CPG-Actor: Reinforcement Learning for Central Pattern Generators

Luigi Campanaro, Siddhant Gangapurwala, Daniele De Martini,
Wolfgang Merkt, and Ioannis Havoutis

Oxford Robotics Institute
{luigi, siddhant, daniele, wolfgang, ioannis}@robots.ox.ac.uk

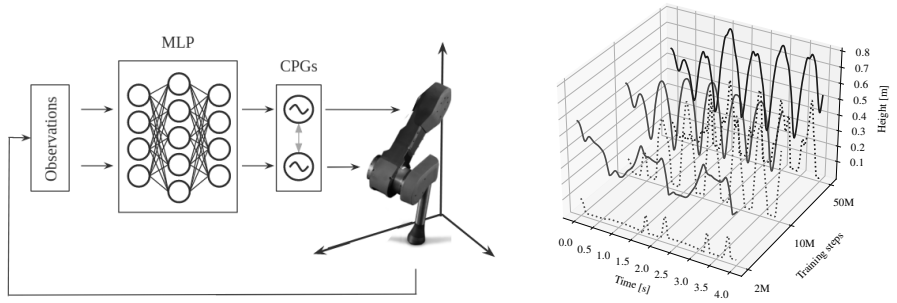
Abstract Central Pattern Generators (CPGs) have several properties desirable for locomotion: they generate smooth trajectories, are robust to perturbations and are simple to implement. However, they are notoriously difficult to tune and commonly operate in an open-loop manner. This paper proposes a new methodology that allows tuning CPG controllers through gradient-based optimisation in a Reinforcement Learning (RL) setting. In particular, we show how CPGs can directly be integrated as the Actor in an Actor-Critic formulation. Additionally, we demonstrate how this change permits us to integrate highly non-linear feedback directly from sensory perception to reshape the oscillators' dynamics. Our results on a locomotion task using a single-leg hopper demonstrate that explicitly using the CPG as the Actor rather than as part of the environment results in a significant increase in the reward gained over time (20x more) compared with previous approaches. Finally, we demonstrate how our closed-loop CPG progressively improves the hopping behaviour for longer training epochs relying only on basic reward functions.

Keywords: Central Pattern Generators, Reinforcement Learning, Feedback Control, Legged Robots

1 Introduction

The increased manoeuvrability associated with legged robots in comparison to wheeled or crawling robots necessitates complex planning and control solutions. The current state-of-the-art for high-performance locomotion are modular, model-based controllers which break down the control problem in different submodules [1].This rigorous approach is rooted in the knowledge of every portion of the motion, but it is also limited by heuristics handcrafted by engineers at each of the stages.

While the field of legged robot control has been dominated over the last decades by conventional control approaches, recently, data-driven methods demonstrated unprecedented results that outpaced most of the classical approaches in terms of robustness and dynamic behaviours [2]. In particular, controllers trained using deep-RL utilise a Neural Network (NN) policy to map sensory information to low-level actuation commands. As a result, controllers trained with



(a) (b)

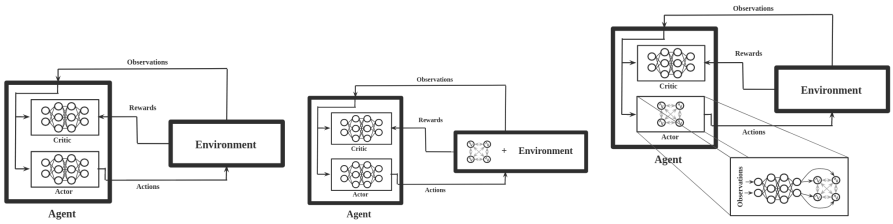
Figure 1: The experiments are carried out on a classic Reinforcement Learning (RL) benchmark – the single-leg hopper based on the ANYmal quadruped robot [3]. It hops along the vertical axis and is controlled by Central Pattern Generators (CPGs). Closed-loop feedback is incorporated using a jointly trained Multilayer Perceptron (MLP) network (Fig. 1a). To demonstrate that the CPG-Actor progressively learns to jump higher peaks of both the hip (solid line) and foot (dotted line) heights (Fig. 1b) are shown.

