Neuro-Evolutionary learning for CPG-RBFN based legged locomotion

Ashutosh Gupta[1*], Manuel Agraz[1*], Ayan Robinson[1]

***Abstract*—Legged robots are suitable for complex environments but can be challenging to control, especially when dealing with unpredictable terrains. Current state-of-the-art model-free approaches suffer from slow learning due to requiring millions of interactions with the environment. The model-free CPG-RBFN controller network alleviates this by leveraging the capabilities of a Central Pattern Generator (CPG) to induce a locomotion gait and a simple to optimize Radial Basis Function Network (RBFN) to shape the CPG signal. However, recent implementations of CPG-RBFN controllers use Probability-Based Black Box Optimization (PI^{BB}) for training, but this method struggles to achieve good performance when trained for different input signals. We introduce an Evolutionary algorithm to further extend the controller's functionalities, specifically addressing concerns related to noise resilience and adaptability to varied input signals. Notably, empirical evaluations were conducted in the simulated OpenAI Gym's Half Cheetah environment to demonstrate the efficacy of the proposed CPG-RBFN controller with neuroevolution. Our results show that the CPG-RBFN controller's capacity to learn a walking behavior amid noisy conditions and diverse input in an open loop without any system feedback, thereby showcasing its potential applicability in real-world legged robotics scenarios. This research contributes a step forward in developing adaptive control strategies for legged robots, emphasizing the practical utility of the proposed approach in challenging and dynamic environments.**

***Index Terms*—Central Pattern Generator (CPG), Radial basis function network (RBFN), Evolutionary Algorithm (EA), Neuroevolution**

I. Introduction

The field of legged robotics holds the potential to shift how robots interact in our environments. Their ability to conquer challenging terrains, unlike wheeled robots [1], will allow them to move in human environments without infrastructure changes. However, legged robot control remains a challenging endeavor.

Contemporary approaches to legged robot control fall into two main categories: model-based and model-free strategies. Model-based methods leverage analytical or learned models, making assumptions about system dynamics and the environment [2]. In contrast, model-free approaches, often employ Reinforcement Learning (RL) to achieve legged locomotion without explicit models. RL approaches regularly require millions or billions of interactions with the environment to learn a suitable policy, which has led researchers to integrate RL and a Central Pattern Generator (CPG) to achieve faster convergence [3]–[5].

* Authors of Equal Contribution
[1] Collaborative Robotics and Intelligent Systems (CoRIS) Institute, Oregon State University, Corvallis, OR 97331, USA.
{guptaash, agrazvam, robinsay}@oregonstate.edu

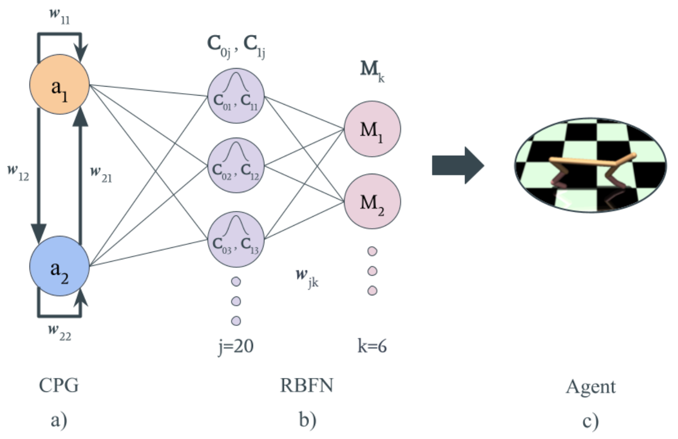


Fig. 1: CPG-RBF Network Architecture. **a)** Central Pattern Generator (CPG) serves as the input to our network. It produces a sinusoidal wave that induces a locomotion gait into the model. **b)** Radial Basis Function Network (RBFN) shapes the CPG input wave into motor outputs. **c)** The motor outputs from the RBFN drive the agent's joints to produce a walking behavior.

RL approaches often use neural networks to represent a policy. Neural networks for locomotion control are usually large and complex [2]. These large complex networks can display state-of-the-art locomotion control but suffer from slow learning. Additionally, these complex models make it hard to analyze and can obfuscate the meaning of the learned policy. To remedy this, researchers introduced a CPG-RBFN controller, demonstrating its adaptability for complex behaviors in six-legged configurations, while providing fast learning and intuition into the learned controller [6]. Their model can be broken down into two parts: the Central Pattern Generator (CPG) induces a locomotion gait into the model, and the Radial Basis Function Network shapes the CPG input signal to produce locomotion for the selected legged configuration. The main advantages of this model are: its generic nature allows it to be applied to different-legged robot morphologies; its reduced number of parameters leads to faster learning; and it does not rely on sensory feedback to generate locomotion. Finally, in [7] the output layer weights are learned through the PI^{BB} algorithm defined by [8].

