Programming a VG-RAM based Neural Network Computer

Alberto F. De Souza, Avelino Forechi, Filipe Wall Mutz, Mariella Berger, Thiago Oliveira-Santos and Claudine Badue

***Abstract*—We propose a Virtual Generalizing Random Access Memory (VG-RAM) Weightless Neural Network (WNN) Computer (V'Ger Computer for short). VG-RAM WNNs are very effective pattern recognition tools, offering fast training (one shot training) and competitive recognition performance, if compared with other current techniques. The V'Ger Computer architecture was inspired on the organization of the human neocortex and is composed of hierarchically organized and recurrently interconnected layers of VG-RAM WNN neurons. One layer is connected to another in a way similar to cortico-cortical feed-forward and feedback connections between functionally adjacent and hierarchically organized areas. We have "programmed" the V'Ger Computer for counting from 0 to 9 three times. Our preliminary experimental results showed that V'Ger is capable of executing this sequence of actions in spite of strong interferences.**

I. Introduction

In this paper, we propose a biologically inspired computer based on Virtual Generalizing Random Access Memory (VG-RAM) Weightless Neural Networks (WNN). VG-RAM WNNs are very effective pattern recognition tools, offering simple implementation and fast training (one shot training) [1].

Similar to traditional computers, the VG-RAM WNN Computer, or V'Ger Computer for short, adopts a hierarchical programming approach with different levels of command abstractions. However, instead of using a predefined programming language, the V'Ger Computer employs image patterns and a biologically inspired VG-RAM WNN architecture, which is capable of learning functions, commands and actions denoted by these image patterns, that can be proposed by the programmer himself.

The V'Ger Computer architecture is composed of three neural layers organized hierarchically and named Program (the highest level), Function and Command (lower level) neural layers; the number of layers can be increased without loss of generality. The Program layer learns programs that can be invoked by the V'Ger Input and are represented as a sequence of functions. The Function layer learns functions, represented as sequences of commands. Finally, the Command layer learns commands, represented as actions on the V'Ger Output. The Program layer receives a program specification via the V'Ger Input in the form of an image and the Command layer might receive sensor data via the Sensor Input, also in the form of images, from the environment probed by V'Ger, which allows V'Ger to monitor the execution of commands, thereby controlling the program as a whole.

Each neural layer of the V'Ger Computer projects its output in form of an image (i) to the layer immediately below in the hierarchy, (ii) to itself, and (iii) to the layer immediately above in the hierarchy. Thanks to its interlayer interconnection pattern, V'Ger is a recurrent WNN. This interlayer interconnection pattern emulates the cortico-cortical feed-forward and feedback connections between functionally adjacent and hierarchically organized areas of the neocortex [2] and allows the creation of neuron state machines that can execute elaborate programs.

For "programming" the V'Ger Computer, the programmer trains each neural layer to output a specific image when it observes specific input images. This programming approach allows the programmer to establish his own "language" of image patterns representing different levels of command abstractions. In addition, this training can be performed in such a way that procedures are repeated in neural layers of lower levels of abstraction as soon as an image pattern is seen in a neural layer of a higher level of abstraction.

We evaluated the performance of the V'Ger Computer using a simple program for counting from 0 to 9 three times in a hierarchical way. Our experimental results showed that V'Ger is capable of executing the sequence of actions required in spite of severe interferences. Currently, we are "programming" V'Ger for executing elaborate programs for controlling a Pioneer 3-DX robot equipped with a Bumblebee stereo camera, a Sick Light Detection and Ranging (LIDAR), and a laptop with a microphone and speakers.

Although there are other approaches to implement recurrent WNNs that can be trained to execute sequences [3, 4, 5], our biologically inspired VG-RAM WNN architecture for executing sequences of actions is unique in its hierarchical organization and structured programming mechanism, and the results we have obtained so far with it are promising.

This paper is organized as follows. After this introduction, in Section II we briefly discuss related work. In Section III, we present the V'Ger Computer architecture. In Section IV, we

Alberto F. De Souza, Avelino Forechi, Filipe Wall Mutz, Mariella Berger, Thiago Oliveira-Santos, and Claudine Badue are with the Departamento de Informática, Universidade Federal do Espírito Santo, 29075-910 – Vitoria – ES, Brazil (phone: +55-27-4009-2138; fax: +55-27-4009-2850; e-mails: alberto@lcad.inf.ufes.br, claudine@lcad.inf.ufes.br).

This work was supported in part by the Conselho Nacional de Desenvolvimento Científico e Tecnológico – CNPq, Brazil (grants 552630/2011-0, 308096/2010-0, and 314485/2009-0) and Fundação de Amparo à Pesquisa do Espírito Santo – FAPES, Brazil (grant 48511579/2009).

describe VG-RAM WNNs and, in Section V, we explain how we have used them to implement the V'Ger Computer. In Section VI, we show how to program V'Ger. Our conclusions and directions for future work follow in Section VIII. A preliminary version of this work was presented as poster [6].

II. Related Work

The General Neural Unit (GNU [3]) is one of earliest examples of recurrent WNN. Sales et al. [7] employed GNU neurons to build a Neural State Machine (NSM) capable of associating visual (images), linguistic (words represented as images), and proprioceptive (displacements in a computer screen also represented as images) information and act upon these associations according to training. V'Ger resembles the system proposed by Sales et al., but uses a different type of neuron and has a much more general and powerful architecture. Gorse and Taylor [4] have shown that recurrent probabilistic RAM (pRAM) WNNs can store complex temporal sequences and learn regular languages. More recently, Souto et al. [5, 8] have shown that a WNN architecture based on pRAM named General Single-layer Sequential WNN (GSSWNN) is equivalent to a probabilistic automaton. These WNNs are single-layer and do not offer the abstraction of neural structured programing attainable with V'Ger.

