frontiers in Neurorobotics

ORIGINAL RESEARCH
published: 28 July 2016
doi: 10.3389/fnbot.2016.00006

Design of Spiking Central Pattern Generators for Multiple Locomotion Gaits in Hexapod Robots by Christiansen Grammar Evolution

Andres Espinal[1], Horacio Rostro-Gonzalez[2], Martin Carpio[1], Erick I. Guerra-Hernandez[2], Manuel Ornelas-Rodriguez[1] and Marco Sotelo-Figueroa[3]*

[1] Leon Institute of Technology, Leon, Mexico, [2] Department of Electronics, DICIS-University of Guanajuato, Salamanca, Mexico, [3] Department of Organizational Studies, División de Ciencias Economico-Administrativas-University of Guanajuato, Guanajuato, Mexico

This paper presents a method to design Spiking Central Pattern Generators (SCPGs) to achieve locomotion at different frequencies on legged robots. It is validated through embedding its designs into a Field-Programmable Gate Array (FPGA) and implemented on a real hexapod robot. The SCPGs are automatically designed by means of a Christiansen Grammar Evolution (CGE)-based methodology. The CGE performs a solution for the configuration (synaptic weights and connections) for each neuron in the SCPG. This is carried out through the indirect representation of candidate solutions that evolve to replicate a specific spike train according to a locomotion pattern (gait) by measuring the similarity between the spike trains and the SPIKE distance to lead the search to a correct configuration. By using this evolutionary approach, several SCPG design specifications can be explicitly added into the SPIKE distance-based fitness function, such as looking for Spiking Neural Networks (SNNs) with minimal connectivity or a Central Pattern Generator (CPG) able to generate different locomotion gaits only by changing the initial input stimuli. The SCPG designs have been successfully implemented on a Spartan 6 FPGA board and a real time validation on a 12 Degrees Of Freedom (DOFs) hexapod robot is presented.

Keywords: central pattern generator, spiking neural network, Christiansen grammar evolution, evolution strategy, SPIKE-distance, legged robot locomotion, FPGA

OPEN ACCESS

Edited by:
Quan Zou,
George Washington University, USA

Reviewed by:
Nicolas Van der Noot,
Université catholique de Louvain, Belgium
Yuanfeng Zhu,
BorderX Lab Inc, USA

****Correspondence:***
Horacio Rostro-Gonzalez
hrostrog@ugto.mx

Received: *08 April 2016*
Accepted: *11 July 2016*
Published: *28 July 2016*

Citation:
Espinal A, Rostro-Gonzalez H, Carpio M, Guerra-Hernandez EI, Ornelas-Rodriguez M and Sotelo-Figueroa M (2016) Design of Spiking Central Pattern Generators for Multiple Locomotion Gaits in Hexapod Robots by Christiansen Grammar Evolution. Front. Neurorobot. 10:6. doi: 10.3389/fnbot.2016.00006

1. INTRODUCTION

Since the early twentieth century, studies have been carried out to explain how the rhythms of locomotor movements in living beings are created (Brown, 1911). It was proposed that the action of walking is carried out by neural mechanisms, in which neurons are inhibiting each other to achieve the control of muscles achieving a rhythmic movement (Brown, 1914). Nowadays there is evidence supporting this idea, behavior-based studies of living beings have demonstrated that neural mechanisms, known either as neural oscillators or CPGs, contribute to locomotion. Although it has been experimentally demostrated that CPGs can endogenously produce rhythmic motor outputs, they do not work isolatedly; CPGs also depend on the information interaction with other parts of the central nervous system (Arena, 2000). Moreover, afferent sensory inputs are used to shape the final motor output (MacKay-Lyons, 2002). CPGs produce other rhythmic behaviors without conscious effort besides locomotion, including respiration, heart beat, swallowing, etc. (Patel, 2009).

Studies about CPGs have produced several mathematical models of such mechanisms, which have been used in both theoretical and practical fields for different purposes. Recently, in robotics, there has been an increasing interest in the design and implementation of biologically inspired CPG-based locomotion systems (Russell et al., 2007; Crespi and Ijspeert, 2008; Wyffels and Schrauwen, 2009; Barron-Zambrano and Torres-Huitzil, 2012; Chen et al., 2012; Hong et al., 2014; Nassour et al., 2014; Park et al., 2014; Rostro-Gonzalez et al., 2015) with rhythmic motions instead of non-biologically plausible methods such as those based on finite-state machines, sine-generators, pre-recorded reference trajectories (Vukobratović and Borovac, 2004) or heuristic control laws (Pratt et al., 2001). There are features of CPGs, which make them suitable as locomotion systems in robotic controllers (Yu et al., 2014); they ensure a uniform and steady rhythm over course of locomotion, they possess stability that makes them robust against disturbances, they can be modified by the sensory feedback signals by means of their behavioral adaptability and one of them can generate different motor behaviors by switching between behaviors arising from changes in parameters.

Even though several implementations of CPG-based locomotion systems for robots have been reported in the state of the art [see Ijspeert (2008), Wu et al. (2009), and Yu et al. (2014) for detailed reviews on CPG research], there is a lack of a well-established design methods for CPG systems (Ijspeert, 2008); however, a generic framework for designing CPGs is proposed in Yu et al. (2014) based on three main aspects on which most CPG studies have focused (see **Figure 1**):

1. *CPG Modeling and Analysis*: This task deals with choosing the type of neuron or oscillator, the kind of coupling (unidirectional or bidirectional connection), and the structure of the connections.
2. *Modulation of CPGs*: In engineering, two components of CPG modulation are used: the parameter tuning and the gait transition. The former is usually achieved by using trial-and-error optimization methods (deterministic and stochastic). The latter component deals with methods for handling several gaits generated by a CPG. Some common approaches implemented to handle the generation of several gaits for CPGs are: changing the pattern of connectivity, varying the properties of the oscillators (reconfiguration), altering the driving signal to the CPG, and using a transient disturbance (environmental adaptability).
3. *System Implementation*: CPG-based locomotion systems can be programed in software and they can run on a microcontroller or in a reconfigurable hardware, such as field-programmable gate array (FPGAs) or neuromorphic systems.

Lately, a CPG development method for hexapod robot locomotion systems, which defines the CPG modeling (network topology) and the CPG parameter tuning by means of a reverse-engineering approach to estimate the parameters of a neural network has been developed (Rostro-Gonzalez et al., 2012); in that study, gait transition is made by changing connectivity patterns because a CPG is created for each gait and finally, they are tested on hardware (Rostro-Gonzalez et al., 2015). In this study, we propose a method based on the CPG design published in

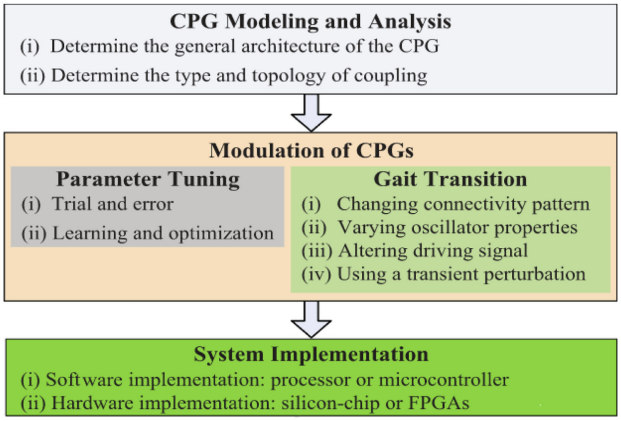


FIGURE 1 | Workflow diagram of CPG design taken from Yu's work (Yu et al., 2014).

the aforementioned study. Our design methodology is based on Christiansen Grammar Evolution (CGE) (Ortega et al., 2007), a kind of optimization algorithm with indirect representation of solutions, which can be used for the development of the Evolutionary Artificial Neural Networks (Yao, 1999; Ding et al., 2013); by using CGE the method dispenses with predefined topology and avoids an explicit training process, which is a difficult task because certain parameters need to be estimated to replicate locomotor patterns (Ijspeert, 2008; Buschmann et al., 2015). CGE defines the presynaptic connections and the weights of a spiking neuron (see Section 2.2.1); once a spiking neuron is connected, its capability of replicating a specific signal is valued by SPIKE-distance (Kreuz et al., 2013). The methodology integrates all the individual design to define a whole CPG. We are capable of generating compact CPG topologies and to create a CPG which generate different gaits.

In legged robots, locomotion can be performed by CPGs, which are mainly described by oscillators or artificial neurons with a lack of biological plausibility, e.g., connectionist models, vector maps and systems of coupled oscillators (Ijspeert, 2008). Recently, there are few efforts to implement SCPGs as locomotion controllers in robotics. SPCGs are built as Spiking Neural Networks (SNNs), the third generation of Artificial Neural Networks (ANNs) (Maass, 1997); these are formed by spiking neurons, whose models are biologically plausible and can process spatio-temporal information naturally as required for rhythmic movements. Some SCPGs have been designed and implemented for locomotion on biped and hexapod robots; in Lewis et al. (2005), an architecture of spiking neurons to generate walking gaits for a biped robot was proposed. Later, Russell et al. (2010) proposed to implement a Genetic Algorithm (GA) to reconfigure weights and network topology of Lewis' network online for changing the locomotion on the same kind of robot. More recently in Rostro-Gonzalez et al. (2015), SNNs are configured as CPGs based on an analysis of six-legged insects' gaits for hexapod robot locomotion. In this work, SCPG-based locomotion systems are designed to imitate different gaits observed in hexapod insects, which are implemented in an FPGA Spartan 6 board and tested on hexapod robots.