When controlling a legged robot, a varying frequency in our locomotion signal allows our robot to increase or decrease its speed. In a real robot, electromagnetic noise can interfere with our locomotion signal when our CPG input signal comes from a hardware chip [9]. With this in mind, the previously mentioned combination of CPG-RBFN architecture and PI^{BB} algorithm, has one problem: its performance is severely reduced when training for varying CPG signals and noise. This performance reduction occurs due to PI^{BB} requiring fixed RBFN parameters. By fixing the RBFN parameters the CPG-RBFN model achieves faster learning but sacrifices being able to capture the complexity of different CPG signals. In [7] they accepted this trade-off because their focus was on fast learning of different behaviors (body posture control and obstacle avoidance) thus they could learn them by using a single CPG configuration.

Our research builds upon the CPG-RBFN controller to increase its robustness to noise and changes in the input signal. We employed an Evolutionary Algorithm (EA) to learn the parameters of the RBFN enabling it to learn a walking behavior on different CPG configurations. We chose an EA because it allows us to perform a more extensive search of the solution space without the need for a domain-specific initialization of parameters. Our approach was tested using OpenAI Gymnasium's Half Cheetah environment. The results show that our controller can learn a walking behavior while having a noisy signal and different input configurations.

The rest of the paper is organized as follows. section II talks about related work in CPG, model-based, model-free approaches, and evolutionary learning controls. section III describes our methodology with the network architecture and the learning algorithm. section IV discusses the experimental setup, the subsequent results, and the analysis from our learning approach. section V explains the conclusions and inferences we draw from this study and the future work we plan to expand towards.

II. Background

1) Central Pattern Generators: A CPG model generates periodic rhythmic signals that can realize the robot movements. For legged robots, walking can be accomplished by periodic alternating back-and-forth motion of the legs. CPG-based legged locomotion methods seek to utilize the ability of the pattern generator to sync with the robot dynamics. One example of this is [10], which employed a network of interconnected Matsuoka oscillators to govern the motion of a five-link planar biped in a simulated environment. Their research demonstrated that stable and adaptable locomotion could be achieved through synchronization between the rhythmic behaviors of the interconnected CPGs and the motions of the robot's mechanical components. [11] employed a CPG network, consisting of Matsuoka oscillators, to control the walking behavior of the Nao robot in both simulation and real-world settings. They utilized a pacemaker oscillator within the CPG network to generate a master signal for controlling the entire network. Furthermore, they optimized the parameters of the CPG network using a genetic algorithm. [12] used a hierarchical control mechanism for legged locomotion where an optimized CPG network is used for joint control and a neural network acts as a high-level controller for modulating the CPG network.

2) Model-based control: Conventional control approaches rely on analytical models describing the robot and environment dynamics. Such models typically solve an optimization problem using a simplified or abstract system model to predict the behavior of the system over a horizon and determine optimal control inputs [13]. Some approaches have also proposed adding bounded uncertainty constraints to the model predictive control to guarantee safe motion [14]. These approaches can achieve some impressive performances and robust motion, as shown in [1]. Many model-based approaches apply a modular controller design, whereby the controller can be broken down into simpler decoupled sub-modules. Different sub-modules can then be tuned to adapt to different behavioral properties such as body height control, step length, and step height. However, these methods suffer from inaccuracies in the approximate system model, which fails to completely capture all the environment dynamics.

3) Model-free control: Deep reinforcement learning (DRL) has emerged as an attractive solution to address some of the challenges faced by traditional model-based methods. Reinforcement learning (RL) approaches overcome the problems with model dynamics inaccuracy by directly learning controllers by interacting with the environment. The primary idea is to tune a controller output based on state feedback from the environment to maximize the expected reward function. As shown in [15], the policy can be trained blind in simulations with proprioceptive feedback and memory-based network architecture to generalize to different terrains. Different policy gradient techniques have also been explored to learn an end-to-end control policy [16], [17]. Policy gradient methods directly learn the parameters of the optimal policy that describes actions for the controller. These methods usually suffer from slow learning with extensive simulation time requirements.

[3] introduces the CPG-actor-critic reinforcement learning method for training CPG controllers for bipedal robots. It focuses on improving robot locomotion by learning an actor policy using policy gradients. In recent studies such as [6], [7], researchers have employed RL-PI^{BB} to learn pre-motor network weights for locomotion control policies of a hexapod robot. They initialize the CPG-RBF network with a warm start and then generate policy rollouts with exploration noise in the parameters. Based on the fitness of each rollout the exploration noise is used to update policy parameters with fitness-weighted averaging. This approach involves developing an open-loop base controller, with modular complex behaviors learned on top of the base policy.

