e(p, t+1) = \omega_e(p, t) + \alpha H_e(\Pi_t, p)(\xi_t - \omega_e(p, t)) \quad (2)$$

$H(\Pi_t, p)$ is a linearly decreasing function around Π_t in the map position space, reaching 0 at a distance h_e. In this work, the learning rate α and the neighborhood radius h_e are constant over time, unlike most SOM-based works. The algorithm performs vector quantization by creating a mapping of the input space over the N prototypes, with the specificity that the prototypes of two close units in the map are also close in $\mathcal{D}$, creating a continuity in the mapping.

B. Consensus Driven Architecture : CxSOM

We consider a *multimodal* input space $D^1 \times D^2 \cdots \times D^K$, where each input is a tuple $(\xi^1_t, \cdots, \xi^K_t)$. Instead of stacking

all the modalities into a unique vector to use a standard Kohonen map, the data are processed by an architecture made of K maps, each map M^k learning on one modality D^k. A three map architecture is shown in figure 3, and a map of the architecture is described in figure 2. A map i takes an *external* input ξ^i, having its values in a modality D^i of the input space, and *contextual* inputs; those are the *BMUs* Π^j_t of the connected maps in the architecture. More precisely, the contextual inputs and the BMU search rely on a dynamic relaxation process. Indeed, in an architecture with a bidirectional connection, computing the best matching unit in the first map changes the contextual input of the second, and thus its best matching unit. It would then modify contextual input of the first map and its best matching unit, and so on. To handle those bidirectional connections and, more generally, loops in a network, the BMU search is realized by moving the best matching units slowly in each map until all the best matching units positions in the architecture are stable.

Let us detail the algorithm in a three-map example. The inputs are noted X_t, Y_t, Z_t and the maps M^X, M^Y, M^Z. Each map has two contextual inputs; therefore, the units in the map have three layers of weights: external weights, taking their values in the external input space, and two contextual weight layers, being map positions in $[0, 1]$ and take as inputs the best matching units of the other maps. The operations realized in a learning timestep t are described in figure 4. They are the following:

1) A multimodal input is drawn and each modality input ξ^i_t is presented to the corresponding map M^i as an external input.
2) Then, the BMU selection is a competition relaxation, in a nested timeline noted τ. One step of the relaxation is described in figure 3. Let $\mathbf{\Pi}(\tau) = (\Pi^X(\tau), \Pi^Y(\tau), \Pi^Z(\tau))$ refer to as the temporary best matching units.
   a) $\Pi^\star(\tau)$ is initialized as the position where the external activity is maximum.
   b) The *external* activity $a_e(\xi, p)$ and the *contextual* activities $a_{cj}(\Pi^j, p)$ for all map j connected to map i, are computed according to equation 1, with their respective inputs and weights, and merged into a global activity:

$$a_g(p) = \sqrt{a_e(p)(\beta a_e(p) + (1-\beta)a_c(p))},\ \beta = 0.5$$

   where $a_c(p)$ is the mean of each a_{cj}.
   c) In each map i, the temporary best matching unit $\Pi^i(\tau)$ is moved by a fixed step towards the position where a_g is maximum.
   d) The timestep is recomputed until $\mathbf{\Pi}(\tau)$ has reached a stable state. The consensual stabilization of the best matching unit is a relaxation competition.
3) The best matching units in all the maps are taken as the final stable value of the best matching unit once this relaxation process is achieved, noted $\mathbf{\Pi}_t = (\Pi^X(\tau), \Pi^Y(\tau), \Pi^Z(\tau))$; this is where the *consensus* is found. Learning is realized as in a standard SOM on external and contextual weights separately, by moving the weights towards their targets, according to equation 1. The learning kernels widths h_e and h_c are different.

Learning is stopped when the weights have converged over the iterations. The process is summarized in figure 4.