III. The V'Ger Computer Architecture

The V'Ger Computer has three neural layers organized hierarchically and named Program (the highest level), Function, and Command (lower level) neural layers, as shown in Fig. 1. The names of the neural layers shown in figure indicate their expected roles. The Program layer, which occupies the highest hierarchical level, learns programs, represented as sequences of functions. The Function layer learns functions, represented as sequences of commands. Finally, the Command layer learns commands, represented as actions on the V'Ger Output. Although throughout this work we present V'Ger with only three layers, the number of layers can be increased without loss of generality.

Each neural layer of the V'Ger Computer projects its output to the layer immediately below in the hierarchy, to itself, and to the layer immediately above in the hierarchy, as shown by the arrows in Fig. 1. So, each neural layer has 3 inputs: (i) the output of the layer immediately above in the hierarchy, (ii) its own output, and (iii) the output of the layer immediately below. These 3 inputs are images.

The neural layers of the V'Ger Computer are two-dimensional arrays of VG-RAM WNN neurons. Each neuron has a number of synapses, which are connected to the inputs of its neural layer, to other neural layers, or to other elements of the architecture of V'Ger, as shown in Fig. 1. This interconnection architecture allows the creation of complex neural state machines.

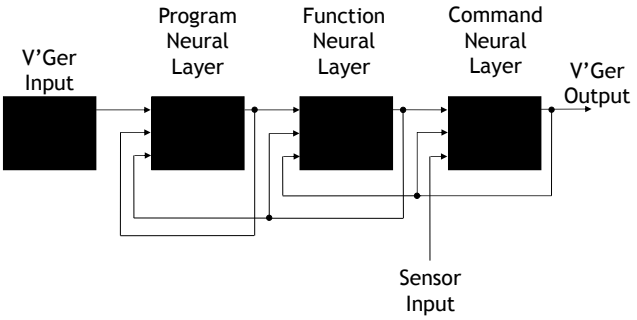

Fig. 1. Hierarchical architecture of the V'Ger Computer.

By examining the interconnection architecture shown in Fig. 1, it is possible to note that one input of the Program layer (V'Ger Input) and one of the Command layer (Sensor Input) are not accounted for. The input of the Program layer receives a program specification via the V'Ger Input in the form of an image and the Command layer might receive sensor data via the Sensor Input, also in the form of images, obtained from, for example, stereo cameras, laser range finders, microphones, etc. The Sensor Input allows V'Ger to monitor the execution of commands, thereby controlling the program as a whole.

Structurally, the neural layers of the V'Ger Computer do not differ – they are all capable of memorizing (learning) relations between input images and output images. That is, given a set of input images and one output image, a V'Ger neural layer can be trained so that, when an input image previously learned (or an approximate version of it) is restated, it provides the corresponding (learned) output. However, from a logical point of view, they differ because, as discussed above, the leftmost neural layers occupy a higher hierarchical level than the rightmost.

IV. VG-RAM WNN

Virtual Generalizing Random Access Memory (VG-RAM) Weightless Neural Networks (WNN) are RAM-based neural networks that, in contrast to weighted neuron networks, do not store knowledge in their synapses but in Random Access Memories (RAM) inside the network's neurons. During the training phase, each neuron of these networks learns input-output pairs, composed of input patterns and corresponding output patterns. These pairs are stored in the neurons' memories. After training, given an input pattern, each neuron searches its memory by comparing the input presented to the network with all inputs learned. The output of each neuron is taken from the pair whose input is nearest to the input presented – the distance function employed is the Hamming distance. If there is more than one pair at the same minimum distance from the input presented, the neuron's output is chosen randomly among these pairs. For more details about VG-RAM WNN, please refer to [1, 9].

VG-RAM WNN can be used for image recognition [10, 11]. In image recognition approaches based on machine learning, given a set of pairs of input images and

corresponding output patterns previously learned by the system, the task is to assign an output pattern to an input image previously unseen by the system. To recognize images using VG-RAM WNN, a basic VG-RAM WNN architecture comprised of two layers – an input layer and a neural layer – can be used. The neurons of the neural layer are connected to the input layer through a set of synapses.

During the training phase, an input image and its corresponding output pattern are set in the input layer and the output of the neural layer, respectively. Each neuron collects a binary input vector from the input layer via its set of synapses. In addition, the expected output of each neuron is read from the neural layer. Finally, this input-output pair is stored in the neuron's memory. After training, the neural layer operates in continuous cycles of three phases, each of which are executed in parallel by one of its neurons: (i) extraction of a binary input vector from the input layer via its set of synapses; (ii) search in its memory for the binary input vector collected from an input image during training that is nearest to the binary input vector collected from the current input image; and (iii) output of the pattern associated with the learned and nearest binary input vector. Note that each neuron acts independently, so each one outputs one element of the output pattern according to what it has learnt. Also, input-output patterns learned during training and stored in the neurons' memories can be saved in secondary memory (disk). Therefore, every time the computer restarts, it is not necessary to train the network again. Finally, the search for the nearest binary input vector in each neuron's memory is performed sequentially and the distance is measured using the Hamming distance. The Hamming distance between two binary patterns can be efficiently computed at machine code level in current 64-bit CPUs and GPUs of personal computers using two instructions: one to identify the bits that differ in 64-bit segments of the two binary patterns, i.e. a bit-wise exclusive-or instruction; and another to count these bits, i.e., a population count instruction.