There are reasons that encourage the study, design, and implementation of SCPGs, e.g. the advances on interfacing prosthetic

robotic devices to amputated humans and spinal injury patients; and because SCPG are made of SNNs, which receive and process the same kind of information as the CPGs on their biological counterpart, they are a reliable and viable option (Russell et al., 2007). The structure of the paper is as follows. In Section 2, the proposed methodology is presented. In Section 3, we present the experimental configuration and the numerical results. Finally, in Section 4, the conclusions and highlights of this study are given.

2. MATERIALS AND METHODS

In this section, we introduce the design methodology of SCPGs and all the required methods for their development. The proposal is an off-line methodology, it is, the algorithm for SCPG design was implemented using JAVA as its programing language then we developed hardware architecture based on VHDL (VHSIC Hardware Description Language) for FPGA targeting and real time simulation on a hexapod robot. The design methodology is a system of inputs, design process and output. The inputs are sets of rhythmic patterns for locomotion of hexapods (see Section 2.1), each of them explicitly defining the periodic signal for each spiking neuron into the SCPG. The design process is based on the divide-and-conquer approach, where a problem is divided into subproblems that are independently solved to be combined into a solution of a whole (Puntambekar, 2008). Here, the problem is to design an SNN (see Section 2.2) that endogenously replicates the input rhythmic patterns. Since the SCPG is built as an SNN, it needs to replicate specific rhythmic patterns to contribute to locomotion, and thus it is very important that each neuron may be able to replicate its expected signal periodically through the connections with other neurons (or even with itself). The design of an SNN for producing rhythmic patterns is achieved by dividing the general design of the SNN into individual ones for each spiking neuron. The spiking neuron design is created by using its initial rhythmic signal's state and the signals from other neurons through CGE (see Section 2.3), which codifies information into solutions such as connections and weights; then, they are perturbed until the spiking neuron replicates the expected signal. Several design specifications can be integrated into the fitness function of the CGE. The output is an SCPG obtained by integrating all the individual neuron designs. Finally, the designed SCPGs are implemented on a Spartan 6 FPGA board and tested on an hexapod robot (see Section 2.4).

2.1. Hexapod Locomotion Gaits

The locomotion of hexapods is achieved by designing SCPGs that are periodically able to replicate rhythmic signals, which contribute to imitate gaits observed in real hexapod insects. To drive the SCPG designs, samples from the rhythmic signals are required; here, three different gait patterns are used [proposed in Rostro-Gonzalez et al. (2015)]. These biologically based patterns are the result of a study about the inter-leg coordination of stick insect (*carausius morosus*) on free walks (Grabowska et al., 2012); in this study, two tetrapod gaits and one tripod gait were reported, labeled in Rostro's work as walking gait (**Figure 2A**), jogging gait (**Figure 2B**), and running gait (**Figure 2C**), respectively. In **Figure 2**, the three locomotion patterns are presented, where the x and y axes show the scale of time (as reference) and neuron labels (according with **Figure 4**), respectively. In such figure, each bar represents the neural activity that stimulates a servomotor in the robot, where Coxa-Left (CL) and Coxa-Right (CR) from 1 to 3 correspond to the servomotors for the coxa (hip joint) of the robot, it is, the articulation in charge of the movement from back to front and viceversa. On the other hand, Femur-Left (FL) and Femur-Right (FR) from 1 to 3 correspond to the servomotors for the femur of the robot, it is, the articulation in charge of the movement from down to up and viceversa. The coordinate movements of these two articulations perform the expected locomotion gaits such as those shown in **Figure 2**.

2.2. Spiking Neural Network

Herein, the CPGs are built as SNNs, similar to other ANNs, and can be defined around three aspects: neuron model, synaptic connections, and message types (Judd, 1990). For this study, we used the simplest form of the integrate-and-fire spiking neuron model, which is based on the discrete-time representation of the membrane potential of the neuron and called BMS neuron model (Soula et al., 2006) (see Section 2.2.1), the synaptic connections are both excitatory (positive values) and inhibitory (negative values) and they are defined by the CGE (see Section 2.3) for each neuron, and finally the message types are spike times.

2.2.1. BMS Neuron Model

The BMS spiking neuron model (Soula et al., 2006) is a discrete-time version of the best-known and widely used generalized Integrate-and-Fire model (gIF) (Gerstner and Kistler, 2002). In the BMS model, the membrane potential V_i and the firing state

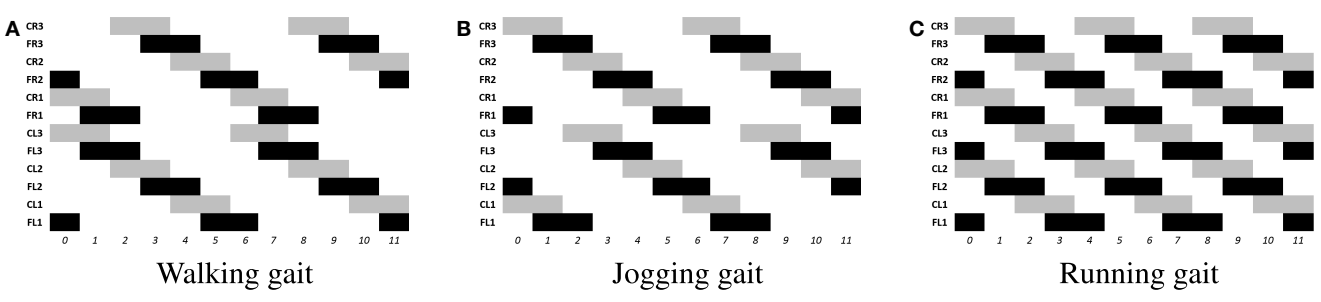


FIGURE 2 | Rhythmic patterns for hexapod locomotion. The black bars correspond to the gait patterns reported in Grabowska et al. (2012), used for moving the femurs, while the gray bars are added in Rostro-Gonzalez et al. (2015) as additional information for moving the coxas and thus achieving the full locomotion in hexapod individuals. **(A)** Walking gait. **(B)** Jogging gait. **(C)** Running gait.

Z_i of the ith neuron at time k are given by equations (1) and (2), respectively.

$$V_i[k] = \gamma V_i[k-1](1 - Z_i[k-1]) + \sum_{j=1}^{N} W_{ij} Z_j[k-1] + I_i^{ext} \tag{1}$$

$$Z_i[k] = \begin{cases} 1 & \text{if } V_i[k] \geq \theta \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

where $\gamma \in [0, 1]$ defines the leak rate. N is the number of neurons in the neural network. W is the matrix of synaptic weights. Finally $I^{(ext)}$ represents an external stimulus, but here $I^{(ext)} = 0$ because SCPGs endogenously generate the rhythmic patterns. When $V_i[k]$ reaches a given threshold θ, then a spike occurs in $Z_i[k]$ equation (2), and the neuron i is reset by the term $(1 - Z_i[k])$ in equation (1).

2.3. Christiansen Grammar Evolution

The Christiansen Grammar Evolution (CGE) (Ortega et al., 2007), like the Grammatical Evolution (GE) (Ryan et al., 1998), is a grammar-based form of Genetic Programing (GP) (Koza, 1992). CGE extends the capabilities of GE in the sense that it can generate both syntactically and semantically correct programs. This is achieved by replacing the Context-Free Grammars with Christiansen Grammars (CG) (Christiansen, 1985).

The CGE can be explained from the basis of GE as follows: the genotypic representation of individuals is lineal, formed by strings of numeric values. These are changed from their genotype representation to their functional phenotypic representation through a mapping process (Dempsey et al., 2009) (also known as indirect representation), which uses a grammar to derive the phenotypic representation of an individual. The search process is made using a search engine (usually a metaheuristic algorithm), which modifies the genotype of the individual with only the knowledge of its fitness value. Since CGE has been inspired from GE, their work flows are similar with the exception of the kind of input grammars and some extra steps on the mapping process (see Sections 2.3.1 and 2.3.2, respectively).

Recently, based on the fact that several algorithms have been used as GE search engines [i.e., Genetic Algorithm (Ryan et al., 1998), Differential Evolution (O'Neill and Brabazon, 2006a), Particle Swarm Optimization (O'Neill and Brabazon, 2006b)], a generic methodology for implementing GE was suggested, which points out their input conditions (problem instance, BNF grammar, and search engine) and process cycle (output or individual's phenotypic representation given by the mapping process and its evaluation by using a fitness function) (Sotelo-Figueroa et al., 2014). From the relationship between GE and CGE, this proposal can be easily adapted for CGE by changing the kind of input grammar and the mapping process.