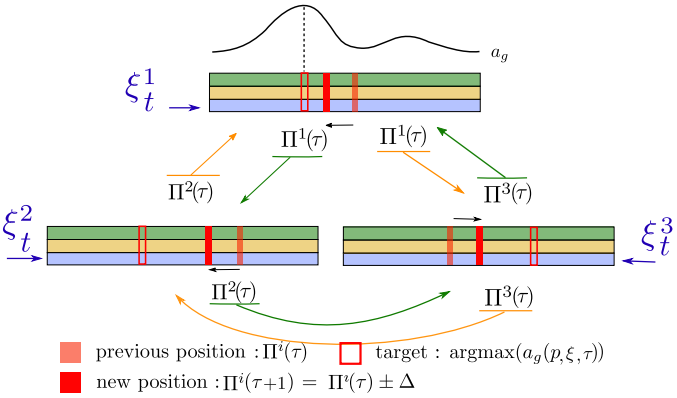

Fig. 3. Relaxation in a 3-map architecture for one timestep t. Once the external inputs ξ^i_t are presented to the maps, the relaxation is a dynamic process indexed by τ. The positions $\Pi(\tau = 0)$ are initialized as the maximum index of a map external activity. Each map has three sets of weights represented in blue (external weights), green, and orange (contextual weights).

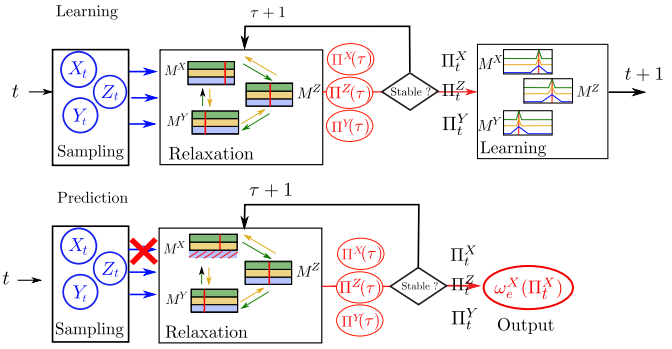

Fig. 4. Algorithm workflow scheme, for learning (top) and prediction (bottom) steps. The external inputs are X, Y and Z.

C. Input Prediction

After learning is realized, the evaluation and prediction rely on tests. Inputs samples are presented to the architecture and the same relaxation algorithm is performed, but the weights are not updated; it is described in figure 4. The activity of a map i can still be computed when the external input is not presented. Its activity is then driven by the contextual inputs only. Thus, relaxation and BMU computation can be performed. The best matching unit weight $\omega^i_e(\Pi^i_t)$ is then a prediction of the missing input; this prediction is what is demonstrated in this paper. The prediction process is performed locally, at a map scale and therefore does not need a global view of the architecture.

III. Prediction of Simulated Inputs

The experiments aim at showing that the architecture is able to predict an input after learning. Indeed, even without an external input, a map still has an activation driven by its

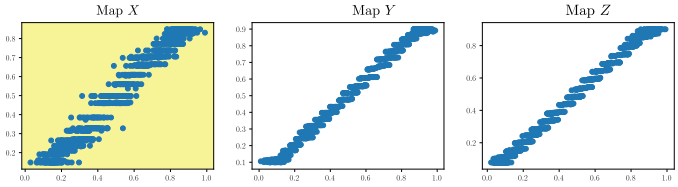

Fig. 5. The BMU weights $\omega_e^X(\Pi^X)$ are plotted according to the actual input X (resp. Y and Z), when map M^X has not received X and is only driven by its contextual inputs, to evaluate the prediction quality of the architecture X. The inputs are taken in a torus, so X,Y and Z are related. The first plot shows that the best matching unit weight is a correct prediction of X. The fact that map M^X does not receive an external input does not affect the behavior of both the other maps, which still achieve vector quantization on their inputs, as shown on the second and third plots.