V. The V'Ger Computer Implementation with VG-RAM WNN

Each neural layer of the V'Ger Computer is a two-dimensional array of $m \times n$ VG-RAM WNN neurons, $N = \{n_{1,1}, n_{1,2}, \ldots, n_{m,n}\}$. Each neuron, $n_{i,j}$, has a set of synapses, $S = \{s_1, s_2, \ldots, s_{|S|}\}$, which are connected to 3 two-dimensional inputs of $m \times n$ elements, (i) $I1 = \{i1_{1,1}, i1_{1,2}, \ldots i1_{m,n}\}$, (ii) $I2 = \{i2_{1,1}, i2_{1,2}, \ldots i2_{m,n}\}$, and (iii) $I3 = \{i3_{1,1}, i3_{1,2}, \ldots i3_{m,n}\}$, given by the output of the layer immediately above in the hierarchy (or from the V'Ger Input), its own layer output and the output of the layer immediately below (or from the Sensor Input), respectively. Note that the layers' inputs have the same size of the neurons array, N.

The set of synapses, S, of each neuron is partitioned into 4 subsets of equal size: (i) $S1 = \{s1_1, s1_2, \ldots, s1_{|S|/4}\}$, (ii) $S2 = \{s2_1, s2_2, \ldots, s2_{|S|/4}\}$, (iii) $S3 = \{s3_1, s3_2, \ldots, s3_{|S|/4}\}$ and (iv) $S4 = \{s4_1, s4_2, \ldots, s4_{|S|/4}\}$. For each neuron, $n_{i,j}$, its synapses in $S1$, $S2$ and $S3$ are randomly connected to $I1$, $I2$ and $I3$, respectively; and its synapses in $S4$ are connected to $I2$ according to a two-dimensional Gaussian distribution with variance σ^2 centered at $i2_{i,j}$; i.e., the coordinates k and l of the elements of $I2$ to which $n_{i,j}$ connects via $S4$ follow the probability density functions:

$$\omega_{i,\sigma^2}(k) = \frac{1}{\sigma\sqrt{2\Pi}} e^{-\frac{(k-i)^2}{2\sigma^2}} \text{ and}$$

$$\omega_{j,\sigma^2}(l) = \frac{1}{\sigma\sqrt{2\Pi}} e^{-\frac{(l-j)^2}{2\sigma^2}},$$

where σ is a parameter of the architecture. This Gaussian synaptic interconnection pattern mimics that observed in many classes of biological neurons [12] and allows the learning and recognition of local features [10]. Both random and Gaussian synaptic interconnection patterns of each neuron are created when the network is built and does not change afterwards.

VG-RAM WNN synapses can only get a single bit from the input. Thus, in order to allow our VG-RAM WNN synapses to deal with images, in which a pixel may assume a range of different values, we use *minchinton cells* [13]. Each neuron's synapse, s_t, forms a minchinton cell with the next, s_{t+1}($s_{|S|}$ forms a minchinton cell with s_1). The type of the minchinton cell we have used returns a one if the synapse s_t of the cell is connected to an input element whose value is larger than the value of the element to which the synapse s_{t+1} is connected; otherwise, it returns a zero.

VI. Programming the V'Ger Computer for Counting

A. Counting Program

We have "programmed" the V'Ger Computer for counting from 0 to 9 three times in a hierarchical way. We called this program "Count from 0 to 9 three times" and used a top-down approach for writing it. It is a very simple program, but it highlights V'Ger hierarchical organization and mechanism of structured programming.

Fig. 2 shows the step (i) of our program. As shown in this figure, in step (i), we specified in the V'Ger Input the name of the program to be executed and indicated what should be the output of the Program layer (the symbol is used for specifying what pattern one neural layer should learn). In fact, we specified in the V'Ger Input an image that, for convenience, can be interpreted by humans as the message "Count from 0 to 9 three times"; and indicated as the output of the Program layer an image that, also for convenience, can be read by humans as the message "Count until 10 1".

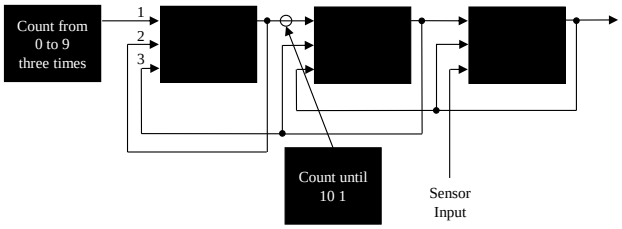

Fig. 2. Step (i) of the program “Count from 0 to 9 three times”.

Once we have specified the items that compound the step (i) of the program, we can then train the Program layer to output the image “Count until 10 1” when it reads from its inputs 1, 2 and 3 the images “Count from 0 to 9 three times”, empty (all pixels equal to zero) and empty, respectively.

Fig. 3 shows the step (ii) of our program. As shown in this figure, in step (ii), we specified again in the V’Ger Input the name of the program and indicated what should be the output of the Program layer. Furthermore, we specified in the content of the Program layer the image “Count until 10 1”.

