Received: 13 September 2025 | Revised: 11 November 2025 | Accepted: 28 November 2025 | Published online: 12 January 2026

RESEARCH ARTICLE

Journal of Climbing and Walking Robots
2026, Vol. 00(00) 1–9
DOI: 10.47852/bonviewJCWR62027653

BON VIEW PUBLISHING

Testing the Effects of Evolved Central Pattern Generators in an Articulated Robot Animat Ecosystem

Christian Hahm[1,*] and Rohan Jay[2]

[1] *Department of Computer and Information Sciences, Temple University, USA*

[2] *Department of Computer Science, Johns Hopkins University, USA*

Abstract: This study added central pattern generators (CPGs) to 3D artificial animal (i.e., animat) articulated robots contained in a simulated ecosystem. Specifically, Hopf and Matsuoka CPG equations were added to the hidden neuron calculations, and the effects of these augmentations on robot behavior were analyzed. The results indicated that robots utilizing sum-and-squash neurons with the CPG equations in the hidden neuron equation performed better than robots using only sum-and-squash neurons without CPGs. The rhythmic motor patterns generated by the CPG dynamics helped the articulated robots to move around the environment, as well as to better accomplish survival and reproduction. The robots equipped with CPGs were able to reproduce more often and with higher energy efficiency than the No CPG robots. The performance advantage in birthrates of robots with CPGs over robots without CPGs increased as the simulation progressed. Overall, the addition of CPGs to the neurons of 3D articulated robots provided useful rhythmic movement that proved beneficial for movement, survival, and reproduction.

Keywords: central pattern generator (CPG), simulated ecosystem, articulated robot animat, locomotion efficiency

1. Introduction

Biological neural networks and animal brains contain certain neural circuits and neurons called central pattern generators (CPGs). CPGs are capable of producing rhythmic outputs endogenously, i.e., without rhythmic sensory input, although in some cases they work with and can be modulated by sensory input [1].

This automatic generation of rhythmic patterns internally is useful to animals because it allows their body to move in rhythmic ways, which helps them in navigating the environment. For example, the rhythmic CPG signals can be routed to the motor neurons to produce regular movements which propel the body such as walking gaits, flapping wings, and slithering or swimming motions.

Consequently, CPGs are also a topic of interest in robotics, where it is desired for the robot to move around so it can accomplish tasks. The more complex the robot, the more complex it can be to control. For example, in a quadruped robot, each leg can have up to 3 degrees of freedom (DOF). A control program for the quadruped will have to control at least 3 DoF * 4 legs = 12 degrees of freedom. The more degrees of freedom it has to control, the larger the search space of possible movements, not only allowing more freedom of movement but also potentially increasing the time it takes to reach a valid solution. CPGs allow for a reduction in the dimensionality of the input, allowing the network to simply modulate the outputting smooth rhythmic signals to the muscles/joints, rather than trying to control every individual muscle/joint with high-precision low-level control. By explicitly adding CPGs to robot control, the complexity of control is reduced, creating a smaller optimization space, therefore accelerating optimization time.

This study tested the effects of adding evolving CPGs to articulated robots in an animat ecosystem. This study used the open-source ERAIASON animat ecosystem [2].

For optimization, evolutionary algorithms were used over gradient-based algorithms for a few reasons. Firstly, evolutionary algorithms (EAs) are highly general; it is easy to add CPGs, or any other parameterized system, into the evolutionary process for optimization. Secondly, evolutionary algorithms are on average better at global exploration than gradient-based algorithms [3], because updates in gradient-based algorithms are more localized to the current best solution, whereas gradient-free algorithms have the opportunity to escape local optima.

2. Literature Review

Marder and Bucher [4] described two mechanisms of CPGs, pacemaker neurons and reciprocal inhibition. The function of both is to control multiple muscles in a repetitive manner using only one chemical signal. In pacemaker neurons, the neurons are connected so they fire together; that way, a signal that triggers one neuron triggers all of the connected neurons. With reciprocal inhibition, you have two sets of neurons A and B which are coupled. When neurons in A are activated, they suppress/inhibit B. Some biological process eventually allows B to trigger, which suppresses A. Similarly, A can eventually become active again, triggering B. This cycle of activation and inhibition can result in a rhythmicity.

Lu et al. [5] described a system of differential equations that incorporated time delays and the strength of synaptic interfaces over time. To make sure that their system was still oscillatory and a closed loop, they used Grunwald–Letnikov fractional derivatives. They then experimented with different fractional orders of their derivatives.

Buono and Golubitsky [6] described a model for controlling quadruped motion using CPGs. The model was described geometrically

***Corresponding author:** Christian Hahm, Department of Computer and Information Sciences, Temple University, USA. Email: christian.hahm@temple.edu

as a cube-like structure, where the four corners of each face represent the four legs and their CPGs, and the top and bottom faces represent two phases of quadruped locomotion. The four CPGs were explicitly coupled. They show that this model can result in various gaits such as pacing, trotting, and walking. Two limitations to this model are that it cannot result in jumping; and that each leg only has one CPG available to it.

Bellegarda and Ijspeert [7] use deep reinforcement learning to train a sim-to-real quadruped robot by learning the parameters of CPG neurons. The authors trained it in simulation and tested it on a Unitree A1 quadruped, and found that it was robust to disturbances even not experienced during training, such as uneven terrain and added mass.

Ijspeert [8] used a genetic algorithm to evolve a CPG system and applied it to a salamander-like physical robot to observe its gait. Their model used a set of differential equations that describe the interactions between the neurons in the CPG. Each neuron had its own bias and time constant, and had weights for its connections to other neurons. A "chromosome" of a neuron described the bias, time constant, and weights. The genetic algorithm optimized the chromosome for each neuron.

Bongard [9] allowed neural networks to evolve controllable "CPG neurons", which output a sinusoidal signal. The frequency of a CPG neuron's signal was able to be controlled by incoming neural signals, such that positive inputs caused a higher frequency, whereas negative inputs caused a lower frequency. The phase was set relative to the time step when the neuron was formed during neurogenesis.

Wu et al. [10] implemented a CPG controller for the stable locomotion of biped robots. They designed Hopf-style oscillators whose outputs were transformed into motor outputs via a multivariate linear mapping model, rather than neural networks. They showed the ability of their simulated biped robots to walk on flat surfaces and uphill.

While the last few studies mentioned used EAs to evolve CPGs, they didn't include evolutionary forces such as food dependency, meaning that it was not as imperative for the robots to move in multiple directions. Their fitness functions also take a more localized approach; for example, Ijspeert [8] considers the number of extrema in the CPG oscillation, as well as the phase difference.