RL exhibit behaviours that cannot be hand-crafted by engineers and are further robust to events encountered during the interaction with the environment. However, widely-used NN architectures, such as MLP, do not naturally produce the oscillatory behaviour exhibited in natural locomotion gaits and as such require long training procedures to learn to perform smooth oscillations.

A third family of controllers have been used with promising results for robot locomotion: CPGs, a biologically-inspired neural network able to produce rhythmic patterns. However, very few design principles are available, especially for the integration of sensor feedback in such systems [4] and, although conceptually promising, we argue that the full potential of CPGs has so far been limited by insufficient sensory-feedback integration.

The ability of Deep-NNs to discover and model highly non-linear relationships among the observation – the inputs – and control signals – the outputs – makes such approaches appealing for control. In particular, based on Deep-NNs, Deep-RL demonstrated very convincing results in solving complex locomotion tasks [2, 5] and it does not require direct supervision (but rather learns through interaction with the task). Hence, we argue that combining Deep-RL with CPGs could improve the latter's comprehension of the surrounding environment. However, optimising Deep-NN architectures in conjunction with CPGs requires adequate methods capable of propagating the gradient from the loss to the parameters, also known as backpropagation.

To address this, this paper introduces a novel way of using Deep-NNs to incorporate feedback into a fully differentiable CPG formulation, and apply Deep-RL to jointly learn the CPG parameters and MLP feedback.



(a) (b) (c)

Figure 2: (a) represents the basic actor-critic Deep-RL method adopted for continuous action space control. (b) illustrates the approach proposed in [10–13], which consists in a classic actor-critic with CPGs embedded in the environment. (c), instead, is the approach proposed in the present work, which includes the CPGs alongside the MLP network in the actor critic architecture.

1.1 Related Work

Our work is related to both the fields of CPG design and RL, in particular to the application of the latter for the optimisation of the former's parameters.

CPGs are very versatile and have been used for different applications including non-contact tasks such as swimmers [6], modular robots [7] and locomotion on small quadrupeds [8]. The trajectories CPGs hereby generate are used as references for each of the actuators during locomotion and a tuning procedure is required to reach coordination. The optimisation of CPG-based controllers usually occurs in simulation through Genetic Algorithms (GA), Particle Swarm Optimisation (PSO) or expert hand-tuning [6, 8].

To navigate on rough terrain sensory feedback is crucial (e.g. in order to handle early or late contact), as shown in [9]: here, a hierarchical controller has been designed, where CPGs relied on a state machine which controlled the activation of the feedback.

Similarly to [9], [8] also uses feedback, this time based on gyroscope velocities and optical flow from a camera to modify the CPGs output in order to maintain balance. However, in [8] the authors first tune CPGs in an open-loop setting and then train a NN with PSO to provide feedback (at this stage the parameters of the CPGs are kept fixed). We follow the same design philosophy in the sense that we preprocess the sensory feedback through a NN; yet, we propose to tune its parameters in conjunction with the CPG.

Actor-critic methods [14] rely on an explicit representation of the policy independent from the value function Fig. 2a.

Researchers applied RL to optimise CPGs in different scenarios [10]. The common factor among them is the formulation of the actor-critic method; yet, they include the CPG controller in the environment – as depicted in Fig. 2b. In other words, the CPG is part of the (black-box) environment dynamics. According to the authors [13], the motivations for including CPGs in the environment are their intrinsic recurrent nature and the amount of time necessary to train them, since CPGs have been considered Recurrent Neural Networks (RNNs)

(which are computationally expensive and slow to train). In [10] during training and inference, the policy outputs a new set of parameters for the CPGs in response to observations from the environment at every time-step. Conversely, in [13] the parameters are fixed and, similarly to [8], CPGs receive inputs from the policy. However, whether the CPGs parameters were new or fixed every time-step, they all considered CPGs as part of the environment rather than making use of their recurrent nature as stateful networks. We exploit this observation in this paper.