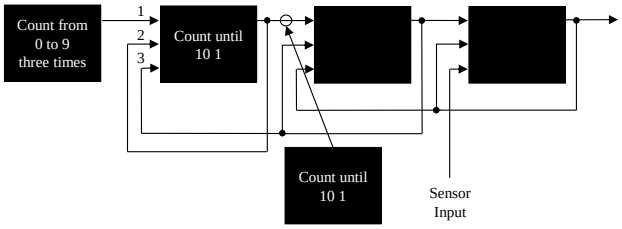

Fig. 3. Step (ii) of the program “Count from 0 to 9 three times”.

Once we have specified the items that compound the step (ii) of the program, we can then train the Program layer to output the image “Count until 10 1” when it reads from its inputs 1, 2 and 3 the images “Count from 0 to 9 three times”, “Count until 10 1” and empty, respectively. That is, up to this point of the program, the Program layer was trained to output the image “Count until 10 1” in two circumstances: first, when its inputs 1, 2, and 3 contain the images “Count from 0 to 9 three times”, empty and empty, respectively; and second, when its inputs 1, 2, and 3 contain the images “Count from 0 to 9 three times”, “Count until 10 1” and empty, respectively. So, given what we have programmed up to this point, if the neural layers are initialized with an empty image and the V’Ger Input with the image “Count from 0 to 9 three times”, after the first cycle, the Program layer will contain the image “Count until 10 1” and, in later cycles, its content will not change (see how VG-RAM layers operates in Section IV).

Fig. 4 shows the step (iii) of our program. As shown in the figure, in step (iii) we indicated what should be the output of the Function layer – the image “After empty” – when it receives in its inputs 1, 2 and 3 the images “Count until 10 1”, empty and empty, respectively.

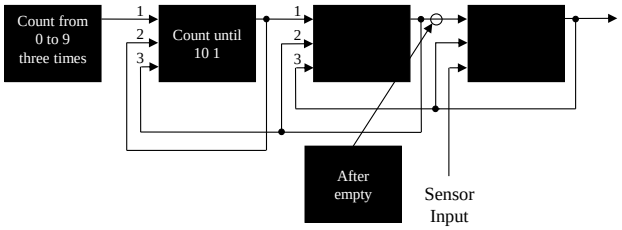

Fig. 4. Step (iii) of the program “Count from 0 to 9 three times”.

Once we have specified the items that compound the step (iii) of the program, we can then train the Function layer to output the image “After empty” when it reads from its inputs 1, 2 and 3 the images “Count until 10 1”, empty and empty, respectively.

In the step (iv) of our program, as shown in Fig. 5, we trained the Function layer to output the image “After empty” when it reads from its inputs 1, 2 and 3 the images “Count until 10 1”, “After empty” and empty, respectively. We also trained the Program layer again to output the image “Count until 10 1” when it reads from its inputs 1, 2, and 3, now, the images “Count from 0 to 9 three times”, “Count until 10 1” and “After empty”, respectively.

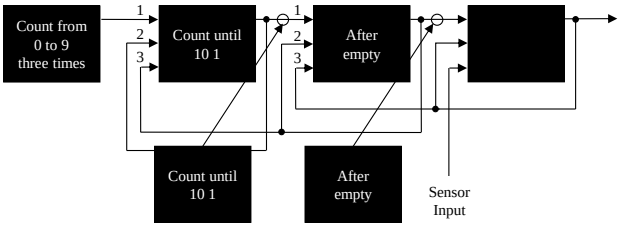

Fig. 5. Step (iv) of the program “Count from 0 to 9 three times”.

In the step (v) of our program, as shown in Fig. 6, we trained the Command layer to output the image “0” when it reads from its inputs 1, 2 and 3 the images “After empty”, empty and empty, respectively. The Sensor Input is not used in this program.

In the step (vi) of our program, as shown in Fig. 7, we trained the Function layer to output “After 0” when it reads from its input 1, 2 and 3 the images ‘Count until 10 1”, “After empty” and “0”, respectively.

In the step (vii) of our program, as shown in Fig. 8, we trained the Program layer again to output the image “Count until 10 1” when it reads from its inputs 1, 2 and 3 the images “Count from 0 to 9 three times”, “Count until 10 1” and “After 0”, respectively.

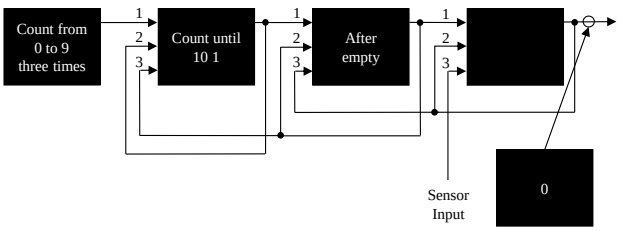

Fig. 6. Step (v) of the program “Count from 0 to 9 three times”.

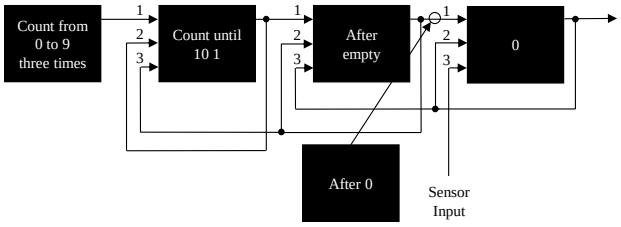

Fig. 7. Step (vi) of the program “Count from 0 to 9 three times”.

