An Adaptive, Self-Organizing Dynamical System for Hierarchical Control of Bio-Inspired Locomotion

Paolo Arena, Senior Member, IEEE, Luigi Fortuna, Fellow, IEEE, Mattia Frasca, Member, IEEE, and Giovanni Sicurella

Abstract—In this paper, dynamical systems made up of locally coupled nonlinear units are used to control the locomotion of bio-inspired robots and, in particular, a simulation of an insect-like hexapod robot. These controllers are inspired by the biological paradigm of central pattern generators and are responsible for generating a locomotion gait. A general structure, which is able to change the locomotion gait according to environmental conditions, is introduced. This structure is based on an adaptive system, implemented by motor maps, and is able to learn the correct locomotion gait on the basis of a reward function. The proposed control system is validated by a large number of simulations carried out in a dynamic environment for simulating legged robots.

Index Terms—Bio-inspired robots, cellular neural networks, dynamical nonlinear systems, locomotion control, motor maps.

I. INTRODUCTION

THE CONTROL OF walking robots is much more challenging than that of wheeled machines, but legged robots offer numerous advantages in terms of maneuverability and the areas they can access. Legged locomotion is the most common way in which animals move. For this reason, biologically inspired solutions have often been adopted to build legged robots [1], [2].

Animals walk in a stereotyped way, adopting rhythmic patterns of movement, called gaits. In a large variety of animals (from invertebrates to vertebrates [3]), the neural control of these stereotyped movements is hierarchically organized. The key point is the presence of a functional unit, called the central pattern generator (CPG), containing all the mechanisms needed to generate the rhythmic pattern of movements. The CPG essentially provides the feedforward signals needed for locomotion even in the absence of sensory feedback and high-level control. The CPG paradigm is one of the most significant applications of bio-inspired solutions to robotics: examples of successful applications are the biomimetic walking robot Scorpion [4], the underwater robots presented in [5], hexapod robots [6]–[8], the quadrupedal walker Tekken [9] and the biped robot [10]. These few examples show how a CPG-based approach allows for the definition of a modular architecture for behavior-based robotics [11]. Seminal works on bio-inspired approaches include those by Brooks [13], and Beer et al. [12]. Other very important approaches, not based on CPG and inspired by studies on the stick insect [14], are discussed in [15].

However, there is still a gap between software architectures based on CPG and theoretical works on CPG models. A widespread approach to modeling CPGs is to use dynamical systems [16]–[19]: CPGs are viewed as networks of coupled nonlinear systems. Some of these works exploit the symmetry of animal gait to build up CPGs and analyze the possible patterns of oscillation by means of considerations on the network symmetry and Hopf bifurcations in symmetrical systems [20]. This strategy considers a network of coupled nonlinear systems with given connections and allows the system parameters to be modulated in order to account for distinct gaits.

There are common assumptions in the dynamical system approach to CPGs: the nonlinear oscillators are often assumed to be identical, the stepping movements of each limb are controlled by a single oscillator, while interlimb coordination is provided by the connections between the oscillators. Moreover, the results are often general and independent of the physiological interpretation of the oscillators: i.e., the oscillator may represent either a single motor-neuron or a population of neurons controlling the movement of a leg or an inter-neuron. In the dynamical approach to CPGs, the term CPG refers to the network controlling the whole motor system, i.e., it comprises both single leg controllers and connections between them.

Cellular nonlinear networks (CNNs) [21] play an important role in this research field. They make it possible to implement nonlinear dynamics like those arising in natural phenomena, through parallel, fully analog circuitry. They therefore provide a framework for the implementation of the CPG for locomotion control, bridging the gap between behavioral CPG-based architectures and dynamical CPG models. Moreover, the intrinsic characteristics of modularity, a key point of the CPG approach, are directly mapped onto a hardware architecture that is modular by definition. The CNN approach is general and can be applied to any locomotion system with particular benefits when the number of actuators grows.

However, in dynamical CPG models locomotion gaits are often regarded as distinct; for instance, a bifurcation in the system model is considered to account for a change in the actual gait. This is in contrast to observed data. Two different gaits are usually identified in insects, tetrapod gait and tripod...
gait. In fact there is a continuous transition between the two gaits and the tripod gait can be considered a fast version of tetrapod gait [14].

This work starts from the implementation of a CPG for a hexapod robot through CNNs presented in [22], extends the model to account for continuous transition between gaits and discusses a novel method to select a given gait. To this end, a high-level control based on a self-organizing map, namely a motor map, is introduced. This map, inspired by the motor cortex, takes the speed of the robot as input and gives the parameters of the CNN generating the gait as output. Clearly, the choice of the input signal is arbitrary and other input signals can be considered.

To investigate the behavior of the system we make use of a dynamic environment simulating the dynamic model of the robot and the control system. This framework is sufficiently accurate to draw conclusions on real implementation of the system.

The novel approach introduced in this paper leads to two important results. First of all, a CPG implementation accounting for a continuous gait and based on CNNs is introduced and a method suitable for real-time control of this network of coupled oscillators is given; in particular this allows control of the speed of the robot with a gait that smoothly changes from alternating tripod to wave gait according to the speed. Then the learning capabilities of the self-organizing map devoted to high-level locomotion control are shown: given a desired locomotion gait, the robot is able to learn walking, passing from a disorganized condition (in which the legs are not synchronized) to organized behavior. It should be pointed out that some a priori knowledge will have to be included; in our case all the mechanisms needed for proper control of each of the legs are known, while interlimb coordination is the result of the learning phase. Finally, it is worth remarking that even though the focus of the paper is on hexapod walking, the approach introduced can be applied to other locomotion systems.

The rest of the paper is organized as follows. Section II gives a brief review of the implementation of CPGs for robot control through CNNs, and then extends this model to account for continuous transitions between gaits, discussing some important properties related to its robustness toward the initial conditions of these dynamical attractors. Section III introduces the high-level control of the CNN CPG, based on self-organizing maps. Section IV gives simulation results validating the approach on a platform for dynamic simulation. Finally, Section V concludes the paper.

II. CNNS IMPLEMENTING CENTRAL PATTERN GENERATORS

A. The Multitemplate Approach to CPG Implementation

In many animals, the motor system is hierarchically organized and the generation of rhythmic movements, needed for instance for walking, takes place at the neural level of the CPG. Signals from the CPG directly control the effector organs, while the CPG receives stimuli from higher-level neurons only to initiate locomotion or to control other high-level features of locomotion (such as the choice of gait).

The CPG paradigm inspired a strategy for the control of robot locomotion based on CNNs [22], [24]. The basic units of these artificial CPGs are nonlinear oscillators coupled together to form a network able to generate a pattern of synchronization that will be used to coordinate the actuators of the robot. Like many other dynamical CPGs, the CNN implementation preserves the generality of the approach: the CNN cells can represent the dynamics of both a single neuron and assemblies of neurons; moreover, the approach is suitable for any locomotion system (in [24] several examples dealing with the control of a worm robot, a lamprey-like robot and a hexapod walker are given). At the same time, the use of CNNs allows a direct VLSI implementation of the control system: a chip for locomotion control implemented by a CNN-based CPG was introduced in [25].