This paper studies evolution in virtual articulated quadrupeds in the context of an ecosystem, where quadrupeds must move towards virtual food blocks and must actively stop at them to eat in order to survive. The food blocks are scattered throughout the environment, meaning that the quadrupeds must be able to change direction. This study compares quadrupeds with evolving CPGs to those without, in order to see if the CPGs still made a significant improvement to their locomotion ability.

The rest of our paper is organized as follows. Section 3.1 describes the quadruped and its kinematics. Section 3.2 describes the CPG models used. Section 3.3 describes the sensors and motors of the quadrupeds, and Section 3.4 describes the training environment. Section 3.5 describes the EA used. Section 4 describes our results using both graphs and statistics, and Section 5 describes possible future work.

3. Research Methodology

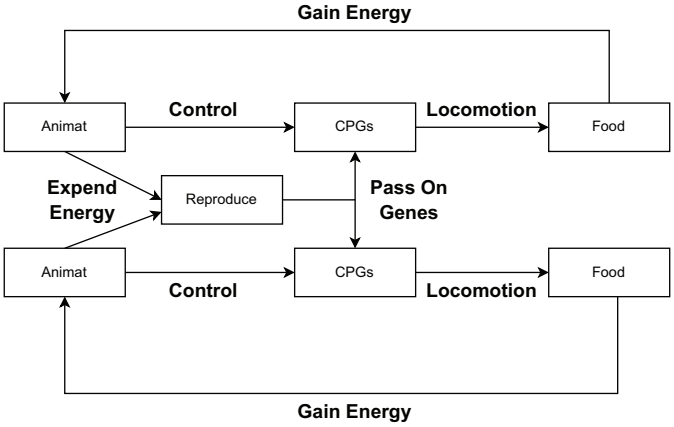
3.1. Experimental overview

In this section, we will discuss our experimental design.

A new animat was born with 25 energy, which is 1/4 the amount contained in a fully grown food block. Consequently, to reproduce, the parent(s) had to contribute 25 energy to the offspring. Energy drained a rate of 0.05 energy per 0.02 seconds, or 2.5 energy per second. This meant that a robot had about 10 seconds to begin eating food if it was to survive. A robot was not permitted to live for more than 90 seconds, regardless of its energy levels.

Figure 1 shows a block diagram of our experimental design. Algorithm 1 shows the high-level algorithm. Neural networks were evolved to control fixed-morphology articulated animal ("animat") robots which lived together in a shared ecosystem. Each animat had to consume energy to stay alive, while using CPGs in their hidden neurons to control the robotic limbs to locomote towards new energy sources. The population persisted either by autonomous reproduction, sexually or asexually, which costed energy. Or, when autonomous reproduction was insufficient to maintain a minimum level of animats, the population was supplemented via evolutionary algorithm. The goal of the simulation experiment was to evolve robots to become better capable of survival and reproduction.

Figure 1
The high-level flow of energy and control in the ecosystem



Algorithm 1
High-level simulation loop

```
while simulation is running do
  if population.count < population_threshold then
    Generate new animat from evolutionary algorithm
  end if

  for animat in population do
    dead ← (energy < 0) ∨ (health < 0) ∨ (age > MAX_AGE)

    if dead then
      remove animat from simulation and population
      compute novelty score
      probabilistically add to novelty search archive
      compute fitness score
      add to reproduction pools
    else
      Update robot sensors for animat
      Compute next state of neural network for animat
      Update robot motors for animat
    end if
  end for
end while
```

3.2. Robot

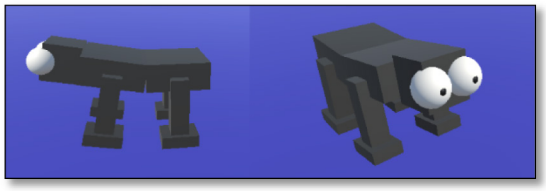
The robots were hand-designed into a quadruped form, such that the robot had 2 torso segments, 4 legs each consisting of a "quad", "calf", and "foot" segment (see Figure 2) and a "head" segment on which the vision sensor was attached (see Figure 3). The robot's joints were ball joints, which could rotate on X, Y, and Z axes between [−30, 30] degrees.

The simulations had a time step of 0.02 seconds.

3.3. Neural network and CPGs

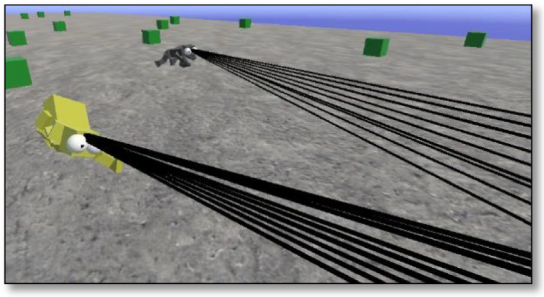
The neural network of a robot was updated every 0.04 seconds. The initial neuroevolution of augmenting topologies (NEAT) genomes (neural networks) were not completely disconnected. Instead, the 27

Figure 2
Quadruped robot used in simulations



Note: The "eyeballs" are decorative, simply used to indicate the position of the visual sensor.

Figure 3
Vision simulation via a circular raycast array



Note: Vision was simulated using a circular arrangement of raycasts emanating from the robot's "head" segment.

vision neurons were sparsely connected to the motor outputs, where each potential connection had a 30% chance to form.

For our CPG models, versions of the Hopf and Matsuoka oscillator differential equations were implemented in our hidden neuron calculations. The coefficients of these equations were evolved as evolutionary parameters in each individual neuron. This allowed evolution to decide the characteristics of the sinusoidal signal produced by the equations. Furthermore, the input signal $I(t)$ into the hidden neuron at time t, coming "externally" from other neurons, was allowed to influence the CPG output as detailed in the sections below.

The output of the CPG oscillation $I_{CPG}(t)$ was scaled by an evolved gain parameter G, then mixed with the raw input signal $I(t)$ using a per-neuron evolved parameter blend ∈ [0, 1] for use in a weighted average, as follows:

$$I_{net}(t) = blend \times I(t) + (1 - blend) \times G \times I_{CPG} \tag{1}$$

Then, $I_{net}(t)$ was squashed through a non-linear activation function, hypertangent, to get the final hidden neuron output O ∈ (−1, 1):

$$O(t) = tanh(I_{net}(t)) \tag{2}$$

The high-level process can be seen in Algorithm 2.