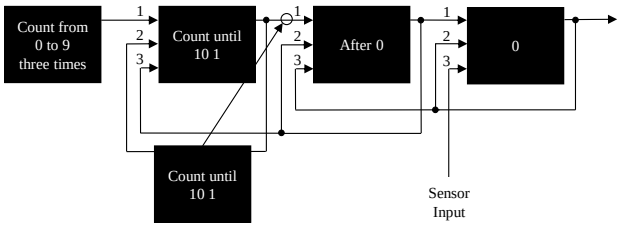

Fig. 8. Step (vii) of the program “Count from 0 to 9 three times”.

In the step (viii) of our program, as shown in Fig. 9, we trained the Command layer to output the message “1” when it reads from its inputs 1, 2 and 3 the images “After 0”, “0” and empty, respectively.

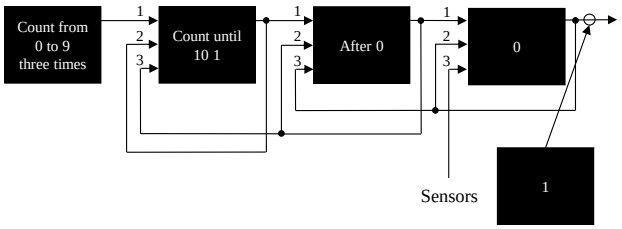

Fig. 9. Step (viii) of the program “Count from 0 to 9 three times”.

In the step (ix) of our program, as shown in Fig. 10, we trained the Function layer to output “After 1” when it reads from its inputs 1, 2 and 3 the images “Count until 10 1”, “After 0” and “1”, respectively.

In the step (x) of our program, as shown in Fig. 11, we trained the Program layer again to output the image “Count until 10 1” when it reads from its inputs 1, 2 and 3, now, the images “Count from 0 to 9 three times”, “Count until 10 1” and “After 1”, respectively.

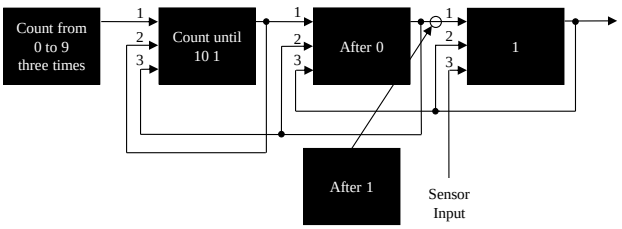

Fig. 10. Step (ix) of the program “Count from 0 to 9 three times”.

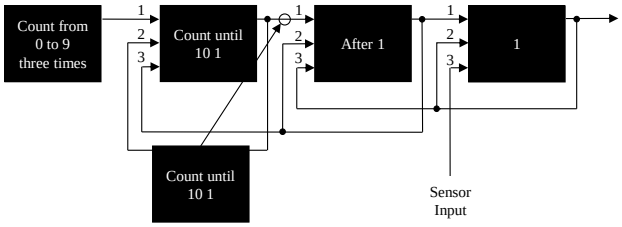

Fig. 11. Step (x) of the program “Count from 0 to 9 three times”.

We repeated steps (viii) to (x) for the numbers 2 to 9, and, in the situation shown in Fig. 12, we programmed the Program layer to advance to “Count until 10 2” and the Function and Command layers to restart a new count from 0 to 9. Training procedures equivalent to those presented earlier can be used to advance to “Count until 10 3”.

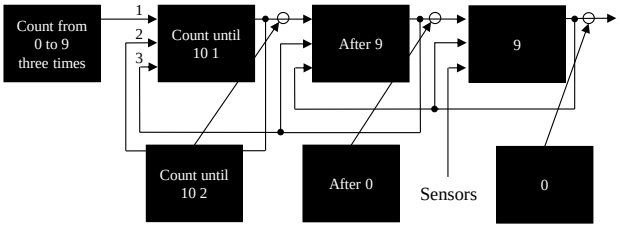

Fig. 12. Step (xi) of the program “Count from 0 to 9 three times”.

To finalize the program, we put the images “Program end”, empty and empty in the content of the Program, Function and Command layers, respectively, and, to restart the program execution, we put the image “Count from 0 to 9 three times” in the V’Ger Input and the empty image in the content of the three neural layers.

The programming procedure described is somewhat tedious and too much detailed. However, programming is this way in most languages. We believe V’Ger programming can be ameliorate through the use of proper tools similar to the compilers used for standard computers (the program described is in the “assembly language” of V’Ger).

B. Mechanism of Selective Attention

Depending on the generalization capacity of the VG-RAM WNN architecture employed in the implementation of the Program layer, when its content advance from “Count until 10 1” to “Count until 10 2”, it might be necessary to retrain the Function layer to correctly generate each of the images (“After 0”, “After 1”, …, “After 9”) required for each possible situation of the Command layer (“0”, “1”, …, “9”). One way to avoid the need for this retraining is to employ in the V’Ger Computer architecture a mechanism equivalent to selective attention [14]. Through selective attention, humans can maintain the behavioral or cognitive processing of interest in the face of distracting or competing stimuli. Through its selective attention mechanism, V’Ger can select what to attend to in an output.

We implemented the mechanism of selective attention using two-mode outputs (a two-color combination, in our

case): in one mode, the output image is used as usual input information to other neural layers of V'Ger; in the other, the output image is used as a mask, which can act as a filter for the input of the neural layers of V'Ger. The second mode of output of a neural layer is learned together with the first (we use two different color channels during training).