4) Neural Evolutionary Learning: Neuroevolution refers to the use of an Evolutionary Algorithm (EA) to optimize the parameters of a neural network. Researchers have shown that neuroevolution approaches can compete with popular algorithms for deep reinforcement learning problems such as DQN, A3C, and ES in the challenging Atari domain and in solving legged robot control problems [18]. Several works, such as [19] and [20], have successfully employed Evolutionary Algorithms (EA) for CPG-based reference generation, demonstrating enhanced adaptability and efficiency in complex locomotion scenarios. The use of EA in these contexts also brings a more robust solution search to the optimization process, enabling the discovery of network configurations and synaptic weights that facilitate effective and adaptive control in robotic systems. Additionally, studies like [21] and [5] have leveraged EA for the optimization of CPG parameters in the context of human movement generation and bipedal locomotion control, respectively.

The integration of CPG with Evolutionary Algorithms emerges as a promising alternative to the conventional CPG-RL approach. This novel combination leverages the strengths of both CPG for biologically inspired rhythmic patterns and EA for efficient optimization, offering a pathway to develop adaptive, efficient, and versatile locomotion models for legged robots.

III. Methodology

Our implementation is based on the CPG-RBFN architecture proposed in [6]. An overview model of the architecture can be seen in Figure 1. In this model, a Central Pattern Generator (CPG) is used to generate sinusoidal input signals to induce a locomotion gait into the model. Additionally, a Radial Basis Function Network (RBFN) shapes the CPG input to control a legged robot. In this study, we used an Evolutionary Algorithm (EA) to learn both the output weights and the RBFN centers of the CPG-RBF network and tested the performance of our approach on OpenAI's Half Cheetah environment.

A. Central Pattern Generator

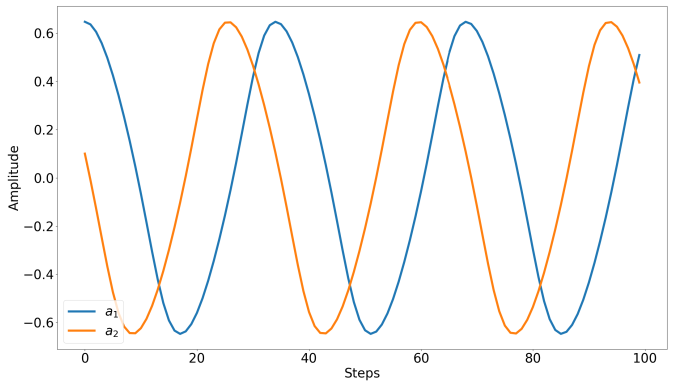
CPGs have interesting properties for robot control such as simple and smooth frequency modulation, and few control parameters. They've been used before to generate periodic movement patterns [12]. We use the sinusoidal wave produced by a CPG to induce a locomotion gait into our model. The CPG network used in our implementation is the SO(2) Neural Oscillator from [22]. This two-neuron model produces sinusoidal output without the need for a sinusoidal input. The model is described by the following set of equations in Equation 1

$$W_{CPG} = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix} = \alpha * \begin{pmatrix} cos(\phi) & sin(\phi) \\ -sin(\phi) & cos(\phi) \end{pmatrix}$$

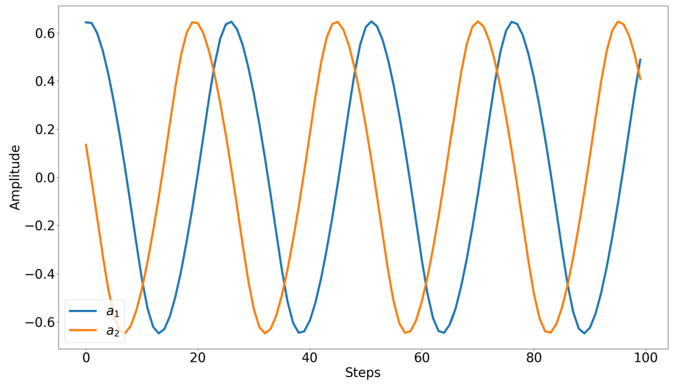
$$a_1(t+1) = w_{11} tanh(a_1(t)) + w_{12} tanh(a_2(t)) \tag{1}$$

$$a_2(t+1) = w_{21} tanh(a_1(t)) + w_{22} tanh(a_2(t))$$

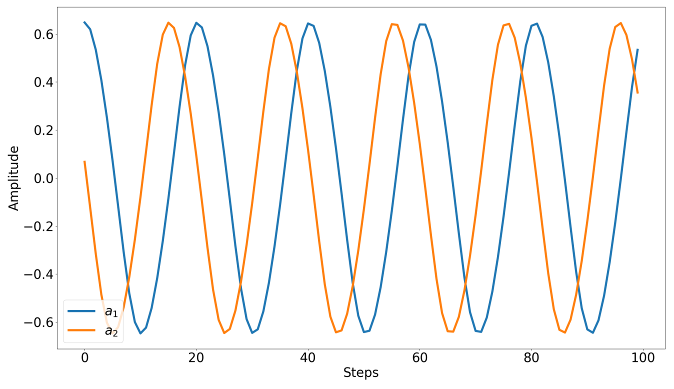
where ϕ has a range of $-\pi < \phi < \pi$ and represents the frequency, α controls the amplitude. The weights W_{CPG} shape the previous activations $a_n(t)$ to produce the next activations $a_n(t+1)$.