3.3.1. Hopf

One of our CPG models implemented a Hopf adaptive oscillator [11] in the hidden neurons. The original Hopf equations are:

$$\frac{d}{dt}x(t) = \gamma\left(\mu - r(t)^2\right)x(t) - \omega(t)y(t) + \epsilon F(t) + \tau sin(\theta(t) - \phi(t)) \tag{3}$$

$$\frac{d}{dt}y = \gamma\left(\mu - r(t)^2\right)y(t) + \omega(t)x(t) \tag{4}$$

$$\frac{d}{dt}\omega(t) = -\epsilon F(t)\frac{y(t)}{r(t)} \tag{5}$$

$$\frac{d}{dt}\alpha(t) = \eta x(t)F(t) \tag{6}$$

$$\frac{d}{dt}\phi(t) = sin\left(\frac{\omega(t)}{\omega_0}\theta_0 - \theta(t) - \phi(t)\right) \tag{7}$$

Algorithm 2
Steps for updating neuron activation with CPGs at timestep *t*

```
Require: blend ∈ [0, 1] is an evolved neuron parameter for blending
CPG values with neuron input
Require: G is an evolved neuron parameter for modulating CPG
influence

for neuron in network do
  I ← 0
  for synapse in incoming_connections do
    w ← synapse.weight
    incoming_neuron ← synapse.from
    a ← incoming_neuron.activation_{t−1}
    I ← I + w * a
  end for
  I ← I + bias

  if neuron.type is hidden then
    I_CPG ← compute relevant CPG
    I_net ← blend × I + (1 − blend) × G × I_CPG
    neuron.activation_t ← tanh(I_net)
  else
    neuron.activation_t ← tanh(I)
  end if
end for
```

where, for a given hidden neuron CPG, ω is the frequency of the oscillation, α is the amplitude of the oscillation, θ is the oscillator phase, and ϕ is the phase difference between oscillator i and a reference oscillator.

μ is a parameter that modulates the oscillation amplitude, γ is a parameter which controls the speed of recovery after perturbation, $F(t)$ is an external learning signal, and τ and ϵ are coupling constants.

Since this experiment did not implement explicit neuron coupling or a reference oscillator, $\phi(t) = 0$, therefore Equation (7) was ignored and set $\phi = 0$ in Equation (3).

This experiment did not utilize an external learning signal, so $F(t) = 0$, causing Equations (5) and (6) to be ignored, and the term $\epsilon F(t)$ modulating x in Equation (3) to disappear. Adaptation and external driving of the oscillatory signal by other neurons were still permitted, by adding a new term $u(t)$ to Equation (3), scaled by an evolved parameter ϵ similarly to how $F(t)$ was scaled. The term $u(t)$ represents the incoming signal into the neuron from other neurons. It was calculated using the original input $I(t)$ to the hidden neuron, normalized and clamped to [−1, 1] using an evolved normalization parameter max_input ∈ [0.1,20], such that $u(t) = \frac{I(t)}{max\_input}$ ∈ [−1,1].

Since Equation (7) is zero, ω never changes, so the function $\omega(t)$ in Equation (5) becomes a constant ω (which here, was evolved).

One is now left with two equations for a simplified Cartesian Hopf oscillator in hidden neuron i:

$$\frac{d}{dt}x(t) = \gamma\left(\mu - r(t)^2\right)x(t) - \omega y(t) + \epsilon u(t) + \tau sin(\theta(t)) \tag{8}$$

$$\frac{d}{dt}y(t) = \gamma\left(\mu - r(t)^2\right)y(t) + \omega x(t) \tag{9}$$

One can convert these from Cartesian form (x,y) to polar form (r, θ), allowing the external signal $u(t)$ to directly specify the amplitude and frequency. Thus, it is possible solve for $\frac{d}{dt}$ $r(t)$and $\frac{d}{dt}$ $\theta(t)$.

Starting with the standard polar substitution:

$$r(t)^2 = x(t)^2 + y(t)^2 \tag{10}$$

$$r(t) = \sqrt{x(t)^2 + y(t)^2} \tag{11}$$

$$\begin{aligned}\frac{d}{dt}r(t) &= \frac{x(t)\frac{d}{dt}x(t)+y(t)\frac{d}{dt}y(t)}{\sqrt{x(t)^2+y(t)^2}} = \frac{1}{r(t)}\left[x(t)\frac{d}{dt}x(t) + y(t)\frac{d}{dt}y(t)\right] \\ &= \frac{1}{r(t)}\left[x(t)\left(\gamma\left(\mu - r(t)^2\right)x(t) - \omega y(t) + \epsilon u(t) + \tau sin(\theta(t))\right)\right. \\ &\left. + y(t)\left(\gamma\left(\mu - r(t)^2\right)y(t) + \omega x(t)\right)\right]\end{aligned} \tag{12}$$

$$\begin{aligned}\frac{d}{dt}r(t) = \frac{1}{r(t)}\left[\gamma x(t)^2\left(\mu - r(t)^2\right) + x(t)\tau sin(\theta(t))\right. \\ \left. -\omega x(t)y(t) + \omega x(t)y(t) + \epsilon x(t)u(t) + \gamma y(t)^2\left(\mu - r(t)^2\right)\right]\end{aligned} \tag{13}$$

$$\begin{aligned}\frac{d}{dt}r(t) = \frac{1}{r(t)}\left[\gamma\left(\mu - r(t)^2\right)\left(x(t)^2 + y(t)^2\right)\right. \\ \left. +\epsilon x(t)u(t) + x(t)\tau sin(\theta(t))\right]\end{aligned} \tag{14}$$

$$\frac{d}{dt}r(t) = \frac{1}{r(t)}\left[\gamma\left(\mu - r(t)^2\right)r(t)^2 + \epsilon x(t)u(t) + x(t)\tau sin(\theta(t))\right] \tag{15}$$

$$\frac{d}{dt}r(t) = \gamma\left(\mu - r(t)^2\right)r(t) + \frac{x(t)}{r(t)}(\epsilon u(t) + \tau sin(\theta(t))) \tag{16}$$

Using $x/r = cos(\theta)$, the final differential equation used to update the oscillator's amplitude r was:

$$\frac{d}{dt}r(t) = \gamma\left(\mu - r(t)^2\right)r(t) + \cos(\theta)(\epsilon u(t) + \tau sin(\theta(t))) \tag{17}$$