2.3.1. Christiansen Grammars

The Christiansen Grammars (CGs) (Christiansen, 1985) are adaptable grammars, which can be modified on the fly while they are being used. According to Shutt's work (Shutt, 1993), CGs are very similar to Extended Attribute Grammars (Watt and Madsen, 1983); CGs are defined as a 5-tuple as follows: $CG = \{\Sigma_T, \Sigma_N, S, P, K\}$, where Σ_T is the set of terminals, Σ_N is the set of non-terminals, $S \in \Sigma_N$ is the axiom of the grammar, P is the productions set, and K is the global information of the grammar.

The key features of the CGs syntax (Ortega et al., 2007) are:

- The non-terminal symbols are written between angled brackets and they are followed by a parenthesized list of their attributes. The value of any attribute can be computed while the grammar is being used. The first attribute of each non-terminal is a CG, which contains the applicable rules to the corresponding symbol.
- The attributes can be either inherited (↓) or synthesized (↑).
- In the production rules, the semantic actions follow their corresponding production rule in brackets, where {} stands for non-semantic action. These actions are usually written in pseudocode.

$$\underbrace{n}_{synaptic\ connections} : \overbrace{id_{1-st}, weight_{1-st}}^{1-st\ configured\ synapse} \mid \ldots \mid \overbrace{id_{n-th}, weight_{n-th}}^{n-th\ configured\ synapse}$$

FIGURE 3 | Structure of the derived word for presynaptic connections of a neuron.

For this study, we propose a CG in order to derive words that represent the presynaptic connectivity and configuration of a single spiking neuron. The neuron can be connected with itself and other neurons. However, a neuron can only allow one connection per neuron in the network; this means that a neuron must have at least one connection and the maximum number of connections is the number of neurons in the network. **Figure 3** shows the structure of a derived word that represents the n configured presynaptic connections of a postsynaptic neuron.

Next, the CG for defining presynaptic connections of a neuron is introduced.

The set of terminals $\Sigma_T = \{1, 2, \ldots, 8, 9, \ldots, N-1, N, +, -, :, |\}$ includes all the characters accepted for a valid word; numbers from 1 to N (number of servomotors in the hexapod robot) are used to define the number of connections or the index of a presynaptic neuron (the labeled servomotors are associated to an index for simplicity of the word), numbers from 1 to 9 are used to define synaptic weights, symbols + and − are used for excitatory and inhibitory weights respectively and symbols : and | are auxiliary for parsing the word.

The set of non-terminals $\Sigma_N = \{$
$<$***neuronSynapses***$>$ ($\downarrow g_i$),
$<$***connections***$>$ ($\downarrow g_i$, $\uparrow g_o$),
$<$***neuronIdList***$>$ ($\downarrow g_i$, $\uparrow n$),
$<$***synapses***$>$ ($\downarrow g_i$),
$<$***synapse***$>$ ($\downarrow g_i$, $\uparrow g_o$),
$<$***weight***$>$ ($\downarrow g_i$),
$<$***sign***$>$ ($\downarrow g_i$),
$<$***digit***$>$ ($\downarrow g_i$)
$\}$

The axiom of the grammar $S = <\textbf{\textit{neuronSynapses}}> (\downarrow g_i)$.

The production set $P = \{$
$<\textbf{\textit{neuronSynapses}}> (\downarrow g_i) \models <\textbf{\textit{connections}}> (\downarrow g_i, \uparrow g_o)$:
$<\textbf{\textit{synapses}}> (\downarrow g_o)$ {},
$<\textbf{\textit{connections}}> (\downarrow g_i, \uparrow g_o) \models <\textbf{\textit{neronIdList}}> (\downarrow g_i, \uparrow n)$ {
$\uparrow g_o = \downarrow g_i \cup <$synapses$> (\downarrow g_i) \models <$synapse$> (\downarrow g_i, \uparrow g_{o_1}) | \ldots |$
$<$synapse$> \left(\downarrow g_{o_{\uparrow n-1}}, \uparrow g_{o_{\uparrow n}}\right)$ {}
},
$<\textbf{\textit{neuronIdList}}> (\downarrow g_i, \uparrow \mathbf{1}) \models 1\{\}$,
...
$<\textbf{\textit{neuronIdList}}> (\downarrow g_i, \uparrow N) \models N\{\}$,
$<\textbf{\textit{synapse}}> (\downarrow g_i, \uparrow g_o) \models <\textbf{\textit{neuronIdList}}> (\downarrow g_i, \uparrow n)$,
$<\textbf{\textit{weight}}> (\downarrow g_i)$ {
$\uparrow g_o = \downarrow g_i - \{<$neuronIdList$> (\downarrow g_i, \uparrow n) \models \uparrow n\{\}\}$},
$<\textbf{\textit{weight}}> (\downarrow g_i) \models <\textbf{\textit{sign}}> (\downarrow g_i) <\textbf{\textit{digit}}> (\downarrow g_i)$ {},
$<\textbf{\textit{sign}}> (\downarrow g_i) \models +\{\}$,
$<\textbf{\textit{sign}}> (\downarrow g_i) \models -\{\}$,
$<\textbf{\textit{digit}}> (\downarrow g_i) \models 1\{\}$,
...
$<\textbf{\textit{digit}}> (\downarrow g_i) \models 9\{\}$
}

There is no global information for this CG, thus $K = \emptyset$.

2.3.2. CGE Mapping Process

The CGE Mapping Process is a deterministic method that transforms individuals from their genotype form into their phenotype form; this allows individuals to be evaluated in the problem context through a fitness function. The genotype-to-phenotype mapping process for the CGE is carried out as follows (Ortega et al., 2007):

1. Choose the leftmost non-terminal symbol in the sentential form being processed.
   1.1. Evaluate the attributes.
   1.2. Select the applicable rules from the first attribute in each non-terminal.
2. Number the n right-hand sides of all the rules for this non-terminal symbol (from 0 to $n-1$), where the rules are in an arbitrary order, which should be maintained during the whole process.
3. If $n > 1$, select the right-hand side of the rule whose number equals codon mod (number of right-hand sides for this non-terminal). Else if $n = 1$, then the unique rule is selected, and the codon is not consumed.
4. Derive the next word by replacing the non-terminal with the selected right-hand side.

In Appendix A, there is an example of a derivation tree generated by the CGE mapping process and the proposed CG.

2.3.3. Search Engine: (1 + 1) – Evolution Strategy

The Evolution Strategies (ESs) (Rechenberg, 1973; Schwefel, 1977) are optimization algorithms based on the concept of the evolution of evolution, because biological processes have been optimized by evolution, and evolution is a biological process in itself (Engelbrecht, 2007). This family of algorithms is part of Evolutionary Algorithms (EAs).

In the literature, several variants of ES algorithms have been proposed. They can be generally described by the following notation $(\mu/\rho^{+}_{,}\lambda)$ – ES. The μ is the size of the parent population, ρ is the number of parents in the crossover, λ is the size of the offspring population and $(^{+}_{,})$-selection operators indicate from which population (s) is (are) individuals selected for the next generation of parents; the plus-selection $(+)$ takes into account both populations, while the comma-selection $(,)$ only takes into account the offspring population.

In ES, the candidate solutions of a d-dimensional problem are formed by the object parameter vector y and the endogenous strategy parameters. The type of components of y depend on the problem to solve ($\mathbb{R}$, $\mathbb{N}$, $\mathbb{B}$ or more complex structures are allowed) (Beyer, 2013), and the number and type of strategy parameters can vary according to the design of the candidate solutions; these strategy parameters are used to self adapt the ES and they are not involved on the fitness calculation of individuals.

For this work $(1 + 1)$ – ES is used; this ES has one parent and only one offspring is generated. It uses the plus-selection operator, whereby the best individual from the parent and its offspring is selected to form the parent population in the next generation. The solutions are formed using the vector y and only one strategy parameter σ. **Algorithm 1** shows the implemented $(1 + 1)$ – ES [based on the theory of ES algorithms (Engelbrecht, 2007; Beyer, 2013)].

In **Algorithm 1**, line 1, the vector y is randomly initialized in the search range for each y component (which depends on the problem to solve) and the strategy parameter is set to be $\sigma \sim 3.0$ (Dortmund, 1995). In line 2, $F(\bullet)$ implies the evaluation of an object parameter vector. In line 4, τ is a learning parameter used to update σ, usually calculated from the dimensionality of the problem $\tau = \frac{1}{\sqrt{d}}$ (Engelbrecht, 2007). Finally, $N(0, 1)$ are random numbers normally distributed with mean equals to 0 and a SD of 1 (Talbi, 2009).

2.3.4. SPIKE-Distance-Based Fitness Functions

Since there is no explicit model for fitness computation in the search space of connectivity configuration, we have to explore how to build up one. We have found that an important criteria to achieve such a goal is to explore the aforementioned search space to make a specific neuron replicated an input rhythmic signal. In functional approximation, an alternate and explicit mathematical expression is constructed for the objective function (Jin, 2005), which in this case is unknown.