(a) CPG parameter set Nominal: $\phi = 0.06\pi$, $\alpha = 1.1$



(b) CPG parameter set Medium Speed: $\phi = 0.08\pi$, $\alpha = 1.1$



(c) CPG parameter set High Speed: $\phi = 0.1\pi$, $\alpha = 1.1$

Fig. 2: Input CPG signals at different frequencies with parameters defined in Table II

B. Radial Basis Function Network

A typical Radial Basis Function (RBF) network has 3 layers: an input layer with no weights, a hidden layer with a non-linear RBF activation function, and an output layer. In our model, the CPG comprises our input layer, and its output is fed to the hidden layer. The hidden layer uses a radial basis function as its activation function. Our RBF activation function is the following Gaussian function in Equation 2

$$RBF_j = e^{-\gamma((a_0 - c_{0j})^2 + (a_1 - c_{1j})^2)} \tag{2}$$

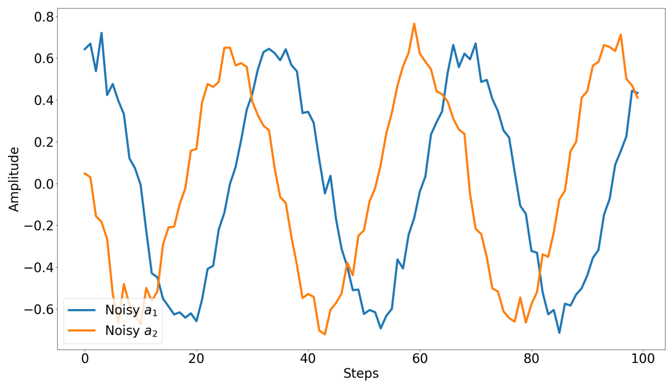


Fig. 3: CPG parameter set Nominal with additive white Gaussian noise with mean $\mu = 0$ and standard deviation sigma $\sigma = 0.05$.

where $\gamma = 25$ determines the decay of the Gaussian function. This value was set experimentally to provide consistent results on our approach and baseline. a_0 and a_1 are the activation outputs from the CPG layer. The learnable parameters c_{0j} and c_{1j} are the centers of each RBF unit. The variable $j = 20$ is the number of RBF units. At 20 RBF units the model learned a simple walking behavior and increasing the number of units yielded no significant improvement in our tests.

Our output layer uses the hyperbolic tangent $tanh(RBF_j)$ activation function. This activation function bounds our output inside the range $[-1, 1]$, the action space of the Half Cheetah environment also uses this range.

C. CPG-RBFN Model

The CPG output by default is wave-shaped and cannot be easily reshaped using only the CPG parameters. By combining it with the RBF network, we can reshape the CPG output by amplifying or reducing specific parts of the CPG signal. These features allow the CPG-RBFN model to produce arbitrarily shaped rhythmic trajectories for the joints of a legged robot [6]. These trajectories are encoded in the w_{jk} weights connecting the RBF layer to the motor outputs as seen in Figure 1. Having just a single hidden layer allows the CPG-RBFN model to have a faster learning and convergence rate when compared to deep neural networks such as those in [2]. Additionally, interpreting the effects of the RBF neurons in the hidden layer is simple, because the RBF layer activation encodes the motor output at a particular phase in the CPG input. Our CPG-RBFN implementation has the following learnable parameters: 20 RBF units, with one pair of centers c_{0j} and c_{1j} each; and 120 weights w_{jk} from the RBF layer to the motor output layer.

D. Neuroevolution

In the Half Cheetah environment, we only get a sparse reward at the end of an episode given by the environment's configuration. Neuroevolution can be used for optimizing neural networks and is comparable to backpropagation [23]. One of the main benefits of neuroevolution is that it removes the need to calculate gradients to optimize the network's parameters. Therefore, to optimize the RBF centers and output weights through neuroevolution we are using an Evolutionary Algorithm (EA). The algorithm keeps a population of individuals (a set of centers and weights). Individuals' fitness is determined by their total rewards after going through the simulation environment. Each individual simulates a total of 1000 steps (corresponding to 10 seconds of real-world time) and gets rewards based on how much they move forward along with a penalty for large actions. To generate offspring we used a mutation operator on individuals selected through Roulette Wheel Selection [24]. The mutation operator selects 10% of the learnable parameters of a parent randomly and adds a Gaussian noise with mean ($\mu = 0$) and standard deviation ($\sigma = 0.01$) to them. The offspring are evaluated and added to the population, then the individuals with a low fitness score are pruned off up to the generation size. The pseudocode can be seen in Algorithm 1.