For $\frac{d}{dt}\ \theta(t)$, starting with the standard polar substitution:

$$\theta(t) = tan^{-1}\left(\frac{y(t)}{x(t)}\right) \tag{18}$$

$$\frac{d}{dt}\theta(t) = \frac{x(t)\frac{d}{dt}y(t) - y(t)\frac{d}{dt}x(t)}{x(t)^2+y(t)^2} \tag{19}$$

$$\frac{d}{dt}\theta(t) = \frac{1}{x(t)^2+y(t)^2}\left[x(t)\frac{d}{dt}y(t) - y(t)\frac{d}{dt}x(t)\right] \tag{20}$$

$$\begin{aligned}\frac{d}{dt}\theta(t) = \frac{1}{x(t)^2+y(t)^2}\left[\gamma\left(\mu - r(t)^2\right)y(t)x(t) + \omega x(t)^2\right. \\ \left. -\gamma\left(\mu - r(t)^2\right)y(t)x(t) + \omega y(t)^2 - \epsilon u(t)y(t) - \tau sin(\theta(t))y(t)\right]\end{aligned} \tag{21}$$

$$\frac{d}{dt}\theta(t) = \frac{1}{x(t)^2+y(t)^2}\left[\omega\left(x(t)^2 + y(t)^2\right) - \epsilon u(t)y(t) - \tau sin(\theta(t))y(t)\right] \tag{22}$$

$$\frac{d}{dt}\theta(t) = \omega - \epsilon\frac{y(t)}{r(t)^2}u(t) - \tau\frac{y(t)}{r(t)^2}sin(\theta(t)) \tag{23}$$

$$\frac{d}{dt}\theta(t) = \omega - \frac{y(t)}{r(t)}\left(\frac{\epsilon}{r(t)}u(t) + \frac{\tau}{r(t)}sin(\theta(t))\right) \tag{24}$$

Using $y/r = sin(\theta)$, the final differential equation used to update the oscillator's phase θ was:

$$\frac{d}{dt}\theta(t) = \omega - \frac{sin(\theta)}{r(t)}(\epsilon u(t) + \tau sin(\theta(t))) \tag{25}$$

Equipped with r and θ, the Hopf output was ultimately calculated as $I_{CPG}(t) = r(t)sin(\theta + \theta_0)$.

The parameter ranges which were allowed are listed in Table 1.

Table 1
The allowed parameter ranges for evolved Hopf parameters

Parameter name	Parameter range
μ	[0.1, 1]
γ	[0.1,3]
ω	[0.1,2]
τ	[0,2]
ϵ	[0,2]
θ_0	[0,2π]
p	[-0.5,0.5]
G	[-1,1]

3.3.2. Matsuoka

For the Matsuoka simulations, each hidden neuron implemented a Matsuoka-style adaptive neuron model, based on the equations from Matsuoka [12]. Rather than hardcoding mutually inhibitory pairs in Matsuoka's study, the network topology was permitted to evolve in this experiment. In this way, a setup of inhibiting pairs could appear if necessary, while also allowing the possibility for other useful connectivity patterns to emerge.

The original Matsuoka equations were written as:

$$\tau\frac{d}{dt}x(t) + x(t) = c - ay_j(t) - bv(t) \tag{26}$$

$$T\frac{d}{dt}v(t) + v(t) = y(t) \tag{27}$$

$$y(t) = max(0, x(t)) \tag{28}$$

where, for a hidden neuron CPG at time t, $x(t)$ is its membrane potential, $y(t)$ is its firing rate, and $v(t)$ is its adaptation/fatigue property. a, b, c are its configurable coefficients, and τ and T are its configurable time constants, which in these simulations were evolved for each individual neuron.

The Matsuoka output $I_{CPG}(t)$ was calculated at each time step to let it contribute towards the hidden neuron's activation. The equation to update a neuron's Matsuoka model at each time step can be determined by simply rearranging the above equations:

$$\frac{d}{dt}x(t) = \frac{c - ay_j(t) - bv(t) - x(t)}{\tau} \tag{29}$$

$$\frac{d}{dt}v(t) = \frac{y(t) - v(t)}{T} \tag{30}$$

Since there was no hardcoded single neuron coupling, the value of the coupled neuron $y_j(t)$ was simulated by using the sum of all external weighted inputs to the neuron. For use in the CPG equations, the external input signal I(t) to the hidden neuron at time t was normalized by dividing it by some evolved parameter $max\_input \in [0.1,20]$ then clamping the result $y_j(t) = \frac{I(t)}{max\_input} \in [-1, 1]$.

The parameter ranges which were allowed are listed in Table 2.

Table 2
The allowed parameter ranges for evolved Matsuoka parameters

Parameter name	Parameter range
T	[0.3,2]
τ	[0.1,1]
a	[-5,5]
b	[1.5,5]
c	[-1,1]
G	[0.1,2]

3.4. Sensorimotor

In order to navigate the environment, the animats had sensor input to collect data. The sensors provided were as follows:

1) 6 touch sensors on each segment (1 per face)
2) 4 rotation sensors on each segment (corresponding to the canonical quaternion)
3) 1 energy sensor
4) 1 health sensor
5) 9 visual sensor raycasts, each with 3 sensor neurons (corresponding to the 3 possible items to detect: floor, food, and robot)
6) 1 mouth sensor (detecting if food is being successfully eaten)
7) 1 sinewave sensor (inputting a predefined sinewave)
8) 1 pain sensor (detecting if the robot is being attacked)

The motors were as follows:

1) 3 motor neurons per joint; 1 for each rotation axis
2) 1 asexual reproduction ("clone") motor neuron
3) 1 sexual reproduction ("mate") motor neuron
4) 1 fight motor neuron (to attack other animats, to decrease their health

3.5. Environment

The environment was a flat plane covered in 150 food blocks and contained a minimum of 50 animats at any given time. When fully consumed, a new food block is spawned. A new food block started from containing 1 unit of energy, and gradually "grew" to contain more energy at a rate of 0.01 energy every 0.02 seconds (or 0.5 food per second), up to a maximum of 100 energy.

3.6. Evolutionary algorithm

3.6.1. Genetic encoding

For genetic encoding, the direct encoding NEAT was used [13], which starts with a minimal network, represented by a list of neurons and connections, and complexifies the network over the evolutionary process by incrementally augmenting the topology with additional neurons and connections.

Our minimal networks sparsely connected random vision neurons to random motor neurons with 30% chance to form each connection. Each network was additionally seeded with 10 random connections and 3 hidden neurons. During mutation, there was a 20% chance to add a new connection, and a 10% chance to add a new node.

For connection mutation: every connection weight had a 90% chance to be perturbed by a small value selected from a Gaussian distribution centered at zero with standard deviation 0.2, otherwise it was replaced with a random value in range [−5, 5]. Each enabled connection had a 1% chance to become disabled, and each disabled connection had a 0.2% chance to become re-enabled.

For node mutation: each bias had a 50% chance to be mutated, in which case it had a 90% chance to be perturbed by a small value selected from a Gaussian distribution centered at zero with standard deviation 0.2, otherwise it was replaced with a random value in range [−5, 5]. There was a 50% chance to mutate every CPG parameter in the node. For a CPG parameter with range $[R_{min}, R_{max}]$, there was a 97% chance to be perturbed by a small value selected from a Gaussian distribution centered at zero with standard deviation $0.1*(R_{max} - R_{min})$, otherwise it was replaced with a random value in range $[R_{min}, R_{max}]$.