The following equations describe the nonlinear oscillator acting as a “neuron” of the artificial CPG:

\[
\begin{align*}
\dot{x}_1 &= k(-x_1 + (1 + \mu)y_h - s y_2 + i_1 + \sum I_{1,s}) \\
\dot{x}_2 &= k(-x_2 + s y_1 + (1 + \mu)y_2 + i_2 + \sum I_{2,s})
\end{align*}
\]

where \(y_h = 0.5([x_i + 1] - [x_i - 1])\) with \(i = \{1,2\}\).

The concept of “neuron” is common to several works on the CPG based on coupled oscillators. It represents the basic oscillator of the CPG and, as will be shown, the dynamical capabilities of the cell can be efficiently exploited to control the leg kinematics of an insect-like hexapod robot. The concept of “neuron” is motivated by an analogy between this cell and an assembly of biological neurons forming an attractor network. The characteristics of the spatial-temporal dynamics arising in arrays of cells of this type are also qualitatively the same as those arising in pools of biological neurons. The terms \(\sum I_{1,s}\) and \(\sum I_{2,s}\) represent the sum of all the synaptic inputs coming from the other neurons and will be clarified below. For the choice of the parameters given in Table I, system (1) admits a periodic solution with slow-fast dynamics. These regular oscillations provide the rhythmic movements for the robot actuators. This second-order oscillator can be mapped in a two-layer CNN (see Appendix I). For this reason we can consider system (1) as formed by two layers indicated by subscripts.

In this analog approach, the \(n\) actuators of the robot can then be associated with \(n\) CPG neurons. Therefore, to coordinate the movements of the \(n\) actuators it is sufficient to properly synchronize the CPG neurons. This can be done by establishing suitable connections among the nonlinear oscillators. Of course, in terms of dynamical system theory this is a nontrivial task that is solved in [22] by further exploiting the analogy with the biological case. Connections can be excitatory or inhibitory and in terms of CNNs they are implemented by choosing the coefficients of template A (for example, as in (7)). In fact, template A, described in Appendix I, allows us to model CNN neurons connected through their outputs \(y_1\) and \(y_2\). In this case, to allow each cell to have its own characteristic connections, the template is

<table>
<thead>
<tr>
<th>TABLE I</th>
<th>PARAMETERS OF SYSTEM (1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\mu)</td>
<td>0.5</td>
</tr>
<tr>
<td>(s)</td>
<td>1.2</td>
</tr>
<tr>
<td>(i_1)</td>
<td>-0.3</td>
</tr>
<tr>
<td>(i_2)</td>
<td>0.3</td>
</tr>
<tr>
<td>(k)</td>
<td>(\frac{1}{2})</td>
</tr>
</tbody>
</table>
space-variant, i.e., connections depend on the given cell. Moreover, since connections determine the synchronization pattern, and thus the locomotion gait, a set of templates for each gait is given.

The advantages of this analog approach are ease of implementation, flexibility and modularity, allowing an arbitrarily large number of actuators to be controlled. The approach was then applied to control the locomotion of a hexapod robot [7, 22] with 12 degrees of freedom. In the more general case, the control of 12 actuators will require 12 neurons. However, the dynamics of the given neuron (1) and the characteristics of the leg allow us to use one neuron for each leg and thus a CPG with six neurons. Each leg of the hexapod has two degrees of freedom: these are controlled by the two state variables of neuron (1). More precisely, referring to Fig. 1 the following relations were considered between the neuron state variables, neuron outputs and joint variables (the joint $\gamma$ is kept at a constant value):

$$\begin{cases}
\alpha = a_\alpha f_2 + b_\alpha \\
\beta = a_\beta h(x_1) + b_\beta \\
\gamma = b_\gamma
\end{cases}$$

(2)

where $a_\alpha = 0.2, b_\alpha = 0, a_\beta = 0.8, b_\beta = 0.1$ and $b_\gamma = -1.5$ are values expressed in radians and the function $h(x)$ is as follows:

$$h(x) = \begin{cases}
1, & \text{if } x \geq 1 \\
x, & \text{if } 0.5 < x < 1 \\
0.5, & \text{if } x \leq 0.5
\end{cases}.$$  

(3)

Hexapod walkers typically adopt several gaits characterized by different speeds: at low speeds the so-called slow gait or wave gait is adopted, medium gait emerges at intermediate speeds, and alternating tripod or fast gait is adopted at high speeds [26, 27]. The characteristics of these locomotion gaits can be defined through the concepts of cycle time, duty factor, and leg phases [28]. The cycle time is the time required for a leg to complete a locomotion cycle and is defined for periodic gaits (like those examined here). The duty factor $df_i$ is the time fraction of a cycle time in which the leg $i$ is in the power stroke phase. The leg phase $\varphi_i$ is the fraction of a cycle period by which the beginning of the return stroke of leg $i$ lags behind the beginning of the return stroke of the left front leg, chosen as a reference.

Using these parameters, it is possible to characterize fast gait as a periodic gait with the same duty factor for all the legs ($df_i = 0.5$) and the following phase lags:

$$\varphi_{L2} = \frac{1}{2}; \varphi_{L3} = 0; \varphi_{R1} = \frac{1}{2}; \varphi_{R2} = 0; \varphi_{R3} = \frac{1}{2}$$

(4)

where the legs (and consequently the corresponding neurons) are numbered from front to rear and labeled as left (L) or right (R).

For medium gait, the duty factor is $df_i = 5/8$ and the phase lags are as follows:

$$\varphi_{L2} = \frac{3}{4}; \varphi_{L3} = \frac{2}{4}; \varphi_{R1} = \frac{2}{4}; \varphi_{R2} = \frac{1}{4}; \varphi_{R3} = 0.$$  

(5)

For slow gait, the duty factor is $df_i = 9/12$ and the phase lags are as follows:

$$\varphi_{L2} = \frac{4}{6}; \varphi_{L3} = \frac{2}{6}; \varphi_{R1} = \frac{3}{6}; \varphi_{R2} = \frac{1}{6}; \varphi_{R3} = \frac{5}{6}.$$  

(6)

These three gaits are obtained by choosing the connections among the six neurons as in Fig. 2. Since the symbols adopted are not standard, it should be pointed out that each circle represents a CPG motor-neuron as in (1) with an appropriately scaled time unit ($k$ in Table I), controlling a hexapod leg. Moreover, the connections indicate the synapses realizing the CPG scheme: inhibitory synapses (i.e., with a negative sign) terminate with a dot, while excitatory ones terminate with an arrow; those indicated with a continuous line refer to additive terms to the first layer (from the second layer of neuron $kl$ if the dot or the arrow are filled, from the first layer otherwise, i.e., $\sum I_{1,s} = \varepsilon y_{2,kl}$ or $\sum I_{1,s} = \varepsilon y_{1,kl}$ in (1), respectively), whereas those in dashed lines refer to additive terms to the second layer (from the second layer if the dot or the arrow are filled, from the first layer otherwise, i.e., $\sum I_{2,s} = \varepsilon y_{2,kl}$ or $\sum I_{2,s} = \varepsilon y_{1,kl}$ in (1), respectively).