ALGORITHM 1 | (1+1) – ES.

1: initialize (y, σ)
2: $F_y := F(y)$
3: **repeat**
4: $\quad \tilde{\sigma} := \sigma e^{\tau \mathbf{N(0,1)}}$
5: $\quad \tilde{y} := y + \tilde{\sigma}(\mathbf{N(0,1)}_1, \ldots, \mathbf{N(0,1)}_d)^T$
6: $\quad F_{\tilde{y}} := F(\tilde{y})$
7: $\quad$ **if** $F_{\tilde{y}} < F_y$ **then**
8: $\quad\quad y := \tilde{y}$
9: $\quad\quad \sigma := \tilde{\sigma}$
10: $\quad$ **end if**
11: **until** stop criterion

For this study, two fitness functions were designed as functional approximations to improve in some aspect the topological design of SCPGs; the fitness functions are based on the Bivariate SPIKE-distance $D_S(\bullet,\bullet)$ (Kreuz et al., 2013) (see Appendix B). In general, both fitness functions have the same purpose, to drive the search by measuring the similarity between a target spike train and a generated spike train. But each fitness function takes into account extra criteria, such as structural aspects or number of rhythmic patterns to be replicated. Next, the designed fitness functions are shown, in equation (3), s_t is the target spike train, and s_g is the spike train generated after the simulation of a presynaptic connection design of a neuron. In equation (4), $\{s_{t_i}\}$ is a set of target spike trains, and $\{s_{g_i}\}$ is a set of the spike train generated after the simulation of a presynaptic connection design of a neuron where $i = 1, \ldots, |\{s_{t_i}\}|$ and $|\{s_{t_i}\}| = |\{s_{g_i}\}|$.

The function given by equation (3) is defined with the intention of achieving compact topologies; it takes into account structural information about the normalized number of connections ($\frac{num\_conn}{N}$, num_conn is the number of presynaptic neurons connected to the current neuron and N is the total number of neurons into the SCPG) required to replicate the input information plus the similarity between spike trains to assign a fitness value to an individual.

$$f_1(s_t, s_g) = D_S(s_t, s_g) + \frac{num\_conn}{N} \tag{3}$$

The function given by equation (4) is defined with the intention of making SCPGs capable of generating several rhythmic patterns (as mentioned in Section 1); it takes into account the accumulation of similarities between spike trains of sets of target spike trains and generated spike trains to assign a fitness value to an individual.

$$f_2(\{s_{t_i}\}, \{s_{g_i}\}) = \sum_{i=1}^{|\{s_{t_i}\}|} D_S(s_{t_i}, s_{g_i}) \tag{4}$$

2.4. Hardware

SCPGs have been implemented in hardware (on an FPGA Spartan 6 board) and successfully validated on a real hexapod robot. The FPGA-based implementation is a fully reconfigurable hardware architecture, which runs the different SCPGs in real time. The FPGA controls all the servomotors (neurons) in the hexapod robot in order to perform the gaits shown in **Figure 2**. The hexapod robot configuration implemented is shown in **Figure 4** (Rostro-Gonzalez et al., 2015); each leg is controlled by two servomotors, i.e., the motors for coxa and femur. Thus, the hardware configuration requires twelve neurons to handle all the servomotors on the implementation.

3. RESULTS

For this study, two experiments were carried out, one for each fitness function, to design SCPGs. Next, the parameters for all the experiments are reported.

The configuration parameters are the same for all experiments, unless a particular case is explicitly specified. The configuration was as follows:

- *BMS model*: a normalized BMS neuron was used, and thus the threshold θ, in equation (2), was set at a value of 1. The leak rate γ, in equation (1), was assigned to 0.5 for ease of implementation on hardware.
- $(1 + 1)$ – ES:
  - The dimension of the search space was $d = 75$.
  - The range of each component of the object parametor vector, $y_j \in [0, 255]$ where $j = 1, \ldots, d$. In the mapping process, each component value is rounded to the closest integer.
  - The function calls were the stop criteria of the algorithm; the number of function calls for designing each neuron varies according to the fitness function used. In the experiments of

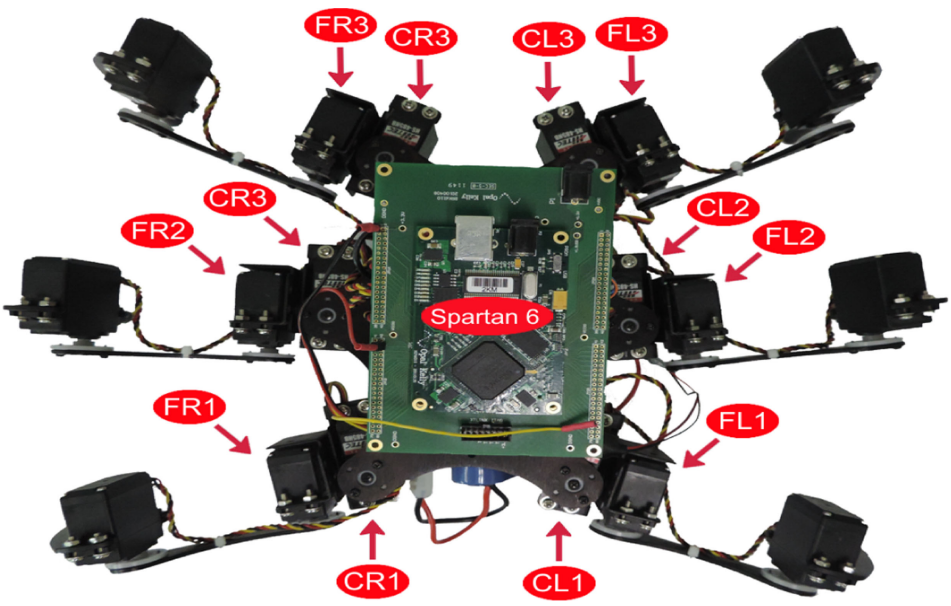


FIGURE 4 | Hexapod robot with FPGA Spartan 6.

the first fitness function, given by equation (3), 50 function calls were used. For the experiment of the second fitness function, given by equation (4), 500 function calls were used.

To validate the design of the SCPGs, numerical tests based on two different fitness functions were carried out, and the results are now presented.

In the first experiment, we considered equation (3) as the fitness function. In this case, we generated an SCPG for each gait; thus the gait transition must be done by changing the connectivity pattern. For this experiment, the expected fitness value on every design is equal to 1; only one presynaptic neuron is expected to stimulate the postsynaptic neuron to reproduce its input signal. The results are given in the following order: walk, jog, and run gaits, respectively. The generated words for presynaptic connectivity and the final topologies resulting from the integration of individuals designs for each gait are reported in **Tables 1–3**. Finally, the weight matrices for each gait are given in equations (15–17) (see Appendix C). As can be observed, the SCPGs were successfully designed by using this fitness function. The number of synaptic is $N = 12$ (the minimum expected), due to that fact that there is a restriction on the number of these during the computation.

TABLE 1 | Configuration of designed SCPG for walking gait by using equation (3).

Neuron (ID)	Presynaptic connectivity	Topology
FL1 (1)	1:2, +4	
CL1 (2)	1:11, +9	
FL2 (3)	1:4, +3	
CL2 (4)	1:5, +1	
FL3 (5)	1:8, +7	
CL3 (6)	1:9, +6	
FR1 (7)	1:6, +1	
CR1 (8)	1:1, +7	
FR2 (9)	1:2, +6	
CR2 (10)	1:3, +3	
FR3 (11)	1:4, +6	
CR3 (12)	1:7, +5	

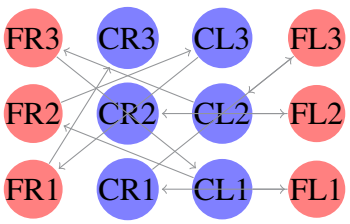
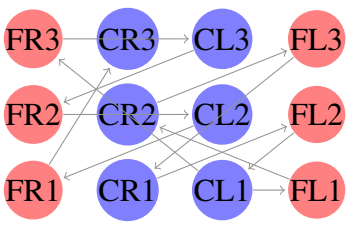


TABLE 2 | Configuration of designed SCPG for jogging gait by using equation (3).

Neuron (ID)	Presynaptic connectivity	Topology
FL1 (1)	1:2, +5	
CL1 (2)	1:3, +5	
FL2 (3)	1:8, +1	
CL2 (4)	1: 9, +7	
FL3 (5)	1:10, +9	
CL3 (6)	1:11, +8	
FR1 (7)	1:4, +2	
CR1 (8)	1:5, +5	
FR2 (9)	1:6, +9	
CR2 (10)	1:1, +8	
FR3 (11)	1:2, +2	
CR3 (12)	1:7, +4	



In the second experiment, we considered equation (4) as the fitness function. Here, we generated a single SCPG for the three gaits. Thus, the gait transition can be performed by altering the driving signal (switching the state of the SNN for the initial state of the desired gait). For this experiment, the expected fitness value for every design is 0. The generated words for presynaptic connectivity and the final topologies generated by the integration of individuals designs are reported in **Table 4**. Finally, the weight matrix is given in equation (18) (see Appendix C). As can be observed, the SCPG was successfully designed by using this fitness function. In terms of hardware design, this method is highly suitable to be provided of sensory information as the driving signal.