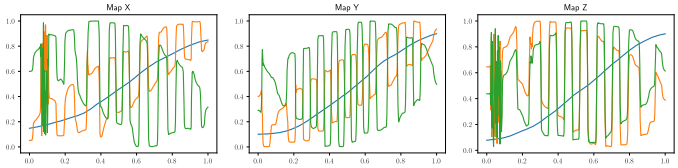

Fig. 6. Disposition of weights in each map. ω_e in blue and the two ω_{cj} in orange and green.

contextual inputs, and thus a best matching unit. The weight $\omega_e^i(\Pi^i)$ of this unit is then used to predict the missing input. Through this process, the architecture is able to activate a map correctly and predict the input.

A. Setup

Before learning on a real dataset, let us show the model's prediction ability on a controlled setup. Inputs $(X, Y, Z) \in [0, 1]$ are coordinates of points located on a circle in 3D-space, to which noise is added in all three dimensions, so that the input is actually taken in a thin torus, see figure 7. The architecture is composed of three maps, which are lines of 500 units, with positions p spanning $[0, 1]$. Each map is connected to both the others and has then two contextual inputs. The neighborhood parameters are $h_e = 0.2$ and $h_c = 0.02$. During the learning phase, each map receives both external and contextual inputs. For plotting, learning is frozen and BMU computation through relaxation is performed on a set of 1000 test inputs (X, Y, Z) taken on the torus. For each input ξ_t some response elements of the maps are gathered: the BMUs $(\Pi_t^X, \Pi_t^Y, \Pi_t^Z)$ and their weights $\omega_e^i(\Pi^i)$. A second test is run without presenting X input to the first map in all the 1000 samples: the global activity of map M^X is only its contextual activity (i.e the average of a_{cy} and a_{cz}). The test elements are plotted to enlight the behavior of the maps: $\omega_e^i(\Pi^i)$ according to the external input ξ^i in figure 5, and the inputs X,Y and Z according to Π^X, Π^Y, Π^Z in figure 7.

B. Discussion

The three plots of figure 5 show that when X input is not presented to the map, $\omega_e^X(\Pi^X)$ can be seen as a valid prediction of the missing observation. The behavior of the other maps M^Y and M^Z is not altered. Let us detail this prediction. The arrangement of the input on a circle is particular: the same value of X corresponds to two distinct points on the

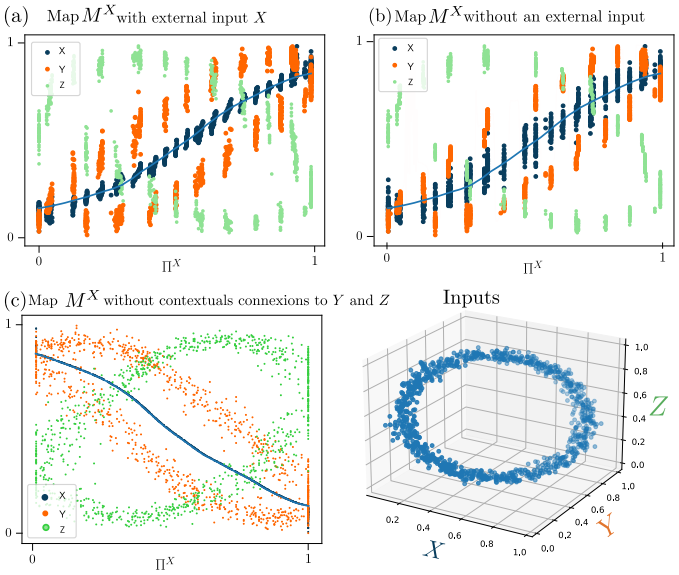

Fig. 7. The 1000 inputs samples (X, Y, Z) are plotted according to the best matching units positions Π^X in map M^X during tests. The first plot shows this repartition when each map has received an input; the second when X has not been presented to map M^X, the third when the maps M^X, M^Y and M^Z are standard Kohonen maps, not connected during learning and tests. The architecture allows map M^X to generate a prediction when the input is not presented, thanks to the other maps.