1.2 Contributions

In this work, we combine the benefits of CPGs and RL and present a new methodology for designing CPG-based controllers. In particular, and in contrast to prior work, we embed the CPG directly as the actor of an Actor-Critic framework instead of it being part of the environment. The advantage of directly embedding a dynamical system is to directly encode knowledge about the characteristics of the task (e.g., periodicity) without resorting to recurrent approaches. The outcome is CPG-Actor, a new architecture that allows end-to-end training of coupled CPGs and a MLP for sensory feedback by means of Deep-RL. In particular, our contributions are:

1. For the first time – to the best of our knowledge – the parameters of the CPGs can be directly trained through state-of-the-art gradient-based optimisation techniques such as Proximal Policy Optimisation (PPO) [15], a powerful RL algorithm). To make this possible, we propose a fully differentiable CPG formulation (Sec. 2.1) along with a novel way for capturing the state of the CPG without unrolling its recurrent state (Sec. 2.1).
2. Exploiting the fully differentiable approach further enables us to incorporate and jointly tune a MLP network in charge of processing feedback in the same pipeline.
3. We demonstrate a roughly twenty times better training performance compared with previous state-of-the-art approaches (Sec. 4).

2 Methodology

As underlying oscillatory equation for our CPG network, we choose to utilise the Hopf oscillator [16] in a tensorial formulation, eq. (2).

Differently to previous approaches presented in Sec. 1.1, we embed CPGs directly as part of the actor in an actor-critic framework as shown in Fig. 2c. Indeed, the policy NN has been replaced by a combination of an MLP network for sensory pre-processing and CPGs for action computation, while the value function is still approximated by an MLP network.

In practice, in our approach the outputs of the actor are the position commands for the motors. In [10], instead, the actor (MLP-network) outputs the parameters of the CPGs, that are then used by the environment (that includes

the CPGs) to compute the motor commands. In this sense, there is a substantial difference in the architectures: in CPG-Actor, both the CPGs' and MLP's parameters are trained, while in [10] only the MLP's parameters are trained and the CPGs' ones are derived at runtime, being the output of the network.

However, a naïve integration of CPGs into the Actor-Critic formulation is error-prone and special care needs to be taken i) to attain differentiability through the CPG actor in order to exploit gradient-based optimisation techniques; ii) not to neglect the hidden state as CPGs are stateful networks.

We are going to analyse these aspects separately in the following sections.

2.1 Differentiable Central Pattern Generators

Since equations in [16] describe a system in continuous time, we need to discretise them for use as a discrete-time robot controller, as in eq. (1):

$$\begin{aligned}
\dot{\theta}_i^t &= 2\pi\nu_i(d_i^t) + \zeta_i^t + \xi_i^t \\
\zeta_i^t &= \textstyle\sum_j r_j^{t-1} w_{ij} \sin(\theta_j^{t-1} - \theta_i^{t-1} - \phi_{ij}) \\
\ddot{r}_i^t &= a_i(\tfrac{a_i}{4}(\rho_i(d_i^t) - r_i^{t-1}) - \dot{r}_i^{t-1}) + \kappa_i^t \\
x_i^t &= r_i^t \cos(\theta_i^t)
\end{aligned} \tag{1}$$

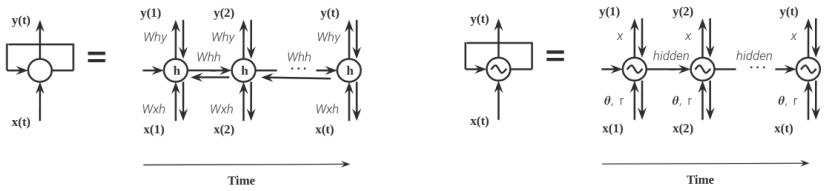
where $\cdot^t$ describes the value at the t-th time-step, θ_i and r_i are the scalar state variables representing the phase and the amplitude of oscillator i respectively, ν_i and ρ_i determine its intrinsic frequency and amplitude as function of the input command signals d_i, and a_i is a positive constant governing the amplitude dynamics. The effects of the couplings between oscillators are accounted in ζ_i and the specific coupling between i and j are defined by the weights w_{ij} and phase ϕ_{ij}. The signal x_i represents the burst produced by the oscillatory centre used as position reference by the motors. Finally, ξ_i and κ_i are the feedback components provided by the MLP network.