Finally, in **Figure 5**, we also present the results of a real-time simulation on the hexapod robot. In this case, we show

TABLE 3 | Configuration of designed SCPG for running gait by using equation (3).

Neuron (ID)	Presynaptic connectivity	Topology
FL1 (1)	1:6, +1	
CL1 (2)	1:7, +9	
FL2 (3)	1:12, +7	
CL2 (4)	1:9, +4	
FL3 (5)	1:6, +5	
CL3 (6)	1:7, +6	
FR1 (7)	1:4, +4	
CR1 (8)	1:9, +5	
FR2 (9)	1:6, +1	
CR2 (10)	1:3, +5	
FR3 (11)	1:8, +9	
CR3 (12)	1:1, +9	

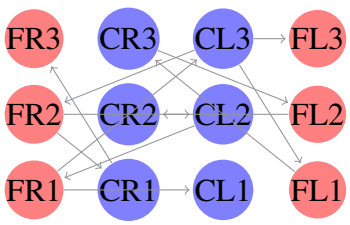
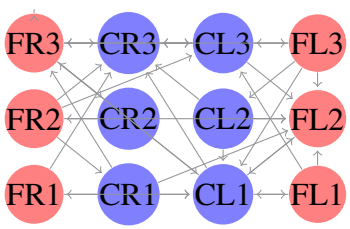


TABLE 4 | Configuration of designed SCPG for all gaits by using equation (4).

Neuron (ID)	Presynaptic connectivity	Topology
FL1 (1)	1:2, +8	
CL1 (2)	7:5, −2|7, +1|8, +2|1, −7|11, +2|3, +4|4, +2	
FL2 (3)	7:6, +1|8, −2|5, +4|9, +4|2, −5|1, −9|4, +8	
CL2 (4)	1:5, +8	
FL3 (5)	1:6, +5	
CL3 (6)	5:9, +3|5, −9|11, +3|12, −1|1, +6	
FR1 (7)	1:8, +5	
CR1 (8)	1:9, +4	
FR2 (9)	1:10, +8	
CR2 (10)	1:11, +9	
FR3 (11)	5:8, −2|2, −4|12, +9|11, +3|6, −3	
CR3 (12)	7:5, +8|2, +1|9, −4|6, −1|4, −3|7, +5|11, −4	



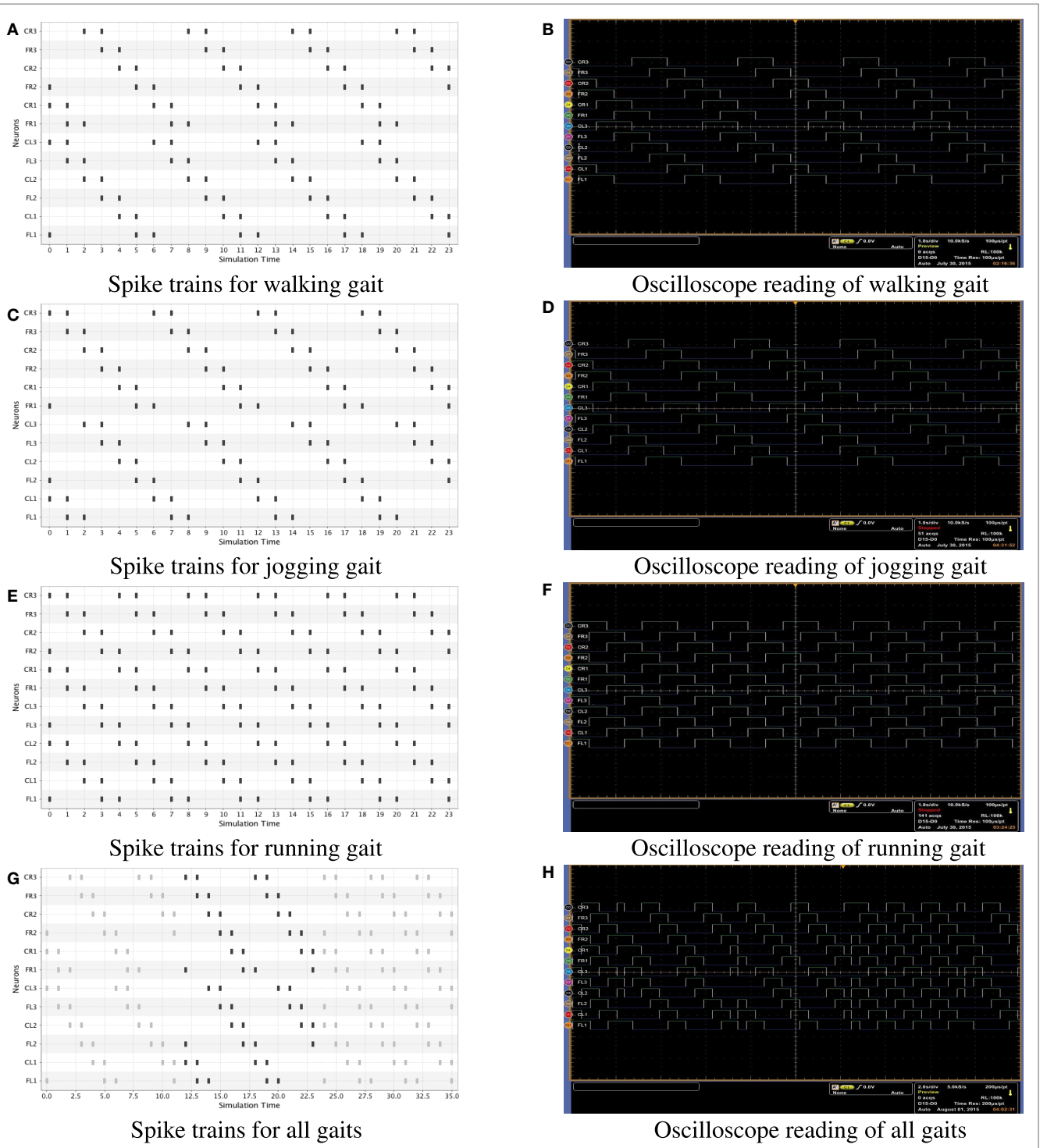


FIGURE 5 | Spike trains generated for all gaits; Figures (A–F) were obtained by equation (3) and Figures (G,H) were obtained by equation (4). In the left side we present numerical simulations in software and on the right side oscilloscope signals in a real time simulation of the hexapod robot. The oscilloscope signals are directly taken from the FPGA. **(A)** Spike trains for walking gait. **(B)** Oscilloscope reading of walking gait. **(C)** Spike trains for jogging gait. **(D)** Oscilloscope reading of jogging gait. **(E)** Spike trains for running gait. **(F)** Oscilloscope reading of running gait. **(G)** Spike trains for all gaits. **(H)** Oscilloscope reading of all gaits.

both the simulation in software (left side) and the oscilloscope signals (right side) generated during the performance of a locomotion pattern (walking, jogging, and running) in the robot. In order to get the real time signals, we used an MSO5204B Mixed Signal Oscilloscope, which has 16 digital channels. We only used 12 of the 18 available servomotors in the robot out

of the 16 digital channels, which was enough to perform the measures.

4. CONCLUSION

In this study, an automatic design methodology involving CPGs built as SNNs for hexapod locomotion has been presented. The proposal follows the divide-and-conquer approach to design the SNNs for given input rhythmic signals. Instead of designing the SNN as a whole, the methodology integrates the individual presynaptic configuration (connectivities and weights) of each locomotor neuron to create the final SNNs. The individual presynaptic configuration is carried out by CGE, which modifies the solutions over the search space of connections and weights. The CGE allows to dispense with a predefined architecture to avoid the explicit learning process for a neuron when replicating a specific rhythmic signal. The aforementioned advantage is possible because the solution indirectly represents the number of presynaptic connections, the indexes of these connections and their respective weights. The quality of the solutions is given by fitness functions that are mainly based on SPIKE-distance.

The proposed methodology has been successfully validated by generating SNNs to replicate rhythmic signals. The two fitness functions used in this work allow us to implement the hexapod gaits transitions in two ways: by changing the pattern connectivity when each gait is produced by one designed SNN and by altering the driving signal when an SNN can produce several rhythmic signals. The designed SCPGs were validated in both computer simulations and hardware implementation running on an FPGA to control an hexapod robot and, all of them showed the desired behaviors. Moreover, this study has reached the results reported in Rostro-Gonzalez et al. (2015) by generating one SCPG for each gait and their implementations, and it has managed to generate SCPGs capable of producing several rhythmic signals taking as basis the proposal of Rostro et al.; this was possible due to the characteristics of the EAs and the SPIKE-distance based fitness function definitions.