For recombination: firstly, one parent was chosen (50/50 chance) from which to select disjoint genes (aka the "disjoint parent"). All sensorimotor neurons were copied from that parent. If both parents contained a connection with the same ID, the connection values were copied from one parent with 50/50 chance. If the connection was enabled in one parent and disabled in the other, there was a 75% to inherit it as disabled, otherwise it was enabled. If one or both nodes connected by that connection did not yet exist in the offspring genome, then the node from that parent was copied as well. If the connection ID was disjoint, belonging to only one parent, then it was copied only if it belonged to the "disjoint parent".

3.6.2. Evolutionary search

The explicit evolutionary search was a combination of objective-based (fitness-based) search, novelty search [14], roulette-wheel selection, and tournament selection.

Rather than evaluating solutions one cohort/generation at a time as in traditional evolutionary search, since animats could die at any moment rather than after a fixed evaluation period, instead an "on-line" or "steady state" genetic algorithm was utilized which maintains a constant-number population of solutions/animats all contained in the same shared environment. The population of animats would die if they 1) expended all their energy, 2) reached a maximum age (here, 90 seconds), and 3) had their health reduced to zero by another animat attacking. Animats could interact with objects (e.g., eat food and mate/fight another animat) only if they detected the object in their vision sensor at a short proximity range of 2 meters or less.

When the population dropped below some minimum number popmin (here popmin = 50), the EA created a new animat via mutation or recombination by pulling one or two "parents" from pools of stored animats. There were 3 pools or "tables" which could be taken from, containing the genomes of dead animats: Fitness Hall of Fame table (containing the fittest animats in the simulation run's history, ranked by fitness), Novelty Hall of Fame table (containing the most novel known animats at the current moment, ranked by novelty), and Fitness Recent Population table (containing the most recent population of animats as first-in-first-out [FIFO], stored with their fitness). To select an animat for reproduction, first one of the three random tables was selected. Then, according to a 50/50 random chance, either standard roulette-wheel (probabilistic) selection or tournament selection (k = 7) was performed on that table.

Animats could also reproduce autonomously, as long as they had enough energy to gift to the offspring. Asexual reproduction, with simple genome cloning and mutation, could be performed by the animat at any time, costing 1/4 of a full-sized food block (25 energy, since each food block contains 100 energy at maximum size). Sexual reproduction, with genome recombination, required 2 animats, 1 looking at the other and both expressing their "mate" neuron, but only costing half the energy (since half was gifted from each parent).

An animat generated by another animat was spawned near its parent(s). A new animat generated by the EA, on the other hand, was spawned in a random location in the environment, at least 8 meters away from any food block.

Our objective-based fitness function, to compute fitness score F for an animat, was as follows:

$$F = (d + d * l + l * f + f) * (1 + R) \tag{31}$$

where $d \in [0, 1]$ ("displacement score") was the displacement of the animat from its birthplace at the time of its death, $l \in [0,1]$ ("look score") was the proportion of time that the animat spent looking at food (closer proximity increased this score), $f \in [0, \infty)$ ("food score") was the amount of food blocks the animat ate during its lifetime, and $R \in [0, \infty)$ ("reproduction score") was the number of times the animat reproduced in its lifetime.

Our novelty function, to compute the behavioral difference D between 2 animat behaviors i and j at a given moment in time was:

$$D = |food\_seen_i - food\_seen_j| + \|pos_j - pos_i\| \tag{32}$$

where *food_seen* is the maximum value of a visual sensor that detects food during that time of its life, and pos is a 2D vector that represents the offset XZ of the animat from its birthplace during that time of its life. These behavioral difference scores are summed and averaged (using the shorter lifespan of the 2 animats). Conceptually, this equation says "two animats are different if one animat was looking at food at a different distance than the other animat, or if one animat moved to a different relative location than the other animat". This was meant to cover the full spectrum of useful behaviors: moving all around the map, and being near/far from food while looking at it. Combined, these factors enable a behavior where the animats walk towards food and eat it in various ways, which simultaneously results in high fitness scores *F*.

To compute the novelty of an animat, the average behavioral distance was found between it and the k-nearest-neighbor animats in a "novelty archive" data structure. Every animat had a 10% chance to be added to the novelty archive when it died.

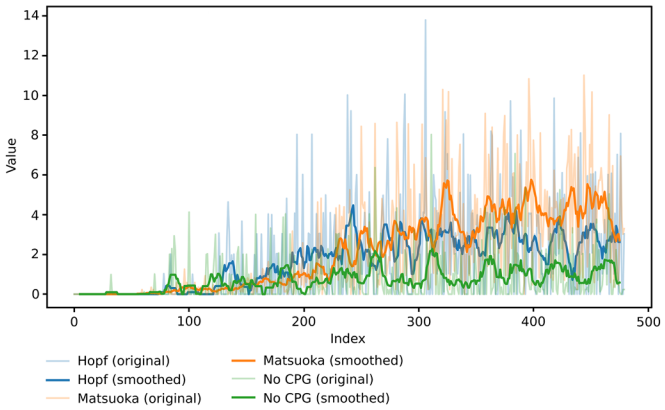
4. Results

Simulations were ran under each condition (No CPG, Hopf, and Matsuoka) for 6 trials each (for 480 * 8 seconds ≈ 1.06 hours per trial), then computed the average across trials for analysis.

4.1. Graphical analysis

Visual inspection of the smoothed birthrate data (Figures 4 and 5) shows that both CPG methods, Hopf and Matsuoka, allowed the robots to perform better than the robots without CPG augmentation. The area under the curve (Figure 4) shows that the area under the curve was ≈ 400 for No CPG ≈ 800 for Hopf, and ≈ 1000 for Matsuoka.

Figure 4
Birthrate per 1000 robots per 8 seconds



Note: Performance comparison for "births per 1000 robots per 8 seconds", original curve and smoothed curve (window = 10).

On the raw metrics, the CPG robots usually outperformed the No CPG robots. The best Hopf and Matsuoka robots were able to achieve higher fitness scores (Figure 6) by the end of the simulation (No CPG: ≈ 90, Hopf: ≈ 150, Matsuoka: ≈ 300). The CPG methods also allowed the best robots to reproduce in higher quantities (Figure 7) by the end of the simulation (No CPG: ≈ 3.5 births/lifetime, Hopf: ≈ 4.75 births/lifetime, Matsuoka: ≈ 7 births/lifetime).