Fig. 13 shows the use of the mask mentioned above. In Fig. 13, the mask learned together with each output pattern learned by the Function layer controls a filter that allows the Function layer neurons to see only the part "Count until 10" of the image "Count to 10 1".

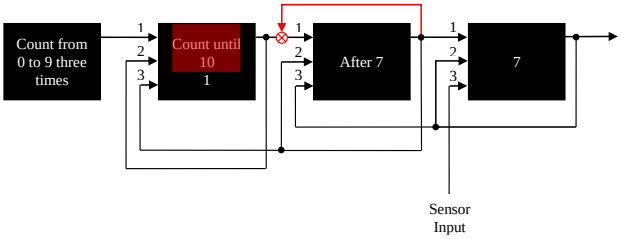

Fig. 13. Mask for the mechanism of selective attention.

Fig. 14 shows a selective attention filter employed in the Command layer input that can also simplify the training of this neural layer. In fact, there are numerous ways of employing the mechanism of selective attention to simplify the V'Ger's programming.

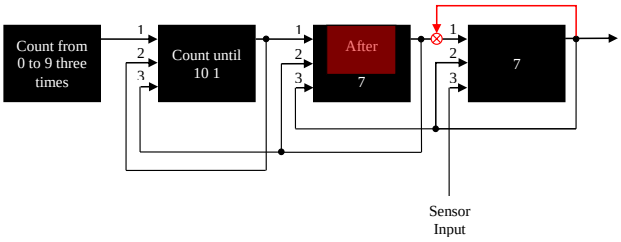

Fig. 14. Other example of the use of the mechanism of selective attention

C. *Demonstration of the Counting Program Execution*

Fig. 15 and Fig. 16 show the training and testing phases of the step (x) of the program "Count from 0 to 9 three times" (Section VI.A) for the number 7, for example. As shown in Fig. 15 and Fig. 16, the output images learned during the training phase are identical to the images outputted during the testing phase, except for some slight noise. A Video that demonstrates the complete execution of the training and testing phases of the counting program can be examined in http://www.inf.ufes.br/~alberto/v-ger.html. The video also shows how V'Ger behaves when submitted to strong interferences during program execution – V'Ger is capable of recover proper program execution even when one of its layers is turned empty or is randomized.

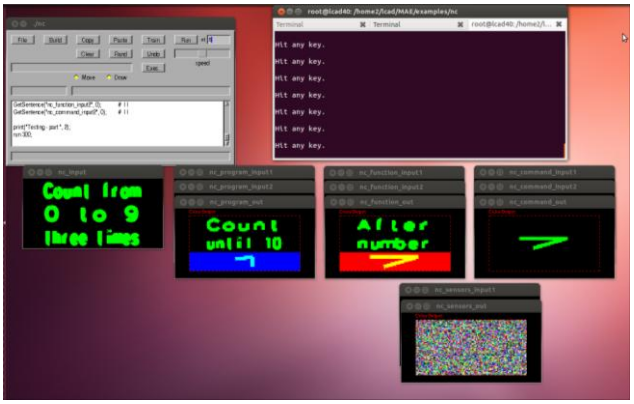

Fig. 15. Training phase of the step (x) of the program "Count from 0 to 9 three times" for the number 7.

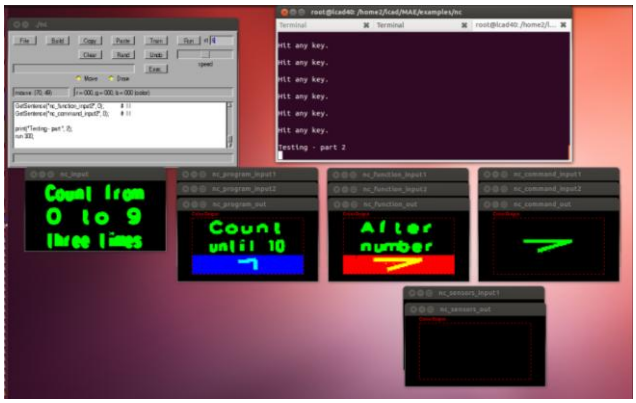

Fig. 16. Testing phase of the step (x) of the program "Count from 0 to 9 three times" for the number 7.

VII. Application of the V'Ger Computer on Autonomous Robotics

Currently, we are "programming" the V'Ger Computer for controlling an autonomous robot, as shown in Fig. 17. To communicate with the robot, V'Ger uses 6 memories: 3 input memories and 3 output memories. The 3 input memories are: (i) Auditory Memory: memory of sounds picked up by a microphone, pre-processed by algorithms of interest; (ii) Visual Memory: memory of stereo images captured by a stereo camera, pre-processed by algorithms of interest; and (iii) Map Memory: memory of maps produced by a Simultaneous Localization and Mapping (SLAM) algorithm [15]. The 3 output memories are: (iv) Image Memory: memory of images that V'Ger would like to find in the visual field of the camera of the robot; (v) Motor Memory: memory of navigation commands that can be sent from V'Ger to the navigation system of the robot; and (vi) Voice Memory: memory of words that V'Ger can speak.