In order to take advantage of modern technology for parallel computation, e.g. GPUs, there is a strong need to translate the equations in [16] into a tensorial formulation (2) which describes the system in a whole enabling batch computations. Let N be the number of CPGs in the network, then:

$$\begin{aligned}
\dot{\Theta}^t &= 2\pi C_\nu(V, D^t) + Z^t \mathbf{1} + \Xi^t \\
Z^t &= (WV) * (\Lambda R^{t-1}) * \sin(\Lambda\Theta^{t-1} - \Lambda^\intercal\Theta^{t-1} - \Phi V) \\
\ddot{R}^t &= (AV) * (\tfrac{AV}{4}(P(V, D^t) - R^{t-1}) - \dot{R}^{t-1}) + K^t \\
X^t &= R^t \cos(\Theta^t)
\end{aligned} \tag{2}$$

Here, $\Theta \in \mathbb{R}^N$ and $R \in \mathbb{R}^N$ are the vectors containing θ_i and r_i, while $\Xi \in \mathbb{R}^N$ and $K \in \mathbb{R}^N$ contain ξ_i and κ_i respectively. $V \in \mathbb{R}^M$ contains the M, constant parameters to be optimised of the network composed by the N CPGs.

This said, $C_\nu : \mathbb{R}^M, \mathbb{R}^d \to \mathbb{R}^N$, $P : \mathbb{R}^M, \mathbb{R}^d \to \mathbb{R}^N$ and $A \in \mathbb{R}^{N \times M}$ are mappings from the set V and the command $D^t \in \mathbb{R}^d$ to the parameters that lead ν_i, ρ_i and a_i respectively. $Z \in \mathbb{R}^{N \times N}$ instead takes into consideration the effects of the couplings of each CPG to each CPG; all the effect to i-th CPG will be then the sum of the i-th row of Z as in $Z\,\mathbf{1}$, where $\mathbf{1}$ is a vector of N elements with



(a) (b)

Figure 3: The images above show the difference between back-propagation for classic RNNs (3a) and CPGs (3b). In particular to train RNNs, the matrices W_{xh}, W_{hy}, W_{hh} have to be tuned, where W_{hh} regulates the evolution between two *hidden states*. Instead, for CPGs only the parameters in $\dot{\theta}_i$ and $\ddot{r}_i$ (eq. (2)) need tuning, while the evolution of the *hidden state* is determined by an integration operation.

value 1. Within Z, $W \in \mathbb{R}^{N\times N\times M}$ and $\Phi \in \mathbb{R}^{N\times N\times M}$ extrapolate the coupling weights and phases from V, while $\Lambda \in \mathbb{R}^{N\times N\times N}$ encodes the connections among the nodes of the CPG network.

The reader can notice how in (2) only already-differentiable operations have been utilised and that the MLP's output, i.e. the CPGs' feedback, is injected as a sum operation, enabling the gradient to backpropagate through the MLP network as well. This further enables us to compute the gradient of each of the parameters in (2) (CPGs and MLP) with respect to the RL policy's loss using the auto differentiation tools provided by PyTorch.

Recurrent state in CPGs In order to efficiently train CPGs in a RL setting, we need to overcome the limitations highlighted in [13]: In fact, CPGs are considered similar to RNNs (due to their internal state) and consequently they would have taken a significant time to train. In this section, we show how we can reframe CPGs as stateless networks and fully determine the state from our observation without the requirement to unroll the RNN.

RNNs are stateful networks, i.e. the state of the previous time-step is needed to compute the following step output. As a consequence, they are computationally more expensive and require a specific procedure to be trained. RNNs rely on Backpropagation Through Time (BPTT),Fig. 3a , which is a gradient-based technique specifically designed to train stateful networks. BPTT unfolds the RNN in time: the unfolded network contains t inputs and outputs, one for each time-step. Undeniably, CPGs have a recurrent nature and as such require storing the previous hidden state. However, differently from RNNs, the transition between consecutive hidden states, represented by the matrix W_{hh}, in CPGs is determined a priori through simple integration operations without the need of tuning W_{hh}. This observation has two significant consequences: Firstly, CPGs do not have to be unrolled to be trained as the output is fully determined given the previous state and the new input. Secondly, eliminating W_{hh} has the additional

benefit of preventing gradient explosion or vanishing during training, Fig. 3b. As a result, CPGs can be framed as a stateless network on condition that the previous state is passed as an input of the system.