Surprisingly, despite the greater ability of Hopf to reproduce compared to No CPG, the best Hopf robots actually ate less food (Figure 8) than the No CPG method (No CPG: ≈ 250 food/lifetime, Hopf: ≈ 175 food/lifetime, Matsuoka: ≈280 food/lifetime). This result is hard to explain from just the data. Possible explanations may be that Hopf robots had better control over when to reproduce using CPGs, and so they were able to more efficiently trigger reproductions than the No CPG robots, who instead had to rely on fixed neuron bias or sensory stimulation to activate their reproduction. In the prior case, fixed neuron bias, the robot either reproduces always and may drain its energy too fast, or never reproduces. In the latter case, where specific sensory stimulation is required to reproduce, the robot may never or only rarely achieve the proper stimulation. These are only speculative explanations for the somewhat counterintuitive result.

Figure 5
Total area under curve (AUC) for birthrate

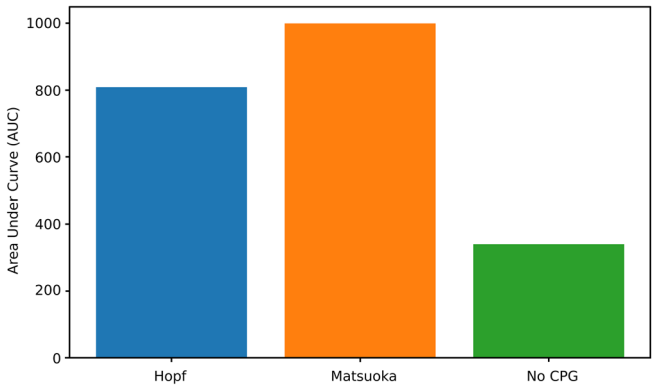


Figure 6
Fitness score in fitness hall of fame

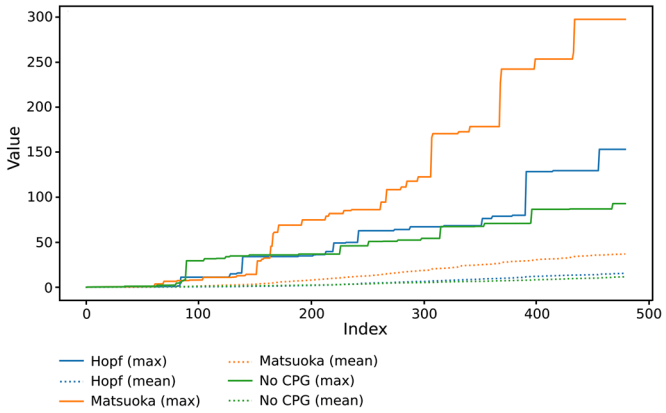
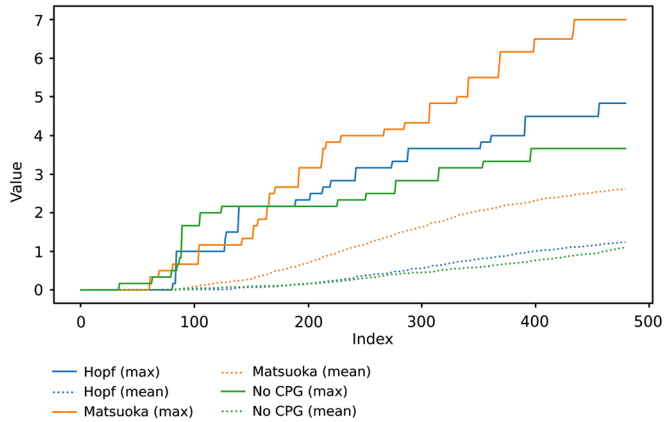


Figure 7
Times reproduced in fitness hall of fame



4.2. Statistical significance tests

To assess whether the use of CPGs significantly improved robot birthrates in our simulations, the raw birthrate data was recorded (see

Figure 8
Food eaten in fitness hall of fame

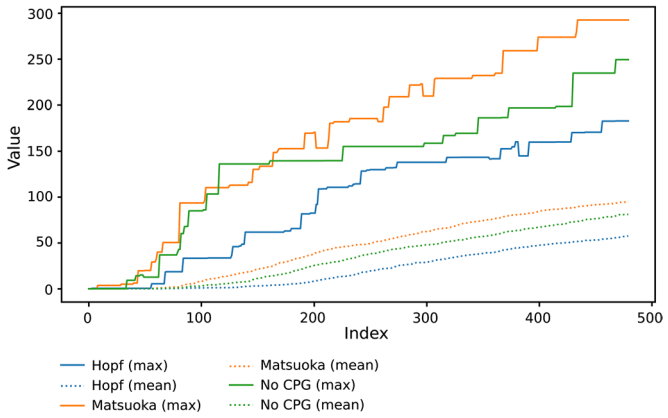
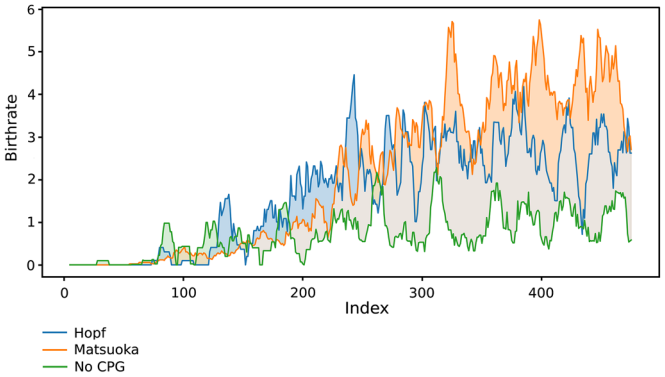


Figure 4) and analyzed with a significance level of 0.05. Wilcoxon signed-rank tests (using SCIPY Python package) were conducted to compare the birthrates between the CPG and No CPG conditions. The results revealed a statistically significant advantage for both the Hopf-based method ($W = 38\,826.5$, $p = 1.14\times10^{-16}$) and the Matsuoka-based method ($W = 62\,659.0$, $p = 3.79 \times 10^{-29}$), indicating that CPGs consistently outperformed the non-CPG baseline across time, allowing robot populations to achieve higher birthrates.

Next was to see whether the benefit on birthrate remained constant, decreased, or increased over the course of simulation. To do this, the statistical analysis was performed on the difference between the CPG and No CPG methods (see Figure 9), with a significance level of 0.05.

Figure 9
Birthrates difference across all three methods



Note: Smoothed, window=10.