From the schemes shown in Fig. 2, the corresponding sets of templates can be written [22], since connections determine the templates of neuron A. The absence of connections in the MTA diagram indicates zero coefficients in the corresponding template. Excitation and inhibition correspond to positive and negative template coefficients. For instance, as regards the motor-neuron L1, the following template A can be written for fast gait (fg) as follows:

$$A_{L1,fg} = \begin{pmatrix}
A_{L1,fg}^{L1} & A_{L1,fg}^{L2} \\
A_{L1,fg}^{A1} & A_{L1,fg}^{A2}
\end{pmatrix}$$

(7)

with

$$A_{L1,fg}^{L1} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 + \mu & \varepsilon_f \end{pmatrix}; A_{L1,fg}^{L2} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & -s & 0 \end{pmatrix};$$

$$A_{L1,fg}^{A1} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & s & 0 \end{pmatrix}; A_{L1,fg}^{A2} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 + \mu & 0 \end{pmatrix}.$$  

(7)

The central coefficients of the templates account for the dynamics of the cell, while the term $\varepsilon_f$ (which is negative, as shown in Table II) in template $A_{L1,fg}^{L2}$ represents the inhibitory connection between neuron R3 and neuron L1, i.e., the sum
In the case of medium gait (mg), template A is

\[ A^{L1,mg} = \left( \begin{array}{ccc}
A_{11,mg} & A_{12,mg} & A_{22,mg}
\end{array} \right) \]

with

\[ A_{11,mg} = \left( \begin{array}{ccc} 0 & 0 & 0 \\
0 & 1 + \mu & 0 \\
0 & 0 & 0 \end{array} \right) ; \]
\[ A_{12,mg} = \left( \begin{array}{ccc} 0 & 0 & 0 \\
0 & 0 & \varepsilon_m \end{array} \right) ; \]
\[ A_{22,mg} = \left( \begin{array}{ccc} 0 & 0 & 0 \\
0 & 0 & 0 \end{array} \right) \]

Finally, in the case of slow gait (sg), the template A is

\[ A^{L1,sg} = \left( \begin{array}{ccc}
A_{11,sg} & A_{12,sg} & A_{22,sg}
\end{array} \right) \]

with

\[ A_{11,sg} = \left( \begin{array}{ccc} 0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \end{array} \right) ; \]
\[ A_{12,sg} = \left( \begin{array}{ccc} 0 & 0 & 0 \\
0 & 0 & 0 \end{array} \right) ; \]
\[ A_{22,sg} = \left( \begin{array}{ccc} 0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \end{array} \right) \]

The templates for the other neurons can be written in the same way and are given in [22]. As can be seen, each set of templates is characterized by the parameter \( \varepsilon \) alone. The other parameters are, in fact, fixed to determine the dynamics of system (1). In general there exists a range for this parameter leading to the generation of the desired pattern. Ideally one would expect that parameter \( \varepsilon \) affects only the final pattern. However, numerical simulations emphasize that also the frequency of oscillation and the transitory phase depend on this parameter. Moreover, this dependence is quite complex and cannot be used to control, for instance, the frequency of oscillations. Thus, in [22], a suitable value is found by carrying out numerical simulations. Suitable values for the parameters for the three basic gaits are given in Table II.

The templates for the other neurons can be written in the same way and are given in [22]. As can be seen, each set of templates is characterized by the parameter \( \varepsilon \) alone. The other parameters are, in fact, fixed to determine the dynamics of system (1). In general there exists a range for this parameter leading to the generation of the desired pattern. Ideally one would expect that parameter \( \varepsilon \) affects only the final pattern. However, numerical simulations emphasize that also the frequency of oscillation and the transitory phase depend on this parameter. Moreover, this dependence is quite complex and cannot be used to control, for instance, the frequency of oscillations. Thus, in [22], a suitable value is found by carrying out numerical simulations. Suitable values for the parameters for the three basic gaits are given in Table II.

It should be pointed out that the main feature of this approach is that the stepping movements of each limb are controlled by a single oscillator, while interlimb coordination is provided by the connections between the oscillators. The interlimb coordination reflects the pattern of synchronization between oscillators: neurons in fact oscillate at the same frequency (despite small changes in the cell parameters) and have constant phase lags.
determining the sequence in which the legs move, i.e., the locomotion gait. Distinct locomotion gaits are implemented by using distinct connections, corresponding to distinct sets of CNN templates. For this reason, this approach is called the multitemplate approach [22].

Control of oscillation frequency is a very important topic in CPG. CPG is an open-loop control strategy; by controlling the oscillation frequency it will be possible to fully exploit one of the main advantages of using nonlinear dynamic systems: the possibility to entrain the frequency of oscillation with the mechanical structure. The important point is that frequency control in our CPG can be addressed in a very simple way by acting on the time constants of each CPG neuron (this topic will be addressed in detail in a forthcoming paper). Other very important issues regarding locomotion control are related to direction control. To this end, a suitable strategy in these CPG systems is to exploit local bifurcation of the cells controlling the middle legs [23]. In fact, if the bias \( \alpha_2 \) of a neuron controlling a middle leg is slightly changed to a new critical value \( \alpha_c \), the dynamical system undergoes a bifurcation, changing its behavior from limit cycle to a fixed point, while the other neurons of the CPG are only weakly influenced by this bifurcation. This inhibits the rhythmic movement of the associated middle leg, which will be blocked in stance, thus allowing the robot to turn in one direction. A very simple way to control the direction of the robot by using direct connections between sensors and middle CPG neurons allows simple reactive obstacle avoidance to be achieved [23]. However, for the sake of brevity, in this paper we will only focus on forward gait and on the aspects of learning connected to forward motion.

In the following sections, we will extend this model to account for continuous transitions between gaits, and then define a continuous generalized gait and discuss a novel strategy to choose the parameters for this gait. The strategy can also be applied to find suitable \( \varepsilon \) values in real time without the need for numerical simulations.

### B. Analysis of Gaits

The CNN-based CPG presented above is a 12th-order nonlinear system. Analysis of the attractors of this system and their basins of attraction is difficult. As the aim of this work is to introduce an adaptive high-level control able to find the parameter of the CPG network while the hexapod is walking in different situations, the initial conditions are not totally random, since the structure starts from stable configurations, but it is quite difficult to define the set of possible initial conditions. We therefore rely on numerical simulations, which show when coexisting attractors are really involved. In other words, it may happen that even though there exist multiple attractors, in practical cases most of them never arise during the simulation. This is the case of the CPG for medium gait, whereas as regards slow gait (9) several other periodic orbits arise. A “swim gait” with two groups of synchronized neurons (R1, R2, R3 and L1, L2, L3) and a “caterpillar gait” (characterized by the following firing sequence: neurons R1 and L1, then neurons R2 and L2, finally neurons R3 and L3; this sequence is inverted with respect to that actually adopted by caterpillars) often coexist with the desired slow gait for a given \( \varepsilon \) value.