3 Evaluation

We evaluate our method on a classic RL benchmark: the hopping leg [17], which due its periodic task is a great fit for the application of CPGs. In fact, a single leg Fig. 1a needs only two joints to hop and this is the minimal configuration required by coupled Hopf-oscillators to express the complete form; less than two would cancel out the coupling terms [16].

We based the environment on a single leg of the ANYmal quadruped robot, which was fixed to a vertical slider. Its mass is 3.42 kg, it is actuated by two series-elastic actuators capable of 40 N m torque, a maximum joint velocity of $15\,\mathrm{rad\,s^{-1}}$ and controlled at 400 Hz. We use PyBullet [18] to simulate the system and use a data-driven method to capture the real system's actuator dynamics.

At every time-step the following observations are captured: the joints' measured positions p_j^m and velocities v_j^m, desired positions p_j^d, the position p_h and the velocity v_h of the hip attached to the rail. While the torques t_j^d and the planar velocity of the foot $v_f^{x,y}$ are instead used in computing the rewards, as described in the following. To train CPG-Actor, we formulate a reward function as the sum of five distinct terms, each of which focusing on different aspects of the desired system:

$$
\begin{aligned}
r_1 &= (1.2 \cdot \max(v_h, 0))^2 & r_4 &= \sum_J -1.e^{-4} \cdot (t_j^d)^2 \\
r_2 &= \sum_J -0.5e^{-2} \cdot (p_j^d - p_j^m)^2 & r_5 &= -1.e^{-2} \cdot \left\| v_f^{x,y} \right\| \\
r_3 &= \sum_J -1.e^{-3} \cdot (v_j^m)^2
\end{aligned}
\tag{3}
$$

where $_J$ stands for joints.

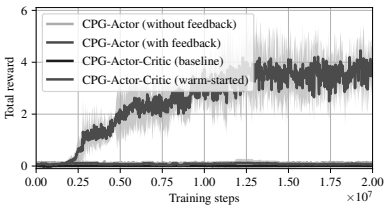
In particular, r_1 promotes vertical jumping, r_2 encourage the reduction of the error between the *desired position* and the *measured position*, r_3 and r_4 reduce respectively the *measured velocity* and the *desired torque* of the motors and finally, r_5 discourage the foot from slipping.

3.1 Experimental setup

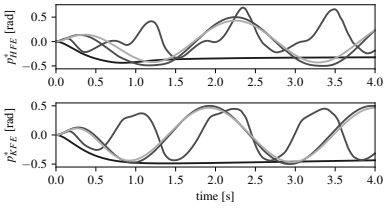
CPG-Actor is compared against [10] using the same environment. Both approaches resort to an actor-critic formulation, precisely running the same critic network with two hidden layers of 64 units each. Indeed, the main difference is the actor, which is described in detail in Sec. 2 for the CPG-Actor case, while [10] relies on a network with two hidden layers of 64 units each.

We trained the approaches for 20M time steps using an Nvidia Quadro M2200 GPU and an Intel(R) Xeon(R) E3-1505M v6 @ 3.00GHz CPU (8 cores) CPU; the process lasted roughly 2 hours.

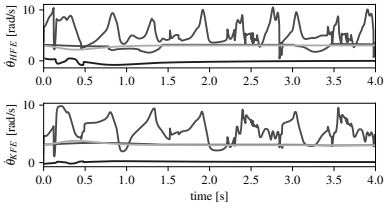
As Sec. 4 illustrates, an appropriate comparison between CPG-Actor and [10] required the latter to be warm-started to generate *desired positions* resulting in visible motions of the leg. Differently from the salamander [16], already tuned parameters are not available for the hopping task, hence a meaningful set from [9] was used as reference. The warm-starting consisted in training the actor network for 100 epochs in a supervised fashion using as target the aforementioned parameters.



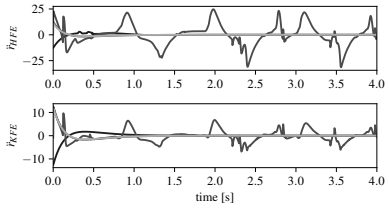
(a) Episode reward over 20M time steps horizon.