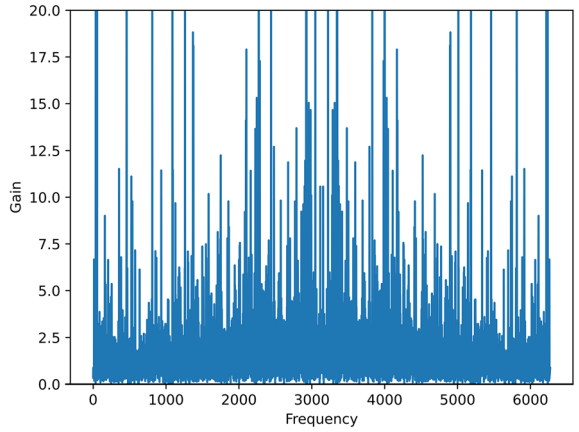
The Kwiatkowski–Phillips–Schmidt–Shin (KPSS) test was performed on the differences between birthrate data using the KPSS function in Python's STATSMODELS library. The KPSS was run on the time series differences between the birthrate using CPGs and the birthrate not using CPGs. The regression parameter was set to "ct", meaning that it tested if the time series was stationary around a trend.[1]

For Hopf vs. No CPG, the KPSS p-value calculated was around 0.0206. The same was done for the Matsuoka vs. No CPG birthrate data. The function was able to find that the actual p-value was less than 0.01; however, it was not able to calculate the exact p-value due to internal interpolation errors inherent to the function. However, the test statistic was 0.248858, higher than the critical value of 0.146. Therefore, in both cases, the difference between the CPG vs. No CPG time series is nonstationary. That is to say, the magnitude of the CPG benefit changed over time.

To determine in what direction the magnitude of benefit was changing, Mann-Kendall trend tests were performed (using PYMANNKENDALL Python package [15]) and linear regression (using SCIPY Python package[2]) on the birthrate differences between the methods. The tests determined the gap between the CPG and non-CPG methods increased over time. For Hopf vs. No CPG, both Mann-Kendall trend analysis (trend = increasing, slope = 0.0027, $p = 4.73 \times 10^{-11}$) and linear regression (slope = 0.0051, $p = 1.24 \times 10^{-10}$, $R2 = 0.083$) determined that the gap was increasing over time. Similarly, for Matsuoka vs. No CPG, both Mann-Kendall trend analysis (trend=increasing, slope = 0.0072, p = 0.00) and linear regression (slope = 0.0096, $p = 9.81 \times 10^{-36}$, $R2 = 0.278$) determined the gap was increasing over time.

The authors also generated the gain portion of a Bode plot in order to see how robust the animats were to changes in the location of the closest food (after two hours of evolution). This was done by treating the input signal as a sum of delta functions, where a delta function was added each time the simulation spawned a new food source which was closer to the animat than the previously spawned food source. The output signal was how much the animat's distance changes based on how much it was able to compensate for the change. The authors took the Fourier transform on the input and output signals in order to break both signals down into their individual frequencies and to see what the gain was for each frequency. The resulting Bode plot is shown in Figure 10. There is a frequency range where the gain is relatively higher, indicating that within this range, animats were less able to quickly adapt to the location change.

Figure 10
Bode plot gain showing response to food frequency changes



Note: The gain part of the Bode plot, which shows the animats' response to difference frequencies at which the location of the closest food changes (e.g. if a new food source spawns close by).

In conclusion, these analyses demonstrate that the CPG-based controllers outperformed the baseline of No CPGs, and the performance difference between the populations also grew over time.

5. Future Directions

In the future, it would be interesting to test CPGs in other frameworks.

For example, how do CPGs affect real-time learning rules such as Hebbian rules, which learn according to neural activation patterns? Since the neural activation patterns become more rhythmic with CPGs,

[1] Statsmodels. (2025). *Statsmodels* (Version 0.15.0) [Software]. https://www.statsmodels.org/dev/generated/statsmodels.tsa.stattools.kpss.html

[2] SciPy. (2025). *SciPy* (Version 1.16.2) [Software]. https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.linregress.html

will the Hebbian plasticity still be meaningful and useful, or instead overshadowed by the rhythmic pattern? This is perhaps one of the most promising next steps, since it is relatively simple to add and combines multiple "realistic"/naturalistic aspects of neuron function.

Soft voxel robots (SVRs) [16] are an alternative to articulated robots and have different properties, namely that they are squishy/soft and made of voxels. It has been shown that soft voxel robots are viable for animat simulations such as performed here [2, 17]. Consequently, it would also be interesting to test these CPG equations on soft voxel robots for the food-seeking and reproduction tasks, since SVRs are known to benefit from rhythmic actuation.

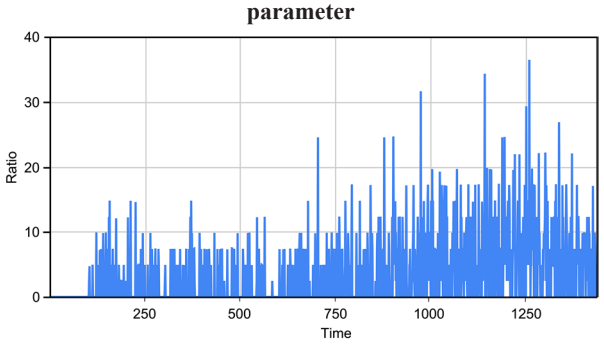
One can also consider using CPGs to augment the abilities of symbolic reasoning systems, such as non-axiomatic reasoning system (NARS) [18, 19]. Though NARS, being general-purpose, may technically be able to handle low-level sensorimotor processes (e.g., NARS can process pixel-level vision activations using logical inference), it is a difficult task for the system that consumes many of the system's resources. The full-scale and fluid control of low-level processes using symbolic reasoning systems is still a difficult research problem. Since CPGs produce rhythmic motor (low-level) movements or patterns that can be triggered by higher-level control processes, CPGs could be useful in symbolic and neurosymbolic AI. For example, NARS making high-level decisions and plans could use CPGs to trigger specific low-level motor movement patterns at specific times, depending on the results of its logical reasoning. This would free the higher-level control and reasoning processes from needing to manage all low-level motor intricacies, allowing NARS more freedom to focus on higher-level processes like planning and goal management, at a higher level of abstraction and generalization. Such CPGs or sinewaves could be integrated into the genetic language of a reasoning system, which could give rise to different species with different motor patterns, and allow NARS (or other logical reasoning systems) to control complex robots more easily.

A preliminary experiment was run with an addition to the reward function, which was changed to multiply the look score by a factor called "norm control", to encourage keeping robots upright; the norm control is first derived from:

$$normcontrol = \sqrt{tilt_x^2 + tilt_y^2} \tag{33}$$

Then $normcontrol$ is given an upper bound u, where $u = K * speed$, where K is derived from a linear quadratic regulator (LQR) control, and $A = \frac{9.8}{FrameRate}$, $B = 1$, $Q = 0.5$, $R = 0.5$. In our experiment, the algorithm provided in the study of Chu et al. [20] was used to solve the discrete-time Riccati equation in the LQR. The LQR control for this upper bound penalized tipping less since if normcontrol is unbounded, the animats would stand still and upright to prevent risk of tipping over. The birthrate results of the preliminary test are graphed in Figure 11.