Fig. 3 illustrates the behavior of the CPG for slow gait with different initial conditions and different \( \varepsilon \) values. Since it is almost impossible to examine all the initial conditions (the space
of initial conditions is \( R^{12} \), we restricted our analysis to a one-dimensional problem. We fixed two points close to the wave gait and caterpillar (swim) gait limit cycles: the initial conditions for our analysis lie on the segment in \( R^{12} \) between these two points. The initial conditions considered are given in Table III, where initial conditions close to slow gait are indicated as \( x_A \), those for the swim pattern as \( x_B \) and those for caterpillar gait as \( x_C \). Fig. 3 was obtained by considering initial conditions of \( x_0 = \sigma x_C + (1 - \sigma)x_A \) for the case shown in Fig. 3(a) and \( x_0 = \sigma x_B + (1 - \sigma)x_A \) for the case shown in Fig. 3(b); Fig. 4 shows the segment of initial conditions taken into account in Fig. 3. As can be seen from Fig. 3, for a given \( \varepsilon \) value, different initial conditions (i.e., different values for the parameter \( \sigma \)) can lead to distinct attractors. Fig. 3(b) is particularly interesting since it shows how with a given \( \varepsilon \) value the steady state of the CPG system passes from the slow-gait limit cycle to caterpillar gait and finally to the swim pattern as the initial conditions pass from slow gait (\( \sigma = 0 \)) to swim pattern (\( \sigma = 1 \)). Therefore, three different limit cycles coexist for that \( \varepsilon \) value.

The presence of multiple attractors makes it impossible to find the correct value for the parameter \( \varepsilon \) with an adaptive method starting from random initial conditions, since at this point this crucially depends on the initial conditions of the 12th-order dynamical system. We therefore introduce a new CPG for slow gait: a scheme of this CPG is shown in Fig. 5. It was obtained by introducing new synapses in the CPG (9) excluding the undesired synchronization patterns. In the swim pattern, for instance, ipsilateral legs are synchronized; to avoid this gait solution for our CPG, we introduce inhibitory synapses between the neurons controlling ipsilateral legs.

**The New Slow Gait:** Since this gait is introduced here for the first time, all the templates are given, as follows:

\[
\begin{align*}
A_{11}^{\text{h}, \text{ang}} &= \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 + \mu & 0 \\ 0 & 0 & 0 \end{pmatrix}; \\
A_{21}^{\text{h}, \text{ang}} &= \begin{pmatrix} 0 & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 0 \end{pmatrix} \\
A_{12}^{\text{L}, \text{ang}} &= \begin{pmatrix} 0 & 0 & 0 \\ 0 & -s & \varepsilon_s \\ 0 & 0,2\varepsilon_s & 0 \end{pmatrix}; \\
A_{12}^{\text{R}, \text{ang}} &= \begin{pmatrix} 0 & \varepsilon_s & 0 \\ 0 & -s & 0 \\ 0 & 0,2\varepsilon_s & 0 \end{pmatrix} \\
A_{12}^{\text{L}, \text{ang}} &= \begin{pmatrix} 0 & \varepsilon_s & 0 \\ 0 & -s & 0 \\ 0 & 0,2\varepsilon_s & 0 \end{pmatrix}; \\
A_{12}^{\text{R}, \text{ang}} &= \begin{pmatrix} 0 & 0 & 0 \\ 0 & -s & 0 \\ 0 & \varepsilon_s & 0 \end{pmatrix}; \\
A_{12}^{\text{L}, \text{ang}} &= \begin{pmatrix} 0 & 0 & 0 \\ 0 & \varepsilon_s & 0 \\ 0 & 0,2\varepsilon_s & 0 \end{pmatrix}; \\
A_{12}^{\text{R}, \text{ang}} &= \begin{pmatrix} 0 & 0 & 0 \\ 0 & \varepsilon_s & -s \\ 0 & 0,2\varepsilon_s & 0 \end{pmatrix}; \\
A_{12}^{\text{L}, \text{ang}} &= \begin{pmatrix} 0 & 0 & 0 \\ 0 & -s & 0 \\ 0 & \varepsilon_s & 0 \end{pmatrix}; \\
A_{12}^{\text{R}, \text{ang}} &= \begin{pmatrix} 0 & 0 & 0 \\ 0 & \varepsilon_s & 0 \\ 0 & 0,2\varepsilon_s & 0 \end{pmatrix}; \\
A_{12}^{\text{L}, \text{ang}} &= \begin{pmatrix} 0 & 0 & 0 \\ 0 & \varepsilon_s & 0 \\ 0 & 0,2\varepsilon_s & 0 \end{pmatrix}; \\
A_{12}^{\text{R}, \text{ang}} &= \begin{pmatrix} 0 & 0 & 0 \\ 0 & \varepsilon_s & -s \\ 0 & 0,2\varepsilon_s & 0 \end{pmatrix}; \\
A_{12}^{\text{L}, \text{ang}} &= \begin{pmatrix} 0 & 0 & 0 \\ 0 & -s & 0 \\ 0 & \varepsilon_s & 0 \end{pmatrix}; \\
A_{12}^{\text{R}, \text{ang}} &= \begin{pmatrix} 0 & 0 & 0 \\ 0 & \varepsilon_s & 0 \\ 0 & 0,2\varepsilon_s & 0 \end{pmatrix}.
\end{align*}
\]
and the CG gait changes from slow gait to fast gait. The behavior of the system is labeled as follows: \( A_{22}^{L1,\text{TRG}} \) is kept, and \( A_{22}^{L2,\text{TRG}} \) changes from slow to fast gait.

Fig. 6 shows how the basins of attraction of undesired attractors are much smaller than for the CPG with templates (9).

C. Continuous Generalized Gait

Behavioral approaches based on CPG (for instance, [4] and [5]) often consider a set of basic motion patterns dealing with different forms of behavior, for example to account for forward locomotion and left and right turning patterns. These basic patterns can be simultaneously stimulated and combined to form the desired pattern. These considerations inspired the idea of fading the distinct basic gaits (i.e., slow, medium, and fast) to obtain a gait able to show continuous transitions between them. The idea consists of activating more than one set of templates at the same time, i.e., as the gait template we consider a weighted sum of the basic templates (7)-(8) and (10). However, an important difference from the behavioral approach emerges. In [4], distinct behavioral modules are simultaneously activated, while in our case we define a new set of connections. In other words, we do not sum the effects of two or more distinct units, but use a dynamical CPG network to generate the locomotion gait.