circle, likewise for Y and Z. In figure 7, the inputs X, Y and Z are plotted according to the BMU of the map M^X, during the test with the input (a), during the test where X is not presented (b), and in the case of a standalone map, without connections to M^Y and M^Z maps (c). These two figures show that self-organization splits the $[0, 1]$ interval of the map positions into small areas, thanks to the wave shape of contextual weights, shown in figure 6. This explanation of this shape is the external neighborhood radius being larger than the contextual ones. This difference ensures weight convergence. The inputs having their BMUs in one of these areas correspond to a contiguous region of the torus, contrary to the case of the standalone map (c) where a same area in the map matches two separate regions of the torus: the values for Y and Z for the same X are randomly mixed up between these regions. The repartition of the inputs in figure 7 (b) shows that when the input is not presented, map M^X still has its best matching unit in the same zone as when X was presented, thanks to the contextual inputs. The best matching unit weight $\omega_e^i(\Pi^i)$ is then close to the input, and is therefore a decent prediction of its value. The same behavior is observed for maps M^Y and M^Z.

In the end, the experiment shows that the architecture delimits zones in a self-organized way, activated either by the external input or only by the contextual inputs. This activation and prediction is done without any external algorithm, and is performed at a map's scale. The map making the prediction only knows the BMU positions of the neighboring maps of the architecture but not the state and weights of the entire architecture. To perform this kind of prediction in a standard self-organizing map, it would be necessary for the predicting

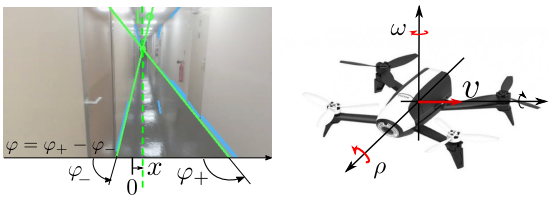

Fig. 8. Left: Vanishing point abscissa x and the two angles used to compute φ. Right: quadcopter with the commands ω, ρ and internal odometry v

algorithm to know the entire map state to select the areas corresponding to the values to predict. This way of mapping data shows rich dynamics of the model and motivates future work.

IV. Application to the Control of a Quadcopter

We propose indoor navigation of quadcopters as a real application for proof of concept. Here, we address the subtask of navigating through a straight narrow corridor thanks to visual control. The quadcopter elevation, as well as the forward speed, are kept constant. The commands are twofold: set a yaw angular speed ω (i.e., turn left or right) and set a roll angle ρ (leaning to the left or right). For example, a constant small positive ρ setting provides a constant *acceleration* of the quadcopter to the left. Internal odometry of the quadcopter provides the current speed v on the left-right axis. Vision processes extract from the front image two scalars: the normalized abscissa of the vanishing point x on the picture (0 means middle), and the difference φ of the angles of the floor lines (φ get far from 0 when the drone gets close to the walls). See figure 8.

The data acquisition is made with a handcrafted controller, where a PID controller is designed to control the lateral speed of the quadcopter (in order to avoid walls) by adjusting the ρ angle. In the same time, the drone keeps x close to zero with a proportional control on ω (i.e., it tries to keep parallel to the corridor axis). The ρ (acceleration) adjustment depends on current v, x, φ. It has to be robust to signal latencies and to the turbulences made by the propellers in such confined flying conditions. Our experiment consists of using four maps, each fed with ρ, v, x, φ rescaled in $[0, 1]$, and learn from observations samples from the handcrafted controller. Then we have removed the ρ input in our architecture, letting the consensus retrieve it and replace the PID controller. The resulting ρ command computed by the map (see figure 9) was applied online to the quadcopter, which flies then safely along the corridor.