Algorithm 1 CPG-RBFN Neuroevolution

```
Require: $genSize > 0$
Require: $num\_generations > 0$
  function NEUROEVOLUTION($genSize, num\_generations$)
      Each $individual$ contains $[c_{0j}, c_{1j}, w_{jk}, fitness]$
      $steps \leftarrow 1000$
      $generation \leftarrow$ random set of $individual$
      for $num\_generations$ do
          $genFitness \leftarrow runEnvironment(generation, steps)$
          $parents \leftarrow rouletteWheelSelect(genFitness)$
          $children \leftarrow mutate(parents)$
          $generation \leftarrow [generation, children]$
          $generation \leftarrow$ sort $generation$ with $fitness$
          $generation \leftarrow$ from $generation$ up to $genSize$
      end for
      $bestIndividual \leftarrow generation[0]$
      return $bestIndividual$
  end function
```

E. Environment

We conduct our experiments in Open AI's Gymnasium simulated environment to validate the effectiveness of the neural-evolutionary locomotion controller. We use the Half Cheetah environment as a simplified legged system to test our algorithm. The robot is a two-dimensional legged system consisting of a front and hind leg, each with three controllable joints. The action space of the robot is the torque applied to all six joints in the range $[-1, 1]$. The observation space from the environment consists of a complete state feedback of the robot consisting of motor angle, angular velocities, body position, and velocities. However, our controller network and the baseline RL-PI^{BB} both are open-loop so we do not use any state feedback from the environment for learning a control policy. We get a positive reward for the forward distance traveled by the robot and penalize large actions on the joints. An episode is terminated after 1000 simulation steps. In this environment, a total episode reward close to 1000 or above would then be considered a good reward. The experiments conducted and the subsequent results are discussed in the next section.

IV. Experiments and Results

We compare our method of Neuro-Evolutionary learning against the baseline of RL-PI^{BB} approach as defined in [7].

A. Baseline setup

The baseline starts the learning with initializing the CPG-RBF network with a warm start of initial output weights (w_{jk}). Then during each epoch, K rollouts of the network are generated by adding a Gaussian exploration noise with mean ($\mu = 0$) and standard deviation (σ). This standard deviation is decayed by a decay factor (η) after every epoch. Exploration noise of each rollout is then reward-weighted averaged and used to update the policy parameters. Reward-weighted averaging of the parameter update is the main crux of Probability-Based Black Box Optimization.

So the baseline essentially does a local search of parameters around the initial start and is more prone to get stuck in a local minima. That is one of the major reasons for our hypothesis that an evolutionary algorithm would perform better as it enables a more extensive parameter space search. The experimental parameters for our method and the baseline are discussed in the next section.

B. Experimental Parameters

For the baseline RL-PI^{BB} and every agent in Neuroevolution, we initialize the network with random output weights instead of a warmstart. Each rollout in the baseline and each agent in our approach runs for 1000 simulation steps corresponding to 10 seconds in the real world. The rest of the experimental parameters for both approaches are listed in Table I.

TABLE I: Experimental parameters

Parameter set	Parameter name	Value
Common	RBF units	20
	Decay in RBF neuron	25
	Maximum simulation steps	1000
Neuroevolution	Generation size	10
	Mutation percentage	10%
	μ for mutation noise	0
	σ for mutation noise	0.01
RL-PI^{BB}	Number of Epochs	500
	Number of Rollouts	10
	μ for exploration noise	0
	Initial σ for exploration noise	0.05
	Decay factor η for σ	0.995

We use the CPG input signal as defined in the implementation of [7] as the nominal signal. To control the locomotion gait speed on a legged robot, the frequency of the locomotion has to be increased. With this in mind, we have defined two more sets of parameters to compare as shown in Table II. Each set of parameters in the table produces sinusoidal output with different frequencies. These differences can be seen in Figure 2. In addition to varying the frequencies, additive white Gaussian noise was also added during training, obtaining an input signal like the one seen on Figure 3.

TABLE II: CPG parameters of the test cases

Set	Frequency (ϕ)	Amplitude (α)
Nominal	0.06π	1.1
Medium Speed	0.08π	1.1
High Speed	0.1π	1.1

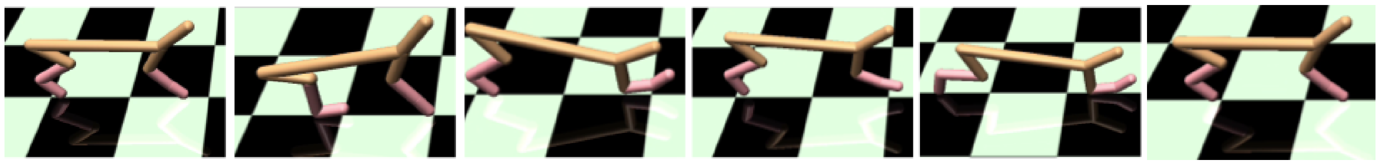
C. Tests and Results

In the first test, we use a single CPG input signal with the Nominal set of parameters from Table II. The signal plot is visualized in Figure 2a. In this test case, we use the same input signal to train all the individuals to get a single control policy for just one specific input. The rolling mean of the best agent reward in each generation for evolutionary learning and of best rollout in each epoch for the baseline RL-PI^{BB} is shown in Figure 5. Firstly we fixed the RBF centers in our method as defined in [7] to compare it against the baseline where they always have fixed centers. It can be seen in Figure 5a that our method struggles to match the performance of RL-PI^{BB} method with fixed centers. So we then learn the RBF centers in the network along with the output weights and as seen in Figure 5b our method can match the performance of the baseline with fixed centers. However, it can be noted that our method takes longer to converge on the nominal input signal.