Our proposal was used for hexapod gaits and their implementation over hexapod robots; however it can be tested on designing SCPGs for other gaits and different legged robots when target spikes are provided for the optimization process. Other criteria can be also added to the fitness functions based on SPIKE-distance, considering aspects of hardware implementation, for example.

AUTHOR CONTRIBUTIONS

AE has contributed on the conception and design of the work in the grammar design and software implementation phases, has acquired and analyzed the CPG's topologies for hexapod locomotion, and has been involved on the draft process. HR-G has contributed on the conception and design of this work in the SNNs and locomotion gaits phases, has analyzed the resulting CPG topologies, and has worked on the draft process, contributing to a critical revision on the paper's content and given his approval for the results. MC and MO-R have contributed on the conception of this work; they have been involved on the draft process, aiding with their critical revision and giving their final approval. EG-H has contributed on the conception and design of the work in the hexapod robot' configuration and implementation of designed CPG over FPGAs phases, has analyzed the designed topologies over real hexapod robots, and has been involved on the draft process. MS-F has contributed on the conception and design of this work in the design part supporting the optimization phase and has been implicated from draft process, to the critical revisions for improving the content.

ACKNOWLEDGMENTS

This research has been supported by the CONACYT project "Aplicación de la Neurociencia Computacional en el Desarrollo de Sistemas Robóticos Biológicamente Inspirados" (No 269798).

FUNDING

This work has been partially funded by the SEP-PRODEP project (Apoyo a la incorporación de NPTC).

REFERENCES

Arena, P. (2000). The central pattern generator: a paradigm for artificial locomotion. *Soft Comput.* 4, 251–266. doi:10.1007/s005000000051

Barron-Zambrano, J. H., and Torres-Huitzil, C. (2012). *CPG Implementations for Robot Locomotion: Analysis and Design*. Rijeka: INTECH Open Access Publisher.

Beyer, H. (2013). *The Theory of Evolution Strategies*. Berlin, Heidelberg: Springer.

Brown, T. G. (1911). The intrinsic factors in the act of progression in the mammal. *Proc. R Soc. Lond. B Biol. Sci.* 84, 308–319. doi:10.1098/rspb.1911.0077

Brown, T. G. (1914). On the nature of the fundamental activity of the nervous centres; together with an analysis of the conditioning of rhythmic activity in progression, and a theory of the evolution of function in the nervous system. *J. Physiol.* 48, 18. doi:10.1113/jphysiol.1914.sp001646

Buschmann, T., Ewald, A., von Twickel, A., and Büschges, A. (2015). Controlling legs for locomotion – insights from robotics and neurobiology. *Bioinspir. Biomim.* 10, 041001. doi:10.1088/1748-3190/10/4/041001

Chen, W., Ren, G., Zhang, J., and Wang, J. (2012). Smooth transition between different gaits of a hexapod robot via a central pattern generators algorithm. *J. Intell. Robot. Syst.* 67, 255–270. doi:10.1007/s10846-012-9661-1

Christiansen, H. (1985). "Syntax, semantics, and implementation strategies for programming languages with powerful abstraction mechanisms," in *The Eighteenth Hawaii International Conference on System Sciences*, Hawaii.

Crespi, A., and Ijspeert, A. J. (2008). Online optimization of swimming and crawling in an amphibious snake robot. *IEEE Trans. Robot.* 24, 75–87. doi:10.1109/TRO.2008.915426

Dempsey, I., O'Neill, M., and Brabazon, A. (2009). *Foundations in Grammatical Evolution For Dynamic Environments*, Vol. 194. Springer.

Ding, S., Li, H., Su, C., Yu, J., and Jin, F. (2013). Evolutionary artificial neural networks: a review. *Artif. Intell. Rev.* 39, 251–260. doi:10.1007/s10462-011-9270-6

Dortmund, T. (1995). *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. New York, NY: Oxford University Press.

Engelbrecht, A. (2007). *Computational Intelligence: An Introduction*. Chichester: Wiley.

Gerstner, W., and Kistler, W. (2002). *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge: Cambridge University Press.

Grabowska, M., Godlewska, E., Schmidt, J., and Daun-Gruhn, S. (2012). Quadrupedal gaits in hexapod animals – inter-leg coordination in free-walking adult stick insects. *J. Exp. Biol.* 215, 4255–4266. doi:10.1242/jeb.073643

Hong, Y.-D., Park, C.-S., and Kim, J.-H. (2014). Stable bipedal walking with a vertical center-of-mass motion by an evolutionary optimized central pattern generator. *IEEE Trans. Ind. Electron.* 61, 2346–2355. doi:10.1109/TIE.2013.2267691

Ijspeert, A. J. (2008). Central pattern generators for locomotion control in animals and robots: a review. *Neural Netw.* 21, 642–653. doi:10.1016/j.neunet.2008.03.014

Jin, Y. (2005). A comprehensive survey of fitness approximation in evolutionary computation. *Soft Comput.* 9, 3–12. doi:10.1007/s00500-003-0328-5

Judd, J. S. (1990). *Neural Network Design and the Complexity of Learning. A Bradford book*. Cambridge: MIT Press.

Koza, J. R. (1992). *Genetic Programming: on the Programming of Computers by Means of Natural Selection. A Bradford book*, Vol. 1. Cambridge: MIT press.

Kreuz, T., Chicharro, D., Houghton, C., Andrzejak, R. G., and Mormann, F. (2013). Monitoring spike train synchrony. *J. Neurophysiol.* 109, 1457–1472. doi:10.1152/jn.00873.2012

Lewis, M. A., Tenore, F., and Etienne-Cummings, R. (2005). "CPG design using inhibitory networks," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation, 2005. ICRA 2005* (Barcelona: IEEE), 3682–3687.

Maass, W. (1997). Networks of spiking neurons: the third generation of neural network models. *Neural Netw.* 10, 1659–1671. doi:10.1016/S0893-6080(97)00011-7

MacKay-Lyons, M. (2002). Central pattern generation of locomotion: a review of the evidence. *Phys. Ther.* 82, 69–83.

Mulansky, M., Bozanic, N., Sburlea, A., and Kreuz, T. (2015). "A guide to time-resolved and parameter-free measures of spike train synchrony," in *International Conference on Event-based Control, Communication, and Signal Processing (EBCCSP)* (IEEE), 1–8.

Nassour, J., Hénaff, P., Benouezdou, F., and Cheng, G. (2014). Multi-layered multi-pattern CPG for adaptive locomotion of humanoid robots. *Biol. Cybern.* 108, 291–303. doi:10.1007/s00422-014-0592-8

O'Neill, M., and Brabazon, A. (2006a). "Grammatical differential evolution," in *International Conference on Artificial Intelligence (ICAI'06)* (Las Vegas, Nevada: CSEA Press).

O'Neill, M., and Brabazon, A. (2006b). Grammatical swarm: the generation of programs by social programming. *Nat. Comput.* 5, 443–462. doi:10.1007/s11047-006-9007-7

Ortega, A., De La Cruz, M., and Alfonseca, M. (2007). Christiansen grammar evolution: grammatical evolution with semantics. *IEEE Trans. Evol. Comput.* 11, 77–90. doi:10.1109/TEVC.2006.880327

Park, C.-S., Hong, Y.-D., and Kim, J.-H. (2014). Evolutionary-optimized central pattern generator for stable modifiable bipedal walking. *IEEE/ASME Trans. Mechatronics* 19, 1374–1383. doi:10.1109/TMECH.2013.2281193

Patel, L. N. (2009). "Central pattern generators: optimisation and application," in *Nature-Inspired Algorithms for Optimisation*, ed. R. Chiong (Berlin; Heidelberg: Springer-Verlag), 235–260.

Pratt, J., Chew, C.-M., Torres, A., Dilworth, P., and Pratt, G. (2001). Virtual model control: an intuitive approach for bipedal locomotion. *Int. J. Robot. Res.* 20, 129–143. doi:10.1177/02783640122067309

Puntambekar, A. (2008). *Advanced Data Structures and Algorithms*. Pune: Technical Publications.

Rechenberg, I. (1973). *Evolutions Strategie: optimierung technischer Systeme nach Prinzipien der biologischen evolution. Problemata, 15*. Stuttgart: Frommann-Holzboog Verlag.

Rostro-Gonzalez, H., Cerna-Garcia, P., Trejo-Caballero, G., Garcia-Capulin, C., Ibarra-Manzano, M., Avina-Cervantes, J., et al. (2015). A CPG system based on spiking neurons for hexapod robot locomotion. *Neurocomputing* 170, 47–54. doi:10.1016/j.neucom.2015.03.090

Rostro-Gonzalez, H., Cessac, B., and Vieville, T. (2012). Parameter estimation in spiking neural networks: a reverse-engineering approach. *J. Neural Eng.* 9, 026024. doi:10.1088/1741-2560/9/2/026024

Russell, A., Orchard, G., Dong, Y., Mihalas, S., Niebur, E., Tapson, J., et al. (2010). Optimization methods for spiking neurons and networks. *IEEE Trans. Neural Netw.* 21, 1950–1962. doi:10.1109/TNN.2010.2083685

Russell, A., Orchard, G., and Etienne-Cummings, R. (2007). "Configuring of spiking central pattern generator networks for bipedal walking using genetic algorithms," in *IEEE International Symposium on Circuits and Systems, 2007. ISCAS 2007* (New Orleans: IEEE), 1525–1528.