More precisely, the continuous generalized (CG) gait is defined through its feedback templates as follows:

\[
A^{\text{CG}} = \alpha A^\text{FE} + \beta A^\text{TRG} + \gamma A^\text{REG}
\]

where \( \alpha, \beta, \) and \( \gamma \) are control parameters having values between 0 and 1 and whose sum is 1, i.e., they should respect the following constraints:

\[
\begin{align*}
0 & \leq \alpha \leq 1 & (12) \\
0 & \leq \beta \leq 1 & (13) \\
0 & \leq \gamma \leq 1 & (14) \\
\alpha + \beta + \gamma & = 1. & (15)
\end{align*}
\]

When \( \alpha = 1 \) and \( \beta = \gamma = 0 \), the CG gait clearly matches the fast gait, while when \( \alpha = \gamma = 0 \) and \( \beta = 1 \) the CG gait corresponds to medium gait and when \( \alpha = \beta = 0 \) and \( \gamma = 1 \) we have slow gait. In Fig. 7, simulation results show a slow transition between the basic gaits. The waveforms of the control variables are shown in Fig. 7(d): for instance, when \( \beta \) is kept at \( \beta = 0 \) and \( \alpha \) is slowly changed from \( \alpha = 0 \) to \( \alpha = 1 \) (correspondingly, thanks to relation (15) \( \gamma \) changes from \( \gamma = 1 \) to \( \gamma = 0 \)), the CG gait changes from slow gait to fast gait.
A schematic representation of a motor map is given in Fig. 8.

The learning algorithm is the key to obtaining a spatial arrangement of both the input and output weight values of the map. This is achieved by considering an extension of the winner-take-all algorithm. At each learning step, when a pattern is given as input, the winner neuron is identified: this is the neuron that best matches the input pattern. Then, a neighborhood of the winner neuron is considered and an update involving both the input and output weights for neurons belonging to this neighborhood is performed. Even though both supervised and unsupervised learning can be applied, only unsupervised learning should be considered if an autonomous self-organizing system for high-level control of locomotion has to be defined. In this case, there is no a priori information on the appropriate control action and no teacher is available. The algorithm has to find the correct control action by itself. The only source of information is provided by the so-called reward function, introduced below, which indicates how well the control is being performed. Weight updating takes place only if the corresponding control action leads to an improvement in the system being controlled; otherwise, the neuron weights are not updated. In this framework, a fundamental role is taken by the reward function. The definition of this function is perhaps the most crucial point in the whole network design. As mentioned above, the motor map is introduced to control the speed of the robot by controlling the parameters of the CPG network. To achieve this, the definition of the reward function is straightforward and takes into account the error between the reference and actual speed of the robot.

The main difficulties of using motor maps for control are due to the fact that the learning phase may be long and especially in the transient phase the random initial values of the control law may be highly inefficient. Moreover, the feasibility of a hardware implementation is seriously conditioned by the size of the map. In order to overcome these difficulties, the solution investigated in [33], where motor maps are used to control the attitude of a hexapod robot, was to adopt a control scheme in which the motor map acts at a higher level: for instance, one can consider inner feedback loops to stabilize the system and thus reduce the number of neurons needed by the motor map as well as the effects of the initial learning phase. In our case as well, the motor map acting as a high-level control will be shown to be effective with a low number of neurons.

### B. The Control Scheme

Fig. 9 shows a block scheme of the control loop. The CPG CNN generates the locomotion gait for the robot: this gait is adaptive, and its parameters are varied by the motor map Controller (MMC). This takes as input the error between the reference speed $v_{\text{ref}}$ and the speed measured and gives as output the parameters controlling the CPG network. In this way, it is possible to obtain an adaptive locomotion algorithm, in which the gait changes with continuity from fast to slow according to the speed of the robot. The most important point is the self-learning capability of the motor map. The controller is able to learn the right control law on the basis of the reward function. As mentioned above, definition of the reward function is straightforward.

Formally, a motor map can be defined as an array of neurons mapping the space $V$ of the input patterns onto the space $U$ of the output actions

$$\Phi : V \to U. \quad (16)$$

which a topographic map is created. Neighboring neurons are thus excited by similar inputs. Successful applications of these maps have been found in the field of pattern recognition, clustering, and so on [30].

An extension of these neural structures is represented by motor maps [31]. These are networks able to react to localized excitation by triggering a movement (like the motor cortex or the superior colliculus in the brain).

To this end, motor maps, unlike Kohonen’s networks, should include storage of an output specific to each neuron site. This is achieved by considering two layers: one devoted to the storage of input weights and one devoted to output weights. The plastic characteristics of the input layer should also be preserved in the assignment of output values, so the learning phase deals with updating both the input and the output weights. This allows the map to perform tasks such as motor control. These considerations led to the idea of using motor maps as adaptive self-organizing controllers. In [32], they were applied to control chaotic systems.
Fig. 8. Schematic representation of a motor map.

Fig. 9. Block scheme of hierarchical control of the robot. The locomotion is generated at the CPG level, the MMC controls the parameters of the CPG on the basis of speed feedback.

ward in this case: it takes into account the error between the reference and actual speed of the robot as follows:

\[ \text{Reward} = -(v_{\text{ref}} - v)^2. \]  

(17)

Of course, the objective of the control is to maximize the reward function. The definition of speed for a walking machine has to be clarified: in fact, while it is unambiguous for a wheeled robot, the same does not hold for a legged robot. Walking robots do not move at a constant speed, but are subjected to accelerations due to the discrete nature of step locomotion. An average of the speed over three complete cycle times was therefore taken into account. The speed profile obtained by considering transitions in the CG gait from fast to medium and finally to slow is shown in Fig. 10.

The MMC controls the CPG by acting on the three parameters \( \alpha, \beta, \) and \( \gamma \). Since these parameters are constrained by relations (12)–(15), there are effectively two degrees of freedom on which the MMC can act. To simplify the problem, we consider a linear transformation from the space \( \beta, \alpha, \gamma \) to a new plane \( \rho, q \) in which (15) holds: details are given in Appendix II. Two output weights should, therefore, be considered for each neuron of the MMC, while the input weight stores the speed value (the input of the controller).

IV. RESULTS

The control strategy was validated on a framework for dynamic simulation built by using the DynaMechs libraries [34], [35]. A dynamic model of a hexapod robot with a mechanical design similar to that of the hexapod robot shown in [7] and in Fig. 1 was designed; this model will henceforward be referred to as HexaDyn.

A. Speed Control

In this section, the results of a complete learning phase are shown. In the first phase, the different gaits are learned one by one. The speed reference is fixed at the nominal value for each gait, while the motor map is allowed to reach the steady state after a long learning phase. In this way, the motor map learns the right parameters for slow, fast and medium gait, namely the basic locomotion patterns defined in Section II. After this first phase, one obtains specialized neurons. Typically one or two neurons will be devoted to each of the basic gaits. Then, in a second phase, intermediate speed references are given to the control system. As a result, the MMC learns to deal with intermediate gaits by updating the weights of other neurons, typically those that have not been specialized in the previous learning phases. Finally, in order to test the capabilities of the control system, a long run with a switching reference signal is carried out. This last step shows how well the MMC learning phase has been accomplished. It should, however, be pointed out that the learning is always enabled, and if a new speed is given as a reference, the motor map is able to find the appropriate new parameters, either by adjusting the weights of previously specialized neurons or by using new neurons.

The results shown in Fig. 11 refer to an MMC with 9 neurons. This was shown to be suitable for the task. Fig. 11 shows the reference signal and the actual speed of HexaDyn (as discussed above, this is an average measure). The speed is normalized with respect to the body length of HexaDyn and is expressed with respect to the time units used in the simulation. Moreover, as
HexaDyn starts from a still position, the most general case is considered.