Figure 11
The birthrate results of a single run with the normcontrol parameter



6. Conclusion

In conclusion, the robots performed better overall with a CPG model attached to their hidden neurons than without a CPG model. Both our Hopf and Matsuoka CPG models provided a similar performance benefit as each other, with the Matsuoka doing better than the Hopf in our trial (in terms of birthrates).

Qualitatively, the robots with CPGs moved more rhythmically, reliably, and smoothly than the robots without CPGs. Yet, the CPG robots were still capable of motor control, such as being able to "switch off" their sinusoidal movements when necessary, such as to stop and fully devour a food block.

Quantitatively, the CPG robots outperformed No CPG robots in the birth and fitness metrics. The Hopf robots had greater reproductive efficiency, able to produce more offspring than No CPG robots while eating less food. Our statistical analysis discovered that the birthrate advantage conferred by CPGs on the animat populations was not constant, but increased over the course of the simulation.

Overall, the data showing improvement in robot survival and reproduction due to CPGs lends credence to the implementation of CPG equations in articulated robots as a potential performance improvement, even for tasks more complicated than locomotion, requiring control such as targeted locomotion (object-seeking).

Acknowledgement

The free version of ChatGPT was used to help with researching and understanding CPGs, performing statistical analysis, and developing the code. The authors manually verified all AI-generated code for correctness.

Ethical Statement

This study does not contain any studies with human or animal subjects performed by any of the authors.

Conflicts of Interest

C.H. is a maintainer of the ERAIASON simulator used in this study, which is distributed free and open-source on GitHub. A paid executable version is also offered as an option. This potential financial interest is disclosed; no other competing interests are declared.

Data Availability Statement

The data that support the findings of this study are openly available in Zenodo at https://doi.org/10.5281/zenodo.17110785.

Author Contribution Statement

Christian Hahm: Conceptualization, Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Resources, Data curation, Writing – original draft, Writing – review & editing, Visualization, Supervision, Project administration. **Rohan Jay:** Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Resources, Data curation, Writing – original draft, Writing – review & editing, Visualization.

References

[1] Ryu, H. X., & Kuo, A. D. (2021). An optimality principle for locomotor central pattern generators. *Scientific Reports*, *11*(1), 13140. https://doi.org/10.1038/s41598-021-91714-1

[2] Hahm, C. (2025). *Experiments in advancing the evolution of artificial animals*. Temple University. https://cis.temple.edu/tagit/publications/TAGIT-TR-24.pdf

[3] Mustafaoglu, Z. S., Pingali, K., & Miikkulainen, R. (2025). Evolutionary policy optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 735–738. https://doi.org/10.1145/3712255.3726711

[4] Marder, E., & Bucher, D. (2001). Central pattern generators and the control of rhythmic movements. *Current Biology*, *11*(23), R986–R996. https://doi.org/10.1016/S0960-9822(01)00581-4

[5] Lu, Q., Wang, X., & Tian, J. (2022). A new biological central pattern generator model and its relationship with the motor units. *Cognitive Neurodynamics*, *16*(1), 135–147. https://doi.org/10.1007/s11571-021-09710-0

[6] Buono, P. L., & Golubitsky, M. (2001). Models of central pattern generators for quadruped locomotion I. Primary gaits: I. Primary gaits. *Journal of Mathematical Biology*, *42*(4), 291–326. https://doi.org/10.1007/s002850000058

[7] Bellegarda, G., & Ijspeert, A. (2022). CPG-RL: Learning central pattern generators for quadruped locomotion. *IEEE Robotics and Automation Letters*, *7*(4), 12547–12554. https://doi.org/10.1109/LRA.2022.3218167

[8] Ijspeert, A. J. (2001). A connectionist central pattern generator for the aquatic and terrestrial gaits of a simulated salamander. *Biological Cybernetics*, *84*(5), 331–348. https://doi.org/10.1007/s004220000211

[9] Bongard, J. (2002). Evolving modular genetic regulatory networks. In *Proceedings of the 2002 Congress on Evolutionary Computation*, 1872–1877. https://doi.org/10.1109/CEC.2002.1004528

[10] Wu, Y., Tang, B., Tang, J., Qiao, S., Pang, X., & Guo, L. (2024). Stable walking of a biped robot controlled by central pattern generator using multivariate linear mapping. *Biomimetics*, *9*(10), 626. https://doi.org/10.3390/biomimetics9100626

[11] Righetti, L., & Ijspeert, A. J. (2006). Programmable central pattern generators: An application to biped locomotion control. In *Proceedings 2006 IEEE International Conference on Robotics and Automation*, 1585–1590. https://doi.org/10.1109/ROBOT.2006.1641933

[12] Matsuoka, K. (2011). Analysis of a neural oscillator. *Biological Cybernetics*, *104*(4), 297–304. https://doi.org/10.1007/s00422-011-0432-z

[13] Stanley, K. O., & Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary Computation*, *10*(2), 99–127. https://doi.org/10.1162/106365602320169811

[14] Lehman, J., & Stanley, K. O. (2011). Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, *19*(2), 189–223. https://doi.org/10.1162/EVCO_a_00025

[15] Hussain, S., & Mahmud, I. (2025). *pymannkendall* (Version 1.4.3) [Software]. https://pypi.org/project/pymannkendall/

[16] Hahm, C. (2025). Neuroevolution for vision-based seeking behavior in 3D soft voxel robots. *Artificial Life and Robotics*, *30*(3), 502–511. https://doi.org/10.1007/s10015-025-01019-z

[17] Meng, H., Zhang, S., Zhang, W., & Ren, Y. (2024). Optimizing actual PID control for walking quadruped soft robots using genetic algorithms. *Scientific Reports*, *14*(1), 25946. https://doi.org/10.1038/s41598-024-77100-7

[18] Wang, P. (1995). *Non-Axiomatic reasoning system: Exploring the essence of intelligence*. Master's Thesis, Indiana University.

[19] Wang, P. (2013). *Non-axiomatic logic: A model of intelligent reasoning*. Singapore: World Scientific. https://doi.org/10.1142/8665

[20] Chu, E. K.-W., Fan, H.-Y., Lin, W.-W., & Wang, C.-S. (2004). Structure-preserving algorithms for periodic discrete-time algebraic Riccati equations. *International Journal of Control*, *77*(8), 767–788. https://doi.org/10.1080/00207170410001714988

How to Cite: Hahm, C., & Jay, R. (2026). Testing the Effects of Evolved Central Pattern Generators in an Articulated Robot Animat Ecosystem. *Journal of Climbing and Walking Robots*. https://doi.org/10.47852/bonviewJCWR62027653