Ryan, C., Collins, J., and O'Neill, M. (1998). "Grammatical evolution: evolving programs for an arbitrary language," in *Proceedings of the First European Workshop on Genetic Programming Lecture Notes in Computer Science 1391*, eds W. Banzhaf, R. Poli, M. Schoenauer, and T. C. Fogarty (Berlin; Heidelberg: Springer-Verlag), 83–96.

Schwefel, H. P. (1977). *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie*, Vol. 1. Basel: Birkhäuser.

Shutt, J. N. (1993). *Recursive Adaptable Grammars*. Master's thesis, Worchester Polytechnic Institute.

Sotelo-Figueroa, M. A., Puga Soberanes, H. J., Carpio, J. M., Fraire Huacuja, H. J., Cruz Reyes, L., and Soria-Alcaraz, J. A. (2014). Improving the bin packing heuristic through grammatical evolution based on swarm intelligence. *Math. Prob. Eng.* 2014, 12. doi:10.1155/2014/545191

Soula, H., Beslon, G., and Mazet, O. (2006). Spontaneous dynamics of asymmetric random recurrent spiking neural networks. *Neural Comput.* 18, 60–79. doi:10.1162/089976606774841567

Talbi, E.-G. (2009). *Metaheuristics: From Design to Implementation*, Vol. 74. Hoboken, NJ: John Wiley & Sons.

Vukobratović, M., and Borovac, B. (2004). Zero-moment point – thirty five years of its life. *Int. J. Humanoid Robot.* 1, 157–173. doi:10.1142/S0219843604000083

Watt, D. A., and Madsen, O. L. (1983). Extended attribute grammars. *Comput. J.* 26, 142–153. doi:10.1093/comjnl/26.2.142

Wu, Q., Liu, C., Zhang, J., and Chen, Q. (2009). Survey of locomotion control of legged robots inspired by biological concept. *Sci. China F* 52, 1715–1729. doi:10.1007/s11432-009-0169-7

Wyffels, F., and Schrauwen, B. (2009). "Design of a central pattern generator using reservoir computing for learning human motion," in *Advanced Technologies for Enhanced Quality of Life, 2009. AT-EQUAL'09* (Iasi: IEEE), 118–122.

Yao, X. (1999). Evolving artificial neural networks. *Proc. IEEE* 87, 1423–1447. doi:10.1109/5.784219

Yu, J., Tan, M., Chen, J., and Zhang, J. (2014). A survey on CPG-inspired control models and system implementation. *IEEE Trans. Neural. Netw. Learn Syst.* 25, 441–456. doi:10.1109/TNNLS.2013.2280596

Conflict of Interest Statement: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

APPENDIX

A. Connectivity Word Derivation Example

This appendix shows an example of the generation of a word that represents the presynaptic connections and their weight configuration through the CGE mapping process and the proposed grammar. The input numeric string is [121, 203, 5, 254, 50, 78, 91, 17, 31], **Figure A1** shows the generated derivation tree for the input numeric string; the sequence of the derivation tree is marked by numerated up/down arrows.

According to the numeration of the arrows, the full process can be described as follows:

1. The axiom of the CGE, $<neuronSynapses>$ $(\downarrow g_i)$ is processed. It only has one production, thus the axiom is replaced by its unique production and no codon is used. The status of the word until now is $<connections>$ $(\downarrow g_i, \uparrow g_o)$: $<synapses>$ $(\downarrow g_o)$.
2. The non-terminal $<connections>$ $(\downarrow g_i, \uparrow g_o)$ is processed, due that is the leftmost non-terminal. As it has only one production, it is replaced by its production. The status of the word until now is $<neuronIdList>$ $(\downarrow g_i, \uparrow n)$: $<synapses>$ $(\downarrow g_o)$.
3. Processing $<neuronIdList>$ $(\downarrow g_i, \uparrow n)$, it is necessary to choose one of the twelve productions. To select the production that replaces the current leftmost non-terminal, we consider the first number of the numeric string to applythe rule $121\ mod\ 12 = 1$; thus the second production is used to replace this non-terminal. The status of the word until now is 2: $<synapses>$ $(\downarrow g_o)$.
4. The non-terminal $<neuronIdList>$ $(\downarrow g_i, \uparrow n)$ synthesizes a numeric value of 2 from the selected production which replaced it. This value will be used in an upper level of the derivation tree.
5. In this step, we apply the semantic rule of the non-terminal $<connections>$ $(\downarrow g_i, \uparrow g_o)$. This rule carries out the synthesis of a new CG called g_o, which contains all the information of the CG g_i plus an unique production to the non-terminal $<synapses>$ $(\downarrow g_o)$. The production contains the elements to generate the presynaptic configuration neuron. The number of elements is indicate by the synthesized number in previous steps.
6. The non-terminal $<synapses>$ $(\downarrow g_o)$ is processed by using the synthesized CG g_o, which already contains a unique production of this non-terminal. The status of the word until now is 2: $<synapse>$ $(\downarrow g_o, \uparrow g_{o1})$ | $<synapse>(\downarrow g_{o_1}, \uparrow g_{o_2})$.
7. Processing $<synapse>(\downarrow g_o, \uparrow g_{o_1})$, it is replaced by its unique production. The status of the word until now is 2: $<neuronIdList>$ $(\downarrow g_o, \uparrow n)$, $<weight>$ $(\downarrow g_o)$ | $<synapse>(\downarrow g_{o_1}, \uparrow g_{o_2})$.
8. The non-terminal $<neuronIdList>$ $(\downarrow g_o, \uparrow n)$ is processed by choosing one of its twelve productions. Applying the rule $203\ mod\ 12 = 11$, its last production is selected. The status of the word until now is 2:12, $<weight>$ $(\downarrow g_o)|<synapse>(\downarrow g_{o_1}, \uparrow g_{o_2})$.
9. The non-terminal $<neuronIdList>$ $(\downarrow g_o, \uparrow n)$ synthesizes a numeric value of 11 from the selected production which replaced it. This value will be used in an upper level of the derivation tree.
10. The leftmost $<weight>$ $(\downarrow g_o)$ is processed. It has only one production, thus it is replaced by its production. The status of the word until now is 2:12, $<sign>$ $(\downarrow g_o)$ $<digit>$ $(\downarrow g_o)$ | $<synapse>(\downarrow g_{o_1}, \uparrow g_{o_2})$.
11. Processing $<sign>$ $(\downarrow g_o)$ based on the rule $5 mod 2 = 1$, its second production is selected to replace it. The status of the word until now is 2:12, $-<digit>$ $(\downarrow g_o)$ | $<synapse>$ $(\downarrow g_{o_1}, \uparrow\ g_{o_2})$.
12. Processing $<digit>$ $(\downarrow g_o)$ by applying the rule $254 mod 9 = 2$, the third production is selected to replace it. The status of the word until now is 2:12, $-3|<synapse>(\downarrow g_{o_1}, \uparrow g_{o_2})$.
13. In this step, it is applied the semantic rule of the non-terminal $<synapses>$ $(\downarrow g_o)$. The rule synthesizes a new CG called g_{o_1}, which contains all the information of the CG g_o less the production of $<neuronIdList>$ $(\downarrow g_o, \uparrow\ n)$ that synthesized the value of 12.
14. Processing $<synapse>(\downarrow g_{o_1}, \uparrow g_{o_2})$ by using the recently sinthetized CG g_{o_1}, it is replaced by its unique production. The status of the word until now is 2:12, $-3|<neuronIdList>$ $(\downarrow g_{o_1}, \uparrow n)$, $<weight>$ $(\downarrow g_{o_1})$.
15. The non-terminal $<neuronIdList>$ $(\downarrow g_{o_1}, \uparrow n)$ is processed by choosing one of its eleven productions. Applying the rule $50 mod 11 = 6$, its seventh production is selected. The status of the word until now is 2:12, $-3|7$, $<weight>$ $(\downarrow g_{o_1})$.
16. The non-terminal $<neuronIdList>$ $(\downarrow g_{o_1}, \uparrow n)$ synthesizes a numeric value of 7 from the selected production which replaced it. This value will be used in an upper level of the derivation tree.
17. The leftmost $<weight>$ $(\downarrow g_{o_1})$ is processed. It has only one production, thus it is replaced by its production. The status of the word until now is 2:12, $-3|7$, $<sign>$ $(\downarrow g_{o_1})<digit>$ $(\downarrow g_{o_1}))$.
18. Processing $<sign>$ $(\downarrow g_{o_1})$ based on the rule $78 mod 2 = 0$, its first production is selected to replace it. The status of the word until now is 2:12, $-3|7, +<digit>$ $(\downarrow g_{o_1}))$.
19. Processing $<digit>$ $(\downarrow g_{o_1})$ by applying the rule $91 mod 9 = 1$, the second production is selected to replace it. The status of the word until now is 2:12, $-3|7, +2$.
20. In this step, it is applied the semantic rule of the non-terminal $<synapses>$ $(\downarrow g_{o_1})$. The rule synthesized a new CG called g_{o_2}, which contains all the information of the CG g_{o_1} less the production of $<neuronIdList>$ $(\downarrow g_o, \uparrow n)$ that synthesized the value of 7. But the derivation ends here and the rest of codons are not used.