(b) Desired positions generated by CPG-Actor-Critic [10] and CPG-Actor.



(c) Comparison between $\dot{\theta}$, eq. (2), generated by CPG-Actor-Critic [10] and CPG-Actor.



(d) Comparison between $\ddot{r}$, eq. (2), generated by CPG-Actor-Critic [10] and CPG-Actor.

Figure 4: (4a) represents how the reward evolves during training, each approache run five times and averaging the rewards. (4b) trajectories generated by the different approaches: [10] warm-start produces an output similar to CPG-Actor without feedback. While CPG-Actor with feedback presents a heavily reshaped signal. The different contribution of the feedback in the two aforementioned approaches is explained by (4c) and (4d). The feedback – in CPG-Actor case – is interacting with the controller, resulting into visibly reshaped $\dot{\theta}$ and $\ddot{r}$ (green lines).

4 Results

4.1 CPG-Actor and previous baselines, comparison

The results of the comparison between CPG-Actor ans [10] can be seen in Fig. 1a. Although the warm-starting procedure results in a performance im-

provement for [10] (red line vs blue line), CPG-Actor (green line) achieves roughly a twenty times higher reward after 20 million training time-steps.

We investigated the reason of such different performances and we argue it lies in the way the feedback affects the CPG controller. Figures 4c and 4d represent the evolution over time of the CPGs. Observing $\dot{\theta}$ and $\ddot{r}$ in experiments with [10] it is evident they do not show responsiveness to the environment, since the blue and the red lines remain almost flat during the whole episode. On the other hand, $\dot{\theta}$ and $\ddot{r}$ in CPG-Actor experiments (green line) demonstrate substantial and roughly periodic modifications over time. Although [10] relies on feedback information to infer the CPGs dynamics, in practise the effects of the feedback signals on the shape of the output variables are rather weak when compared to CPG-Actor, as visible in Fig. 4b: in the case of CPG-Actor the original CPG's cosine output is heavily reshaped by the feedback, while [10] presents an almost-sinusoidal behaviour. Hence, to achieve successful hopping strong feedback information is crucial.

To further assess our intuition, we show CPG-Actor's open-loop (i.e. without feedback) behaviour (orange line), which shows performances on par with [10] after warm-start. Indeed, albeit explicitly penalised by eq. (3), both led to policies with the foot sliding on the floor and, as such, with low vertical velocity (yet slightly oscillating as if hopping); this behaviour results in low final rewards even after a large number of training episodes (20 M). It is then evident that the direct propagation of the gradient through a differentiable CPGs allows CPG-Actor to learn an effective correction to the open-loop behaviour through the sensor feedback.

4.2 Evaluation of progressive task achievement

The last set of experiments presented assess how CPGs' outputs and the overall behaviour evolve over the course of the learning. The plots in Fig. 1 present the system at 1, 20 and 50 million time-steps of training. Figure 1b, shows the progress of the hopper in learning to jump; indeed, the continuous and dotted lines – respectively indicating the hip and the foot position – start quite low at the beginning of the training, to almost double the height after 50 millions time-steps.

5 Discussion and Future work

We propose CPG-Actor, an effective and novel method to tune CPG controllers through gradient-based optimisation in a RL setting.

In this context, we showed how CPGs can directly be integrated as the Actor in an Actor-Critic formulation and additionally, we demonstrated how this method permits us to include highly non-linear feedback to reshape the oscillators' dynamics.

Our results on a locomotion task using a single-leg hopper demonstrated that explicitly using the CPG as an Actor rather than as part of the environment

results in a significant increase in the reward gained over time compared with previous approaches.

Finally, we demonstrated how our closed-loop CPG progressively improves the hopping behaviour relying only on basic reward functions.

In the future, we plan to extend the present approach to the full locomotion task by utilising the same architecture shown in Fig. 1a with a CPG-network made of 12 neurons in order to be able to control a quadruped robot with 12 DOFs.

Acknowledgements

This work was supported by the EPSRC grant 'Robust Legged Locomotion' [EP/S002383/1], the UKRI/EPSRC RAIN [EP/R026084/1] and ORCA [EP/R026173/1] Hubs and the EU H2020 Project MEMMO (780684). The Titan V used for this research was donated by the NVIDIA Corporation. This work was part of the Human-Machine Collaboration Programme, supported by a gift from Amazon Web Services.