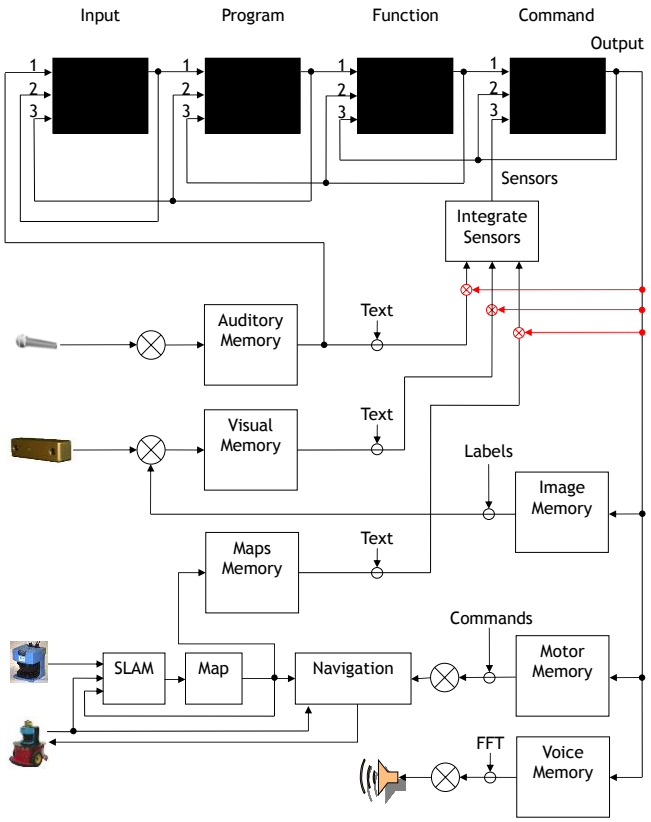

Fig. 17. V'Ger architecture for autonomous robot control.

The 6 memories mentioned above are implemented with layers of VG-RAM WNN neurons as those employed in the implementation of V'Ger (Section V). The inputs of the neural layers that implement the input memories are images representing: (i) the sound (typically spectrograms of words) captured by a microphone; (ii) objects imaged by the stereo camera; and (iii) the map of a specific place. The outputs learned by these neural layers are images with texts describing: (i) words or other sounds captured by the microphone; (ii) objects imaged by the stereo camera; and (iii) positions in the map. These outputs are clustered in a single image by the Integrate Sensors module.

The inputs of the neural layers that implement the output memories are also images with texts generated by the Command layer of V'Ger. These images, generated by the Command Layer, are colored and each output memory is sensible to only one color. In this way, the Command layer can send commands to more than one output memory at the same time. The outputs learned by these neural layers are images with patterns that define: (iv) a specific object of interest imaged by the stereo camera; (v) navigation commands, such as "rotate 10 degrees", "walk forward 30 cm" or "navigate to the position (12, 15)"; and (iv) words to be communicated by V'Ger to humans.

A. *Phonological Loop*

The sound captured by the microphone is filtered (symbol $\otimes$) via fast Fourier Transform (FFT). FFTs of voice signals made at intervals of about 25ms (usually with overlap of about 15ms), or spectrograms, may be grouped over time of a few seconds into two-dimensional images. These images form the input of the Auditory Memory, which represents the Wernicke area – one of the two parts of the cerebral cortex involved in the understanding of written and spoken language [16, 17]. The spectrogram/image (with text describing words) pairs are used to train the Auditory Memory, which constitutes a voice recognition system.

The Auditory Memory is the only one connected to the V'Ger Input and the V'Ger Input is also a neural layer, which has the same interconnection architecture of the other hierarchical levels of V'Ger. The interconnection architecture of the V'Ger Input allows it to maintain a memory of the verbal program requested via microphone and emulates the human phonological loop [14].

It is important to note that the Auditory Memory is also connected to the Sensor Input. This connection is important because it allows the influence of voice commands during program executions.

B. *"What" Stream*

The Visual Memory tries to emulate the "what" stream – one of the two main pathways of the visual information that travels from the occipital lobe to the temporal lobes and is involved with object identification and recognition [18]. It is also a pattern recognition system, which outputs a pattern (label) previously learned for each stereo image of interest presented in its entry, i.e., the Visual Memory allows to label objects in the visual field of the robot. It also allows to identify how far away these objects are and in what extent the objects are at the left or at the right side of the robot.

To identify how far away an object of interest is and to what extent this object is at the right or at the left side of the robot, a SURF transform [19] is applied to the stereo image. The most prominent SURF features of the object of interest are used to guide a filter (symbol $\otimes$) that takes to the input of the Visual Memory only the frame associated with these features of the object of interest. The angle and the distance of the image region with the SURF features of the object of interest are coded as bars in the output image of the Visual Memory, which allows the V'Ger Computer to estimate the position of the object recognized in its visual field.

C. *Choice of Objects of Interest*

V'Ger chooses an object of interest using images outputted by the Command layer with text describing the object of interest, which are recognized by the Image Memory. The Image Memory is trained with image (with text describing objects of interest)/code (of the object of interest) pairs. All neurons of the Image Memory learn the same code for a given text describing an object of interest; this code is used as an index to a list of objects of interest that is known by the filter of the Visual Memory (symbol $\otimes$).

D. *"Where" Stream*

The map used as input to the Map Memory is a two-dimensional grid map centered in the robot. This map represents the "where" stream – the second of the two main pathways of the visual information that travels from the occipital lobe to the parietal lobe and is involved with processing the object's spatial location relevant to the viewer [18]. The Map Memory is a pattern recognition system that outputs a pattern (label) learned previously for each image of interest presented in its input, i.e., the Map Memory allows to label map positions.