For the subsequent tests, we always compare neuroevolution with learned RBF centers to the baseline with fixed centers. The baseline always has fixed RBF centers as that was the method originally proposed in [7]. For each test, we generate the same rolling mean of the best reward of the agent in each generation for neuroevolution and for best rollout in each epoch in the baseline.

For the second test, we wanted to test the adaptability of the controller towards different input signals. So we train the policy over all three parameter sets from Table II to get a single policy that can handle different velocities for the robot. In our approach, during training each agent in the generation randomly gets one of the three CPG signals as input for learning. For comparison to the baseline, during each epoch for RL-PI^{BB} the rollouts generated randomly get one of the three CPG signals as input for learning. As seen in the reward curves in Figure 6, our method outperforms the baseline significantly getting over 1000 reward whereas the baseline gets stuck around the 400 mark. This shows that our method is more adaptive to different frequencies in the input and can achieve varying velocities with the robot. During training, we observed that the baseline struggles to perform well on alternating signals due to the limitations posed by the fixed RBF centers. The RL-PI^{BB} method was also observed to be sensitive to a good initialization of output weights (w_{jk}), requiring some prior knowledge of the environment and the system.

For the third test, we wanted to test the robustness of the controller towards any noise that might be present in



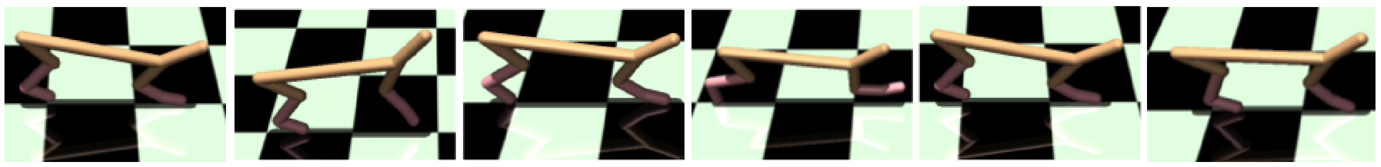
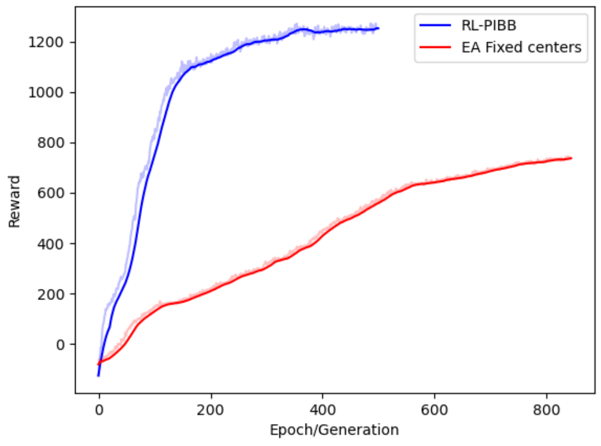
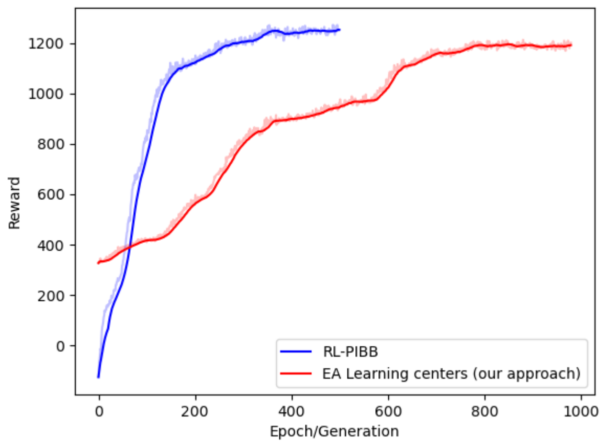


Fig. 4: Sequence of learned motion for a) Nominal input, b) Alternating Noisy input