References

1. C. D. Bellicoso, F. Jenelten, C. Gehring, and M. Hutter, "Dynamic locomotion through online nonlinear motion optimization for quadrupedal robots," *IEEE Robot. Autom. Lett.*, vol. 3, no. 3, pp. 2261–2268, July 2018.
2. J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," *Science Robotics*, vol. 5, no. 47, 2020. [Online]. Available: https://robotics.sciencemag.org/content/5/47/eabc5986
3. M. Hutter, C. Gehring, D. Jud, A. Lauber, C. D. Bellicoso, V. Tsounis, J. Hwangbo, K. Bodie, P. Fankhauser, M. Bloesch, R. Diethelm, S. Bachmann, A. Melzer, and M. Hoepflinger, "Anymal - a highly mobile and dynamic quadrupedal robot," in *Proc. IEEE/RSJ Int. Conf. Intell. Rob. Sys. (IROS)*, 2016, pp. 38–44.
4. L. Righetti and A. J. Ijspeert, "Pattern generators with sensory feedback for the control of quadruped locomotion," in *Proc. IEEE Int. Conf. Rob. Autom. (ICRA)*, May 2008, pp. 819–824.
5. 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, no. 26, 2019. [Online]. Available: https://robotics.sciencemag.org/content/4/26/eaau5872
6. A. J. Ijspeert, "Central pattern generators for locomotion control in animals and robots: A review," *Neural Networks*, vol. 21, no. 4, pp. 642 – 653, 2008.
7. S. Bonardi, M. Vespignani, R. Möckel, J. Van den Kieboom, S. Pouya, A. Spröwitz, and A. Ijspeert, "Automatic generation of reduced cpg control networks for locomotion of arbitrary modular robot structures," *Proc. Robotics: Science and Systems (RSS)*, 2014.
8. S. Gay, J. Santos-Victor, and A. Ijspeert, "Learning robot gait stability using neural networks as sensory feedback function for central pattern generators," in *Proc. IEEE/RSJ Int. Conf. Intell. Rob. Sys. (IROS)*, 2013, pp. 194–201.

9. M. Ajallooeian, S. Gay, A. Tuleu, A. Spröwitz, and A. J. Ijspeert, "Modular control of limit cycle locomotion over unperceived rough terrain," in *Proc. IEEE/RSJ Int. Conf. Intell. Rob. Sys. (IROS)*, Tokyo, 2013, pp. 3390–3397.
10. Y. Cho, S. Manzoor, and Y. Choi, "Adaptation to environmental change using reinforcement learning for robotic salamander," *Intell. Serv. Robot.*, vol. 12, no. 3, p. 209–218, Jul. 2019. [Online]. Available: https://doi.org/10.1007/s11370-019-00279-6
11. A. L. Ciancio, L. Zollo, E. Guglielmelli, D. Caligiore, and G. Baldassarre, "Hierarchical reinforcement learning and central pattern generators for modeling the development of rhythmic manipulation skills," in *2011 IEEE International Conference on Development and Learning (ICDL)*, vol. 2, 2011, pp. 1–8.
12. Y. Nakamura, T. Mori, M. aki Sato, and S. Ishii, "Reinforcement learning for a biped robot based on a cpg-actor-critic method," *Neural Networks*, vol. 20, no. 6, pp. 723 – 735, 2007.
13. S. Fukunaga, Y. Nakamura, K. Aso, and S. Ishii, "Reinforcement learning for a snake-like robot controlled by a central pattern generator," in *Proc. IEEE Conf. Rob., Aut. Mech. (ICMA)*, vol. 2, 2004, pp. 909–914 vol.2.
14. R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html
15. J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347
16. A. J. Ijspeert, A. Crespi, D. Ryczko, and J.-M. Cabelguen, "From swimming to walking with a salamander robot driven by a spinal cord model," *Science*, vol. 315, no. 5817, pp. 1416–1420, 2007.
17. G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," 2016. [Online]. Available: http://arxiv.org/abs/1606.01540
18. E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," http://pybullet.org, 2016–2020.