First, the reference speed is set to \( v = 0.08 \). This corresponds to slow gait, as can be seen in the panel at \( t \approx 1200 \text{t.u.} \). This panel shows a zoom of the simulation results, illustrating the actual locomotion gait in terms of the legs that are in swing (black bars) or stance (white bars). Then the reference is switched. Other basic patterns are visible in the small panels: in particular, fast gait occurs at \( t \approx 3400 \text{t.u.} \), and medium gait at \( t \approx 5700 \text{t.u.} \). The long transient phase before the fast gait speed is reached is due to the fact that a new neuron (and not the one previously specialized for fast gait) is used for the control, so the MMC has to learn the right weights for the fast gait. Several examples of intermediate patterns are also shown: for instance, the panels at \( t \approx 2200 \text{t.u.} \), and \( t \approx 4600 \text{t.u.} \), are intermediate patterns between fast and medium gait and between medium and slow gait, respectively. Finally, the panel at \( t \approx 3800 \text{t.u.} \) shows an example of a transition between fast gait and an intermediate pattern. It may be noticed that the transition is less smooth than in the case of Fig. 7; this is due to the fact that after the transition, a MMC neuron, with output weights totally different from those of the neuron previously activated, is used for the control. Moreover, it is worth noting that when the slow gait speed is given as a reference for the second time (last panel at \( t \approx 7200 \text{t.u.} \)), the transient phase is very short thanks to the fact that the MMC has learned very well how to control the CPG to generate this locomotion gait.

The weights of the MMC neurons at the end of the simulation are shown in Fig. 12. The input and the two output weights are shown for each neuron. The neurons are ordered by increasing input weight, i.e., HexaDyn speed, values. As can be observed, the neurons labeled as 5, 6, and 9 are devoted to the generation of slow, medium and fast gait, respectively. The fast gait neuron, for instance, has a very high value for the output weight controlling parameter \( p \) and a low value for the output weight controlling parameter \( q \). Taking into account (25), this leads to \( \alpha \approx 1, \beta \approx 0, \) and \( \gamma \approx 0 \), which is exactly what one expects to find. The same holds for slow gait, while for medium gait one expects to find negative values for the variable \( p \). Since this is not the case, it appears evident that the MMC finds another possible solution for the problem. Moreover, it can be noticed that neurons 1–4 are devoted to dealing with low velocities; thus, they are useful in the start-up phase.

B. The Problem of Learning

The problem discussed above can be reconsidered from another viewpoint by reducing the number of parameters to be investigated. If only one basic locomotion gait (i.e., slow, medium or fast) is considered and so \( \epsilon \) is the unknown parameter, the MMC can be used to allow the robot to learn the locomotion gait: the robot starts from unorganized leg movements and learns how to synchronize the legs in order to achieve a pattern of coordinated movements. Let us for instance focus on fast gait: we set \( \alpha = 1, \beta = 0, \) and \( \gamma = 0 \) and take the parameter \( \epsilon = -\epsilon_f \) as the unknown parameter. Unlike the case examined previously, the motor map output weights are now scalar variables and directly map the values of the parameter \( \epsilon \). Moreover, in this case the learning rate is constant (it does not require the specific precaution adopted in the case of GC gait—see Remark 1 in Appendix II).

Fig. 13(a) shows the trend of the parameter \( \epsilon \). It should be pointed out that, as learning is always active, small fluctuations around the average value are present. Moreover, the average value is similar to the value given in Table II. Fig. 13(b) illustrates the winning neuron. In this case two neurons have been trained for the fast gait. This is not a general result, since in most cases only one neuron is trained. However, this redundancy is due to the fact that only one reference input is presented to the system, so further learning steps can lead to a higher specialization.

V. Conclusion

In this paper, an approach based on dynamical systems for the control of locomotion in bio-inspired robots, and in particular in an insect-like hexapod robot, has been presented. The structure
Fig. 12. Weights for each MMC neuron after simulation run. Input weights map the input pattern (i.e., the speed of the hexapod), while the output weights map the parameters of the CG gait in the plane $p - q$.

Fig. 13. Learning a given locomotion gait (fast gait): (a) trend of the parameter $\varepsilon$ and (b) winning neuron.

of the control system inspired by the biological paradigm of the CPG, with a hierarchical organization, allows us to propose an innovative strategy for both generation and control of the locomotion gait.

The CNN architecture has been shown to be suitable for locomotion generation, which takes place at the level of a system of coupled nonlinear units implemented by a CNN. In particular, the possibility of changing the locomotion gait is achieved thanks to the programmable interconnections of the architecture devoted to control. The main innovative results of the paper concern the higher level adaptive controller included in the locomotion control system, which has been shown to be able to find the parameters of the CPG structure and to choose the locomotion gait according to environmental conditions. This higher level system is implemented by motor maps and is able to learn the correct locomotion gait on the basis of a reward function.

This allows us to achieve two important results: the first concerns the possibility of learning the locomotion gait on-line; the second is related to the capability to adapt gaits to the environmental conditions. The proposed control system has been validated by a large number of simulations carried out in a dynamic environment for legged robots. These simulations show good performance by the controller.

It should be remarked that the scheme introduced in this paper is not an evolved controller. The structure of the control system is known a priori. In fact, the design of the CNN-based CPG can easily be addressed by using simple principles inspired by the analogy with the biological case [22], but numerical simulations are needed to find the proper parameters. The approach based on motor maps allows us to find these parameters by on-line learning, and to design an adaptive controller able to control the locomotion gait and the choice of the locomotion gait. These
characteristics lead us to real-time learning. It may be objected that the learning phase requires quite a large number of epochs. Up to now, this seems unavoidable, because learning how to coordinate movements is the result of a long and complex process.

**APPENDIX I**

**CNNs**

CNNs were introduced by Chua [21], [36] in 1988. His idea was to use an array of simple, identical, locally interconnected nonlinear dynamical circuits, called cells, to build large-scale analog signal processing systems. The cell was defined as the nonlinear first-order circuit shown in Fig. 14(a). Each cell mutually interacts with its nearest neighbors by means of the voltage controlled current sources

\[ I_{x_0}(i,j;k,l) = A(i,j;k,l)u_{kl} \]

and

\[ I_{x_1}(i,j;k,l) = B(i,j;k,l)u_{kl} \]

The constant coefficients \( A(i,j;k,l) \) and \( B(i,j;k,l) \) are known as the cloning templates: if they are equal for each cell, they are called space-invariant. More specifically, the template \( A \) is known as the feedback template.

A CNN is described by the state equations of all cells

\[
C \cdot \frac{dx_{ij}}{dt} = -\frac{1}{R_x} x_{ij}(t) + \sum_{C(k,l) \in N_i(i,j)} A(i,j;k,l) y_{kl} + \sum_{C(k,l) \in N_j(i,j)} B(i,j;k,l) u_{kl} + I_{ij} \tag{19}
\]

with \( i = 1,2,\ldots, M \) and \( j = 1,2,\ldots, N \), where

\[ N_i(i,j) = \{ C(k,l) \mid \max(|k-i|,|l-j|) \leq r \} \]

with \( k = 1,2,\ldots, M \) and \( l = 1,2,\ldots, N \) is the \( r \)-neighborhood.