(a) RL-PI^{BB} vs Evolutionary learning with fixed RBF centers



(b) RL-PI^{BB} vs Evolutionary learning with learnt RBF centers

Fig. 5: Reward curve on Nominal CPG input

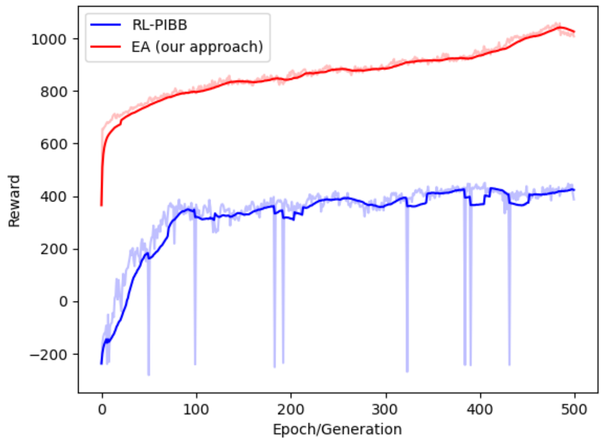


Fig. 6: Reward curve for RL-PI^{BB} vs Evolutionary on Alternating CPG input

the system. Noise could be introduced in the system either from the environment or from the hardware when deployed in the real world. By making the controller robust to noisy input, we could enable it to be deployed on budget hardware where the input sinusoidal signal is generated from any source. To emulate noise in the simulations, we introduce additive white Gaussian noise to each of the three CPG input signals. An example of noisy input signal is shown in Figure 3. Again during training each agent in the generation and each rollout in the epoch randomly gets one of the three CPG signals with added noise as input for learning. As seen in the reward curves in Figure 7, our method excels at demonstrating robustness to random noise in the system as compared to the baseline. We can achieve close to 1200 reward in just 500 generations of training. The baseline RL-PI^{BB} fails to achieve any meaningful locomotion behavior with added noise to the system.

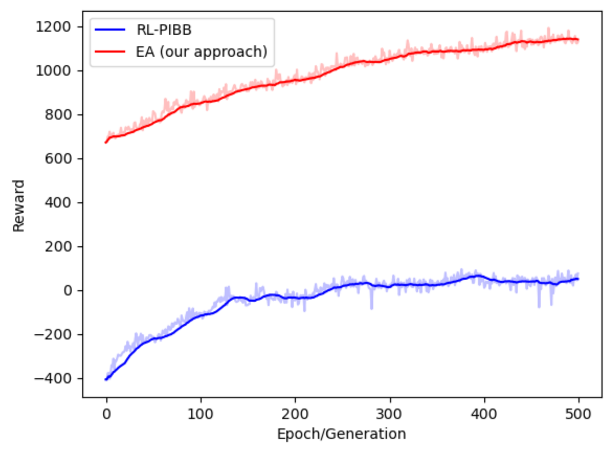


Fig. 7: Reward curve for RL-PI^{BB} vs Evolutionary on Alternating Noisy CPG input

We visualize a small sequence of the learned motion from our method in Figure 4. On the top sequence is the motion from the first test using just the nominal CPG input signal. The bottom sequence shows the motion from the third test using all three input signals with added noise. In both cases as seen we can achieve meaningful cyclic locomotion behavior for the robot.

V. Conclusion and Future Work

Our method extends the CPG-RBFN controller presented in [6] with a neuroevolutionary learning algorithm. We compare our method to the Probability-Based Black Box Optimization approach proposed by the same authors in [7]. Our method enhances the controller's adaptability to changes in the input signal of varying frequencies and successfully learns motion for different velocities of the robot. Our method also demonstrates robustness to noise present in the system. As compared to the RL-PI^{BB} learning, our method explores the solution space better which therefore removes the need for a warmstart or good initialization of network parameters. Not depending on a good initialization also further eliminates the need for any prior extensive system knowledge. Therefore, we have improved upon locomotion for a legged system with a very simplistic network and achieved robustness to noise and adaptiveness to changes in the input.

For future work, we plan to extend our approach to other legged systems such as quadrupeds and 3D systems to test the generalizability over different legged architecture and morphologies. It would be worth exploring the changes needed to deploy the same controller for dynamically stable-legged robots.

References

[1] I. Dakhli, E. Maherzi, and M. Besbes, "Robust walking control algorithm of biped robot in rough ground," in *2016 13th International Multi-Conference on Systems, Signals & Devices (SSD)*, pp. 762–767, Mar. 2016.

[2] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, "Learning agile and dynamic motor skills for legged robots," *Science Robotics*, vol. 4, p. eaau5872, Jan. 2019.

[3] Y. Nakamura, T. Mori, M.-a. Sato, and S. Ishii, "Reinforcement learning for a biped robot based on a CPG-actor-critic method," *Neural Networks*, vol. 20, pp. 723–735, Aug. 2007.

[4] W. Ouyang, H. Chi, J. Pang, W. Liang, and Q. Ren, "Adaptive Locomotion Control of a Hexapod Robot via Bio-Inspired Learning," *Frontiers in Neurorobotics*, vol. 15, p. 627157, Jan. 2021.