V. Conclusion

This paper extends the understanding of a self-organizing map architecture, CxSOM, where only a position is shared between maps. It shows in particular that after the architecture has been trained, a map is able to predict a missing input, driven by its connections to the other maps and a dynamic relaxation process. This prediction is realized in a decentralized way, without using an external algorithm to process the outputs of the maps. The simplicity of the information shared between

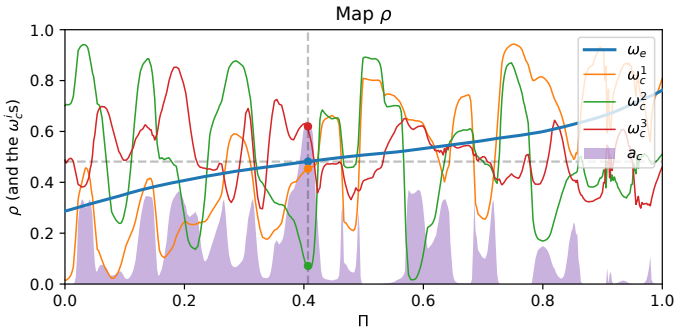

Fig. 9. The activation of the map ρ allowing to retrieve the $\rho = \omega_e(\Pi)$ command, from position Π where a_c (in purple) is maximal.

maps makes CxSOM scalable to larger architectures. Such a simple system (1D, few maps, sharing only positions) exhibits some rich dynamics. Moreover, the model has been easily applied to real inputs with the quadricopter. More fundamentally, we can consider any graph to define an architecture thanks to the genericity of the model formulation. It orients future work on larger and diverse architectures (cycles, high/low arity, etc.). The formulation also includes recurrent maps with time-delayed connections, such as [10]. Using CxSOM in the context of sequence processing and memory is then also a relevant direction for future work.

References

[1] Tom Binzegger, Rodney J. Douglas, and Kevan A. C. Martin. Cortical architecture. In M. De Gregorio, V. Di Maio, M. Frucci, and C. Musio, editors, *Brain, Vision, and Artifical Intelligence*. Springer-Verlag, 2005.

[2] T. Kohonen. The self-organizing map. In *Proc. IEEE*, 1990.

[3] N. Gonnier, Y. Boniface, and H. Frezza-Buet. Consensus driven self-organization: Towards non hierarchical multi-map architectures. In *ICONIP*, 2020.

[4] J. Lampinen and E. Oja. Clustering properties of hierarchical self-organizing maps. *Journal of Mathematical Imaging and Vision*, 1992.

[5] M. Johnsson, C. Balkenius, and G. Hesslow. Associative self-organizing map. In *Proc. IJCCI*, 2009.

[6] M. Lefort, Y. Boniface, and B. Girau. Self-organization of neural maps using a modulated bcm rule within a multimodal architecture. In *Proc. BICS*, 2010.

[7] Barbara Hammer, Alessio Micheli, Alessandro Sperduti, and Marc Strickert. Recursive self-organizing network models. *Neural Networks*, 2004.

[8] M. Hagenbuchner, A. Sperduti, and Ah Chung Tsoi. A self-organizing map for adaptive processing of structured data. *IEEE Transactions on Neural Networks*, 14, 2003.

[9] Barbara Hammer, Alessio Micheli, Nicolas Neubauer, Alessandro Sperduti, and Marc Strickert. Self-Organizing Maps for Time Series. In *WSOM*, 2005.

[10] Jérémy Fix and H. Frezza-Buet. Look and feel what and how recurrent self-organizing maps learn. In *WSOM+*, 2019.

[11] German I. Parisi, Jun Tani, Cornelius Weber, and Stefan Wermter. Lifelong learning of spatiotemporal representations with dual-memory recurrent self-organization. *Frontiers in Neurorobotics*, 2018.

[12] Dana Lahat, Tülay Adalı, and Christian Jutten. Multimodal Data Fusion: An Overview of Methods, Challenges and Prospects. *Proceedings of the IEEE*, 2015.

[13] O. Ménard and H. Frezza-Buet. Model of multi-modal cortical processing: Coherent learning in self-organizing modules. *Neural Networks*, 2005.

[14] B. Khouzam and H.Frezza-Buet. Distributed recurrent self-organization for tracking the state of non-stationary partially observable dynamical systems. *Biologically Inspired Cognitive Architectures*, 2013.