B. Bivariate SPIKE-Distance

The SPIKE-Distance (Kreuz et al., 2013) is a parameter-free and timescale-adaptive measure for estimating the degree of synchrony between spike trains. In general, the distance is defined as a temporal average of the spike trains' dissimilarity profiles. There are several variants of SPIKE-distance, for this study the Bivariate

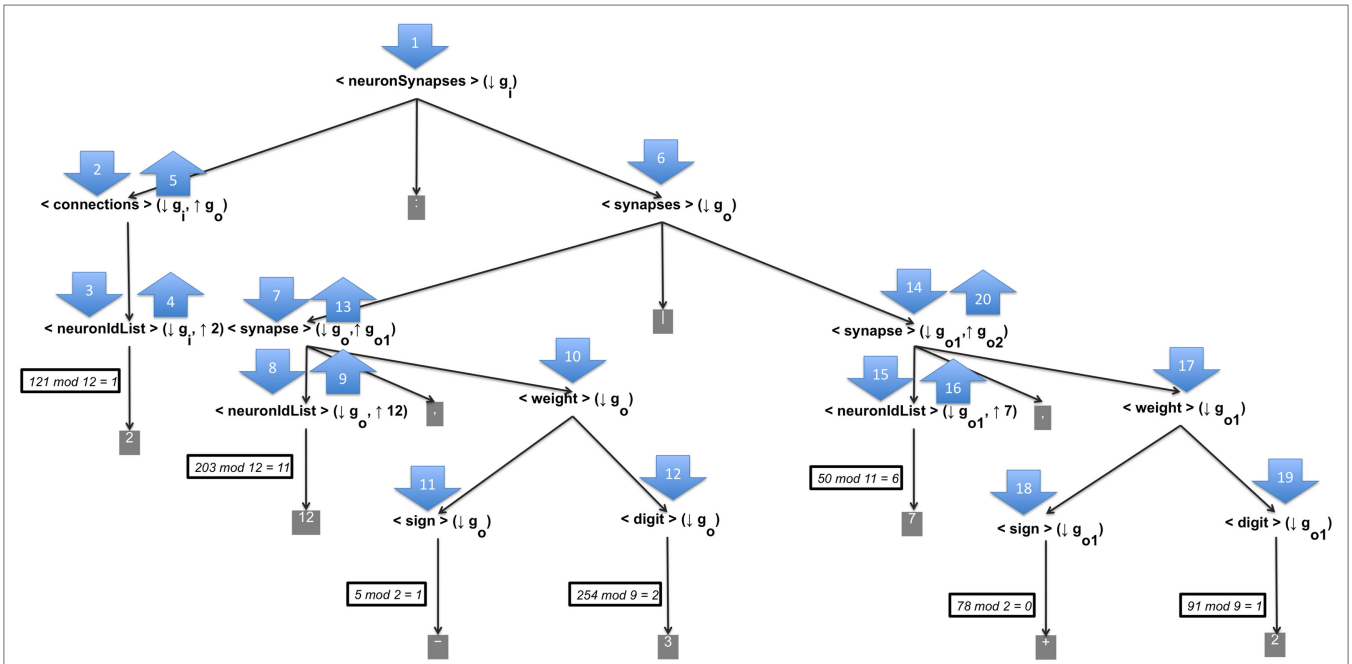


FIGURE A1 | Example of a derivation tree by the CGE's mapping process. For the input numeric string [121, 203, 5, 254, 50, 78, 91, 17, 31] the CGE's mapping procees generates the word 2:12, −3|7, +2.

SPIKE-distance is used. The Bivariate SPIKE-distance is defined by equation (A1).

$$D_S(s_1, s_2) = \frac{1}{T}\int_{t=0}^{T} S(t)dt \tag{A1}$$

where T is the overall length of the spike trains; for this study, it is 24 times (from time 0 to 23) for each spike train in all experiments. And $S(t)$ is the dissimilarity profile between two spike trains. Before starting the dissimilarity profile calculations, the spike trains must be formatted as follows:

1. To add auxiliary leading spikes at time $t = 0$ and auxiliary trailing spikes at time $t = T$ to each spike train; this is to avoid ambiguities in some calculations.
2. To label the spike times in the spike trains $n = 1$, 2 by t_i^n, $i = 1, \ldots, M_n$; the maximum number of spike times of the nth spike train is given by M_n.

The computation of $S(t)$ is based on four corner spikes surrounding the current time t (Mulansky et al., 2015), the preceding spikes $t_P^{(n)}(t)$ and the following spikes $t_F^{(n)}(t)$ where $n = 1, 2$; they are computedby using equations (A2) and (A3), respectively.

$$t_P^{(n)}(t) = max\left(t_i^{(n)} | t_i^{(n)} \leq t\right) \tag{A2}$$

$$t_F^{(n)}(t) = min\left(t_i^{(n)} | t_i^{(n)} > t\right) \tag{A3}$$

The inter-spike interval $x_{ISI}^{(n)}(t)$, is given by equation (A4).

$$x_{ISI}^{(n)}(t) = t_F^{(n)}(t) - t_P^{(n)}(t) \tag{A4}$$

For each corner spikes, the distance to the closest spike of the other spike train is computed. Equations (A5) and (A6) show the distances for the spike train $n = 1$, this is similarly made with $\Delta t_P^{(2)}(t)$ and $\Delta t_F^{(2)}(t)$ for the spike train $n = 2$.

$$\Delta t_P^{(1)}(t) = min\left(|t_P^{(1)}(t) - t_i^{(2)}|\right) \tag{A5}$$

$$\Delta t_F^{(1)}(t) = min\left(|t_F^{(1)}(t) - t_i^{(2)}|\right) \tag{A6}$$

Next, the distance of the corner spikes to the current time is calculated by equations (A7) and (A8).

$$x_P^{(n)}(t) = t - t_P^{(n)}(t) \tag{A7}$$

$$x_F^{(n)}(t) = t_F^{(n)}(t) - t \tag{A8}$$

The weighted distance of each spike train $n = 1$, 2 is given by equation (A9).

$$S_n(t) = \frac{\Delta t_P^{(n)}(t) x_F^{(n)}(t) + \Delta t_F^{(n)}(t) x_P^{(n)}(t)}{x_{ISI}^{(n)}(t)} \tag{A9}$$

Finally, the disimilarity profile $S(t)$ is calculated by equation (A10).

$$S(t) = \frac{S_1(t) x_{ISI}^{(2)}(t) + S_2(t) x_{ISI}^{(1)}(t)}{2\left(x_{ISI}^{(1)}(t) + x_{ISI}^{(2)}(t)\right)^2} \tag{A10}$$

The Bivariate SPIKE-distance is not a metric as such, but it has useful features such as: it is bound to $0 \leq D_S \leq 1$, if $s_1 = s_2$ then $D_S(s_1, s_2) = 0$ and the distance between s_1 and s_2 is symmetric $D_S(s_1, s_2) = D_S(s_2, s_1)$.

C. Weight Matrixes

$$W_w = \begin{vmatrix} 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 9 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 7 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 5 & 0 & 0 & 0 & 0 & 0 \end{vmatrix} \quad \text{(A11)}$$

$$W_r = \begin{vmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 9 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 7 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 6 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 9 & 0 & 0 & 0 & 0 \\ 9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{vmatrix} \quad \text{(A13)}$$

$$W_j = \begin{vmatrix} 0 & 5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 7 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 9 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 8 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 9 & 0 & 0 & 0 & 0 & 0 & 0 \\ 8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 \end{vmatrix} \quad \text{(A12)}$$

$$Waio = \begin{vmatrix} 0 & 8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -7 & 0 & 4 & 2 & -2 & 0 & 1 & 2 & 0 & 0 & 2 & 0 \\ -9 & -5 & 0 & 8 & 4 & 1 & 0 & -2 & 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 6 & 0 & 0 & 0 & -9 & 0 & 0 & 0 & 3 & 0 & 3 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 8 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 9 & 0 \\ 0 & -4 & 0 & 0 & 0 & -3 & 0 & -2 & 0 & 0 & 3 & 9 \\ 0 & 1 & 0 & -3 & 8 & -1 & 5 & 0 & -4 & 0 & -4 & 0 \end{vmatrix} \quad \text{(A14)}$$