E. *Control of the Robot Navigation System*

The Motor Memory is trained with image (with text describing motor commands)/code (of the motor commands) pairs. All neurons of the Motor Memory learn the same code for a given text describing a command; this code is used as an index for a list of commands of interest (symbol $\otimes$).

F. *Generation of Voice Messages*

The Voice Memory is trained with image (with text describing words to be pronounced by V'Ger)/spectrogram pairs and represents the Broca's area – a region in the frontal lobe of the human brain with functions linked to speech production [20]. The spectrograms learned by the Voice Memory are transformed into sound by a filter that implements the inverse FFT (symbol $\otimes$).

VIII. Conclusions and Future Work

We presented a biologically inspired computer based on Virtual Generalizing Random Access Memory (VG-RAM) Weightless Neural Networks (WNN). Different from traditional computers, which use a predefined programming language to create a program, the V'Ger Computer employs a biologically inspired VG-RAM WNN architecture, which is capable of learning functions, commands and actions denoted by image patterns proposed by the programmer himself.

To evaluate the performance of the V'Ger Computer, we "programmed" it for counting from 0 to 9 three times in a hierarchical way. Our experimental results showed that V'Ger is capable of executing this sequence of actions even under severe interferences.

Currently, we are "programming" the V'Ger Computer for controlling an autonomous robot and our preliminary results (not shown here) are promising – most of the modules of Fig. 17 are already implemented. As future work, we plan to finish all modules and start experimenting with the proposed architecture for autonomous robot control.

References

[1] I. Aleksander, "From WISARD to MAGNUS: A family of weightless virtual neural machines", in *RAM-Based Neural Networks*, J. Austin, Ed., World Scientific, pp. 18-30, 1998.

[2] G. M. Shepherd, *The Synaptic Organization of the Brain*, Fourth Edition, Oxford University Press, 2004.

[3] I. Aleksander, "Neural systems engineering: towards a unified design discipline?", *Computing & Control Engineering Journal*, vol. 1, no. 6, pp. 259-265, 1990.

[4] D. Gorse, J. G. Taylor, "Enconding temporal structure in probabilistic RAM networks", *Proc. of IEE International Conference on Neural Networks*, pp. 369-372, 1991.

[5] M. C. P. de Souto, T. B. Ludermir, W. R. de Oliveira, "Equivalence Between RAM-based Neural Networks and Probabilistic Automata", *IEEE Transactions on Neural Networks*, vol. 16, no. 4, pp. 996-999, 2005.

[6] A. F. De Souza, "V Ger: Virtual generalizing random access memory weightless neural network computer", poster presented at the *International Joint Conference on Neural Networks, IJCNN'2013*.

[7] N. J. Sales, R. G. Evans, I. Aleksander, "Successful Naïve Representation Grounding", *Artificial Intelligence Review*, vol. 10, no. 1-2, pp. 83-102, 1996.

[8] M. C. P. de Souto, J. C. M. Oliveira, T. B. Ludermir, "A tool to implement probabilistic automata in RAM-based neural networks", *Proc. of The International Joint Conference on Neural Networks*, pp. 1054-1060, 2011.

[9] T. B. Ludermir, A. C. P. L. F. Carvalho, A. P. Braga, M. D. Souto, "Weightless neural models: a review of current and past works", *Neural Computing Surveys*, vol. 2, pp. 41-61, 1999.

[10] A. F. De Souza, C. Badue, F. T. Pedroni, E. Oliveira, S. S. Dias, H. Oliveira, S. F. Souza, "Face recognition with VG-RAM weightless neural networks", *Proc. of the International Conference on Artificial Neural Networks (ICANN'2008)*, pp. 951–960, 2008.

[11] M. Berger, A. Forechi, A. F. De Souza, J. Oliveira Neto, L. P. Veronese, V. N. Neves, C. Badue, "Traffic sign recognition with VG-RAM weightless neural networks", *Proc. of the IEEE International Conference on Intelligent Systems Design and Applications (ISDA'2012)*, pp. 315-319, 2012.

[12] E. R. Kandel, J. H. Schwartz, T. M. Jessell, *Principles of Neural Science*, Prentice-Hall International Inc, 2000.

[13] R. J. Mitchell, J. M. Bishop, S. K. Box, J. F. Hawker, "Comparison of some methods for processing grey level data in weightless networks", in *RAM-based Neural Networks*, J. Austin, Ed., World Scientific, pp. 61-70, 1998.

[14] R. J. Sternberg, *Cognitive Psychology, 5th Edition*, Cengage Learning, 2008.

[15] S. Thrun, W. Burgard, D. Fox, *Probabilistic Robotics*, The MIT Press, 2005.

[16] I. DeWitt, J. P. Rauschecker, "Phoneme and word recognition in the auditory ventral stream", *Proc. of the National Academy of Sciences*, vol. 109, no. 8, pp. E505-E514, 2012.

[17] I. DeWitt, J. P. Rauschecker, "Wernicke's area revisited: Parallel streams and word processing", *Brain and Language*, vol. 127, no. 2, pp. 181-191, 2013.

[18] M. A. Goodale, A. D. Milner, "Separate visual pathways for perception and action", *Trends in Neuroscience*, vol. 15, no. 1, pp. 20–25, 1992.

[19] H. Bay, A. Ess, T. Tuytelaars, L. V. Gool, "Speeded-up robust features (SURF)", *Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346-359, 2008.

[20] C. Cantalupo, W. D. Hopkins, "Asymmetric broca's area in great apes", *Nature*, vol. 414, pp. 505-505, 2001.