This model is known as the Chua–Yang model or linear CNN. The Chua–Yang model has been generalized in many different ways. These generalizations allow the inclusion in the model (19) of nonlinear interactions, direct dependence on the state variables of the neighborhood cells and different grids, and lead to a more general definition for CNNs [37]: a CNN is an \( n \)-dimensional array of mainly identical dynamic systems, called cells, which satisfies two properties: 1) most interactions are local within a finite radius \( r \) and 2) all state variables are continuous valued signals. It follows that a more complete CNN model including some of the above-mentioned generalizations is described by the following state equations (setting \( C = 1, R_x = 1 \)):

\[
\frac{dx_{ij}}{dt} = -x_{ij}(t) + \sum_{C(k,l) \in N_i(i,j)} \{ \hat{A}_{ij;k,l}(y_{kl}(t),y_{ij}(t)) \\
+ \hat{B}_{ij;k,l}(u_{kl}(t),u_{ij}(t)) \} + \hat{C}_{ij;k,l}(x_{kl}(t),x_{ij}(t)) + I_{ij} \tag{20}
\]

with

\[ y_{ij} = f(x_{ij}) \]

where \( \hat{A}_{ij;k,l}(\cdot,\cdot), \hat{B}_{ij;k,l}(\cdot,\cdot) \) and \( \hat{C}_{ij;k,l}(\cdot,\cdot) \) are two-variable nonlinear functions (the nonlinear templates) and \( f(\cdot) \) is the output nonlinearity.

Significant attention has been directed to studying the dynamic properties of the various CNN models. One of the most challenging issues is surely stability [21]. In fact, the particular structure, high order and nonlinearity of these systems create serious problems. Almost all kinds of dynamic behavior, ranging from simple equilibria to chaos, have been reported among the different kinds of networks.

![Fig. 14. (a) The CNN cell scheme. (b) A bidimensional CNN array.](image)

![Fig. 15. (a) Characteristics of parameter \( a_x \) as a function of the actual error. (b) \( \eta \) is varied as time goes by to speed up the learning phase.](image)
It is now clear that CNN behavior is basically dictated by the templates. However, the choice of templates that are suitable to achieve a desired processing task is hard to accomplish in a direct way.

This leads to the so-called learning and design problem [39], [40]. The term design is used when the desired task can be translated into a set of local dynamic rules, while the term learning is used when the templates need to be obtained by learning techniques, so pairs of inputs and outputs must correspond. Good results have been obtained with discrete-time CNNs in simple cases, but this is a really difficult problem for continuous-time models. Most of the templates currently available were obtained by intuitive principles and refined by trial and error with the aid of simulators.

**APPENDIX II**

**UNSUPERVISED LEARNING FOR MOTOR MAPS**

The unsupervised learning algorithm for the motor map can be described in the following five steps.

Step 1) In the first step, the topology of the network is established. The number of neurons is chosen and the reward function is established. The number of neurons needed for a given task is chosen by a trial-and-error strategy, thus once numerical results indicate that the number of neurons is too low, one must return to this step and modify the dimensions of the map. At this step the weights of the map are randomly fixed.

Step 2) An input pattern is presented and the neuron whose input weight best matches it is established as the winner. Therefore, to establish the winner neuron, the distance between the neuron input weight and the input pattern is computed for each neuron, considering the absolute value of the difference between the weights and the index $i$ takes into account the neighborhood of the winner neuron. In supervised learning, $f(t)$ is the target, while in unsupervised learning it is varied, as discussed above.

Step 3) Once the winner neuron has been chosen, its output weight is used to perform the control action $f(t)$. This is not used directly, but a random variable is added to the value to guarantee a random search for possible solutions, as follows:

$$f(t) = w_{\text{winner, out}} + a_s \lambda$$

where $w_{\text{winner, out}}$ is the output weight of the winner neuron, $a_s$ is a parameter determining the mean value of the search step for the neuron $s$, and $\lambda$ is a Gaussian random variable with a zero mean. Then the increase $\Delta R$ in the reward function is computed and, if this value exceeds the average increase $b_s$ gained at the neuron $s$, the next step (updating of the output weights) is performed; otherwise this step is skipped. The mean increase in the reward function is updated as follows:

$$b_s^{\text{new}} = b_s^{\text{old}} + \rho (\Delta R - b_s^{\text{old}})$$

where $\rho$ is a positive value. Moreover, $a_s$ is decreased as more experience is gained (this holds for the winner neuron and for the neighboring neurons), according to the following rule:

$$a_i^{\text{new}} = a_i^{\text{old}} + \eta b \left( a - a_i^{\text{old}} \right)$$

where $i$ indicates the generic neuron to be updated (the winner and its neighbors), $a$ is a threshold the search step should converge to, and $\eta b$ is the learning rate, while $\xi$ takes into account the fact that the parameters of the neurons to be updated are varied by different amounts, defining the extent and the shape of the neighborhood.

Step 4) If $\Delta R > b_s$, the weights of the winner neuron and those of its neighbors are updated following the rule:

$$w_{i, \text{in}}(t+1) = w_{i, \text{in}}(t) + \eta \xi (v(t) - w_{i, \text{in}}(t))$$

$$w_{i, \text{out}}(t+1) = w_{i, \text{out}}(t) + \eta \xi (f(t) - w_{i, \text{out}}(t))$$

where $\eta$ is the learning rate, $\xi v, w_{\text{in}}$, and $w_{\text{out}}$ are the neighborhood function, the input pattern, the input weights and the output weights, respectively.

Step 5) Steps 2)–4) are repeated. If one wishes to preserve a residual plasticity for a later re-adaptation, by choosing $a \neq 0$ in step 3), the learning is always active and so steps 2)–4) are always repeated. Otherwise, by setting $a = 0$, the learning phase stops when the weights converge.

**Remark 1:** Some adaptations of the algorithm have been taken into account to achieve real-time features. These adaptations mainly involve the parameters $a_s$ and $\eta$. We assume that $a_s$ is function of the actual error $e = v - \hat{v}_{\text{ref}}$, so that when the error is low, small stochastic changes in the weights (i.e., in the control action) are considered, while when large errors occur the random search term is greater. Moreover, to speed up the algorithm, the learning rate $\eta$ is a function of the simulation time, so that in the initial phase (when a new reference speed is taken into account) a high learning rate guarantees rapid learning, while as time goes on the low learning rate guarantees a good tradeoff between memory and innovation capabilities. This is not needed when only one parameter is to be learned (Section IV-B).

Moreover, the neighborhood function $\xi$ is assumed to be unitary. This means adopting a winner-takes-all strategy, simplifying the learning phase. Finally, the stochastic term is implemented by a chaotic circuit used as a noise generator as in [32]. This simplification was introduced in view of a hardware implementation of the whole control algorithm.