[5] G. Endo, J. Morimoto, T. Matsubara, J. Nakanishi, and G. Cheng, "Learning CPG-based Biped Locomotion with a Policy Gradient Method: Application to a Humanoid Robot," *The International Journal of Robotics Research*, vol. 27, pp. 213–228, Feb. 2008.

[6] M. Thor, T. Kulvicius, and P. Manoonpong, "Generic Neural Locomotion Control Framework for Legged Robots," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, pp. 4013–4025, Sept. 2021.

[7] M. Thor and P. Manoonpong, "Versatile modular neural locomotion control with fast learning," July 2021. arXiv:2107.07844 [cs].

[8] F. Stulp and O. Sigaud, "Policy Improvement Methods: Between Black-Box Optimization and Episodic Reinforcement Learning," Oct. 2012.

[9] A. H. Cohen, "10 Control of a robot leg with an adaptive aVLSI CPG chip 11 12 M. Anthony Lewis *, Mitra J. Hartmann , Ralph Etienne-Cummings , 13," 2001.

[10] G. Taga, Y. Yamaguchi, and H. Shimizu, "Self-organized control of bipedal locomotion by neural oscillators in unpredictable environment," *Biological Cybernetics*, vol. 65, pp. 147–159, July 1991.

[11] J. Cristiano, D. Puig, and M. A. García, "Locomotion Control of a Biped Robot through a Feedback CPG Network," in *ROBOT2013: First Iberian Robotics Conference* (M. A. Armada, A. Sanfeliu, and M. Ferre, eds.), vol. 252, pp. 527–540, Cham: Springer International Publishing, 2014.

[12] S. Auddy, S. Magg, and S. Wermter, "Hierarchical Control for Bipedal Locomotion using Central Pattern Generators and Neural Networks," in *2019 Joint IEEE 9th International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, (Oslo, Norway), pp. 13–18, IEEE, Aug. 2019.

[13] M. Kasaei, A. Ahmadi, N. Lau, and A. Pereira, "A Robust Model-Based Biped Locomotion Framework Based on Three-Mass Model: From Planning to Control," in *2020 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pp. 257–262, Apr. 2020.

[14] N. A. Villa and P.-B. Wieber, "Model predictive control of biped walking with bounded uncertainties," in *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, pp. 836–841, Nov. 2017. ISSN: 2164-0580.

[15] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," *Science Robotics*, vol. 5, p. eabc5986, Oct. 2020. Publisher: American Association for the Advancement of Science.

[16] X. Wu, S. Liu, T. Zhang, L. Yang, Y. Li, and T. Wang, "Motion Control for Biped Robot via DDPG-based Deep Reinforcement Learning," in *2018 WRC Symposium on Advanced Robotics and Automation (WRC SARA)*, pp. 40–45, Aug. 2018.

[17] M. Navaneethakrishnan, P. Pushpa, T. T, T. A. Mohanaprakash, B. Dhanwanth, and F. A. A. S, "Design of Biped Robot Using Reinforcement Learning and Asynchronous Actor-Critical Agent (A3C) Algorithm," in *2023 2nd International Conference on Vision Towards Emerging Trends in Communication and Networking Technologies (ViTECoN)*, pp. 1–6, May 2023.

[18] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune, "Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning," Apr. 2018. arXiv:1712.06567 [cs].

[19] K. Adak, *Quadruped locomotion reference synthesis with central pattern generators tuned by evolutionary algorithms*. PhD thesis, 2013.

[20] A. A. Saputra, T. Takeda, J. Botzheim, and N. Kubota, "Multi-objective evolutionary algorithm for neural oscillator based robot locomotion," in *IECON 2015 - 41st Annual Conference of the IEEE Industrial Electronics Society*, (Yokohama), pp. 002655–002660, IEEE, Nov. 2015.

[21] C. Bauer, S. Braun, Y. Chen, W. Jakob, and R. Mikut, "Optimization of Artificial Central Pattern Generators with Evolutionary Algorithms," *Proc., 18. Workshop Computational Intelligence, Universitätsverlag Karlsruhe*, pp. 40–54, Dec. 2008.

[22] F. Pasemann, M. Hild, and K. Zahedi, "SO(2)-Networks as Neural Oscillators," in *Computational Methods in Neural Modeling* (G. Goos, J. Hartmanis, J. Van Leeuwen, J. Mira, and J. R. Álvarez, eds.), vol. 2686, pp. 144–151, Berlin, Heidelberg: Springer Berlin Heidelberg, 2003. Series Title: Lecture Notes in Computer Science.

[23] K. O. Stanley, J. Clune, J. Lehman, and R. Miikkulainen, "Designing neural networks through neuroevolution," *Nature Machine Intelligence*, vol. 1, pp. 24–35, Jan. 2019.

[24] N. Behera, "Analysis of microarray gene expression data using information theory and stochastic algorithm," in *Handbook of Statistics*, vol. 43, pp. 349–378, Elsevier, 2020.