**Remark 2:** $\alpha, \beta$, and $\gamma$ are controlled by the MMC. They should be updated taking into consideration the constraints (12)–(15): $\alpha, \beta$, and $\gamma$ lie on the triangle of Fig. 16. Thus, in effect, the problem has two degrees of freedom. As the weights in the motor map algorithm are updated according to (21), which considers a small random term, in this case one should guarantee that adding the random term respects the above conditions. Therefore, to simplify the problem, we consider

**APPENDIX II**

**UNSUPERVISED LEARNING FOR MOTOR MAPS**

The unsupervised learning algorithm for the motor map can be described in the following five steps.

Step 1) In the first step, the topology of the network is established. The number of neurons is chosen and the reward function is established. The number of neurons needed for a given task is chosen by a trial-and-error strategy, thus once numerical results indicate that the number of neurons is too low, one must return to this step and modify the dimensions of the map. At this step the weights of the map are randomly fixed.

Step 2) An input pattern is presented and the neuron whose input weight best matches it is established as the winner. Therefore, to establish the winner neuron, the distance between the neuron input weight and the input pattern is computed for each neuron, considering the absolute value of the difference between these two vectors.

Step 3) Once the winner neuron has been chosen, its output weight is used to perform the control action $f(t)$. This is not used directly, but a random variable is added to the value to guarantee a random search for possible solutions, as follows:

$$f(t) = w_{\text{winner, out}} + a_s \lambda$$

where $w_{\text{winner, out}}$ is the output weight of the winner neuron, $a_s$ is a parameter determining the mean value of the search step for the neuron $s$, and $\lambda$ is a Gaussian random variable with a zero mean. Then the increase $\Delta R$ in the reward function is computed and, if this value exceeds the average increase $b_s$ gained at the neuron $s$, the next step (updating of the output weights) is performed; otherwise this step is skipped. The mean increase in the reward function is updated as follows:

$$b_s^{\text{new}} = b_s^{\text{old}} + \rho (\Delta R - b_s^{\text{old}})$$

where $\rho$ is a positive value. Moreover, $a_s$ is decreased as more experience is gained (this holds for the winner neuron and for the neighboring neurons), according to the following rule:

$$a_i^{\text{new}} = a_i^{\text{old}} + \eta b \left( a - a_i^{\text{old}} \right)$$

where $i$ indicates the generic neuron to be updated (the winner and its neighbors), $a$ is a threshold the search step should converge to, and $\eta b$ is the learning rate, while $\xi$ takes into account the fact that the parameters of the neurons to be updated are varied by different amounts, defining the extent and the shape of the neighborhood.

Step 4) If $\Delta R > b_s$, the weights of the winner neuron and those of its neighbors are updated following the rule:

$$w_{i, \text{in}}(t+1) = w_{i, \text{in}}(t) + \eta \xi (v(t) - w_{i, \text{in}}(t))$$

$$w_{i, \text{out}}(t+1) = w_{i, \text{out}}(t) + \eta \xi (f(t) - w_{i, \text{out}}(t))$$

where $\eta$ is the learning rate, $\xi v, w_{\text{in}}$, and $w_{\text{out}}$ are the neighborhood function, the input pattern, the input weights and the output weights, respectively, and the index $i$ takes into account the neighborhood of the winner neuron. In supervised learning, $f(t)$ is the target, while in unsupervised learning it is varied, as discussed above.

Step 5) Steps 2)–4) are repeated. If one wishes to preserve a residual plasticity for a later re-adaptation, by choosing $a \neq 0$ in step 3), the learning is always active and so steps 2)–4) are always repeated. Otherwise, by setting $a = 0$, the learning phase stops when the weights converge.

**Remark 1:** Some adaptations of the algorithm have been taken into account to achieve real-time features. These adaptations mainly involve the parameters $a_s$ and $\eta$. We assume that $a_s$ is function of the actual error $e = v - \hat{v}_{\text{ref}}$, so that when the error is low, small stochastic changes in the weights (i.e., in the control action) are considered, while when large errors occur the random search term is greater. Moreover, to speed up the algorithm, the learning rate $\eta$ is a function of the simulation time, so that in the initial phase (when a new reference speed is taken into account) a high learning rate guarantees rapid learning, while as time goes on the low learning rate guarantees a good tradeoff between memory and innovation capabilities. This is not needed when only one parameter is to be learned (Section IV-B).

Moreover, the neighborhood function $\xi$ is assumed to be unitary. This means adopting a winner-takes-all strategy, simplifying the learning phase. Finally, the stochastic term is implemented by a chaotic circuit used as a noise generator as in [32]. This simplification was introduced in view of a hardware implementation of the whole control algorithm.

**Remark 2:** $\alpha, \beta$, and $\gamma$ are controlled by the MMC. They should be updated taking into consideration the constraints (12)–(15): $\alpha, \beta$, and $\gamma$ lie on the triangle of Fig. 16. Thus, in effect, the problem has two degrees of freedom. As the weights in the motor map algorithm are updated according to (21), which considers a small random term, in this case one should guarantee that adding the random term respects the above conditions. Therefore, to simplify the problem, we consider
two coordinates \( p \) and \( q \) in the triangle and the following transform from the plane \( p-q \) to the space \( \beta, \alpha, \gamma \):

\[
\begin{align*}
\alpha &= \frac{1}{2} + \frac{1}{2\sqrt{3}} p - \frac{1}{\sqrt{3}} q \\
\beta &= \frac{1}{2} - \frac{1}{2\sqrt{3}} \gamma \\
\gamma &= \sqrt{\frac{3}{q}} q
\end{align*}
\]

(25)

**Fig. 16. Coordinates in the dashed triangle.**

---

**REFERENCES**


---

**Paolo Arena** received the electronic engineering degree and the Ph.D. degree in electrical engineering in 1990 and in 1994, respectively, from The University of Catania, Catania, Italy. He is currently an Associate Professor of System Theory. He has published more than 130 technical papers and several international patents. His research interests include adaptive and learning systems, nonlinear systems, neural networks, collective behavior in living and artificial neural structures, and bio-image analysis. He is an Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—PART I: FUNDAMENTAL THEORY AND APPLICATIONS.
Luigi Fortuna (F’02) has been a Full Professor of System Theory at the University of Catania, Catania, Italy, since 1994. He has published more than 280 technical papers and is co-author of six books, including Cellular Neural Networks (Berlin, Germany: Springer, 1999). He holds several U.S. patents. His scientific interests include nonlinear science and complexity, chaos, and cellular neural networks with applications in bioengineering.

Mattia Frasca (S’01–M’04) was born in Siracusa, Italy, in 1976. He received the electronics engineering degree in 2000 and received the Ph.D. degree in 2003 in electronics and automation engineering from the University of Catania, Catania, Italy.

Currently, he is a Lecturer at the University of Catania. His scientific interests include nonlinear systems and chaos, cellular neural networks, and bio-inspired robotics.

Giovanni Sicurella was born in Catania, Italy, in 1973. He received the computer science engineering degree in 2003 from the University of Catania.

Currently, he is a System Engineer at the SST Group, Corporate R&D of STMicroelectronics, Catania. His research interests include bio-inspired robotics, cellular neural networks, and control and synchronization of biological complex dynamical systems and networks.