A decentralized approach to nature-inspired robot locomotion

L.G. Baracco

Vrije Universiteit Amsterdam,
De Boelelaan 1105, 1081 HV Amsterdam, Netherlands

Abstract. Locomotion stands as a fundamental requirement for diverse robot types. Nevertheless, prevailing solutions predominantly concentrate on optimizing the locomotive abilities of specific robot body types. In this paper, we present a novel joint-based model that draws inspiration from Central Pattern Generators (CPGs). Our model is designed to overcome the limitations of morphology-specific approaches and provide a decentralized solution, enabling locomotion regardless of the robot's shape.

Keywords: Evolutionary Algorithms · Neuroevolution · PGPE · ClipUp · Modular Neural Networks · Fourier Series · Structured Control Net· Modular robotics · Robot · Locomotion · Decentralized controller

1 Introduction

Embedded systems have shown to be extremely useful for an increasingly larger number of tasks (3). However, these systems require a solid baseline to perform the most essential tasks such as object retrieval or natural language processing. One of these essential tasks is locomotion.

Locomotion presents itself as a surprisingly intricate task that requires robots of different sizes and shapes to move in a both stable and efficient way (14). Due to the inherent complexity of this problem, previous studies tackled this issue using vastly different approaches. The most common approach is to use state-of-the-art reinforcement learning, rewarding the speed and energy usage trade-off of the robot (10). On the other hand, extensive research has been pursued in nature-inspired methods. For example, some researchers have imitated the brain's Central Pattern generators (CPGs) that allow the modulation of non-rhythmic input signals into a rhythmic and periodic output (12). CPGs have been simulated through the use of numerical integration methods like the Fehlberg method (4) which continuously integrates the previous output to produce a new one, or the use of sinusoidal waves (1), which receive non-periodic state information to generate a periodic output. Other nature-inspired solutions involved the use of evolutionary algorithms. A notable example is that of Geijtenbeek et al. (8) who employed cross-over, selection, and mutation mechanisms to optimize the walking gait of bipedal robots. Sehnke et al. (20) instead developed a variation on natural evolutionary strategies called PGPE specifically designed

for robot locomotion which aims to estimate the gradients with respect to the parameters of a neural network (natural gradients) over computing them with backpropagation.

An aspect often overlooked in these implementations is the inability to produce a model that successfully transfers to different body shapes. Most reinforcement learning and CPG-based models require a fixed input and a fixed output, limiting their applicability to robots with the same physical architecture (see **Figure 1**). A solution to this problem has been proposed by Huang et al. (11), who created a modular decentralized reinforcement learning policy trained on various body shapes. This model can not only successfully govern different body shapes exhibiting different gaits, but has also the ability to control new unforeseen shapes in a zero-shot manner. However, this attempt has not yet been emulated using nature-inspired methods.

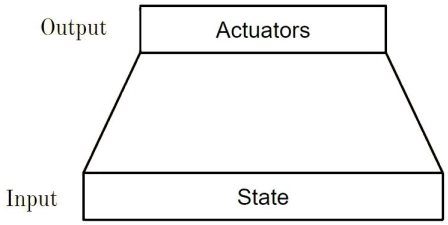


Fig. 1: **Centralized Locomotor Neural Network**
Standard architecture of a neural network employed in locomotion: the state of the entire body generates output for all the joints.

In this paper, we introduce DecLoco (Decentralized Locomotor), a modular decentralized architecture that uses nature-inspired methods. Specifically, we will develop a controller that governs a single joint. Each controller will receive the state information of the joint and of its closest neighboring joints. This state will be processed by a neural network architecture that includes a Fourier series to ensure periodicity, and will finally output the desired angle of the single joint. This decentralized setup allows each joint to be controlled independently while having limited knowledge of the state of other joints within its proximity. In addition, the optimization of this network will be performed using the aforementioned PGPE. The assumption behind the architecture is that solutions that turn linear information into rhythmic output do not require global coordination, as the motion is ensured by local periodic gaits.

The goal of this research is to assess whether nature-inspired methods that perform successfully on distributed controllers can generalize in a decentralized environment. The underlying hypothesis is that centralized coordination is less crucial for achieving rhythmic movement, and that ensuring local periodic gaits is sufficient for facilitating efficient locomotion, irrespective of the robot's body

shape. By investigating this assumption, we aim to shed light on the feasibility and effectiveness of employing nature-inspired approaches in decentralized locomotion systems.

We are, however, not interested in assessing the global performance of our solution. The goal of our decentralized model is not to compete with state-of-the-art centralized models, but to provide a model that produces reliable locomotion across various body shapes.

The results indicate that DecLoco has the ability to generalize across different shapes. In addition, each joint's controller does not require neighbors' state information, as its removal does not impact the performance of the model. In other words, the joint's own state information is enough to produce stable periodic gaits.

2 Methods

This section outlines the methodology employed for the optimization of virtual robot controllers using a variant of Natural Evolution Strategies (NES). The objective is to develop decentralized controllers, where each joint of the virtual robot is controlled by its own neural network modules. We will illustrate the mechanism of NES, and the specific algorithm used to solve this particular problem. Furthermore, we will illustrate how these neural networks utilize a topology designed for locomotion inspired by CPGs to ensure stable periodic gaits. In conclusion, an explanation of the data structure representing the robot bodies and of the message-passing mechanism used to increase coordination is given.

2.1 Natural Evolution Strategies

The optimization of the controller's neural networks is performed using a variant of NES (26). NES are a set of optimization techniques aimed at estimating the gradient of an objective function, instead of computing it directly. The starting point of NES is similar to that of other evolutionary algorithms (2), where a population of solutions is generated and then iteratively regenerated with the goal of maximizing or minimizing a fitness function. The key difference between NES and other evolutionary techniques is that the population is generated with a distribution over the parameter space. In each generation, the distribution is biased toward the expected increase or decrease of the objective function, thus approximating the gradient of the objective function with respect to the parameter space. The use of these natural gradients over gradient descent or ascent provides better scalability toward large parameter space and more resilience in fuzzy environments where small perturbations of the parameters can lead to unpredictably small or large changes in the objective function.

2.2 PGPE and ClipUp optimization

The variant of natural evolution strategies used is called Policy Gradients with Parameter-based Exploration, also known as PGPE (20). PGPE is an algorithm specifically designed for neuroevolution (15) of Neural Networks employed in reinforcement learning tasks.

PGPE does not involve a model of the environment and its rewards over the action space like other reinforcement learning techniques (25) but rather maps the current state of the policy and of the environment to the best possible action. As illustrated by Sehnke et al. (20), This proves particularly advantageous in continuous Markovian environments where the state s_t of an environment is fully dependent on the previous state s_{t-1} and its corresponding taken action a_{t-1}. This scenario commonly occurs in robot locomotion tasks within obstacle-free environments.

As previously mentioned, PGPE operates iteratively in an evolutionary manner. The algorithm starts by taking the center solution (the center is randomly defined in the first generation) and sampling a population of new solutions normally distributed with the center solution as the mean. For each solution $\mu + \epsilon$, a mirror solution $\mu - \epsilon$ is computed and vice versa. For each pair, a direction of improvement over the fitness can be derived. A weighted average of the fitness directions is then computed, which will point to the center distribution of the following iteration.

In our implementation, the PGPE algorithm is used along with the ClipUp (clipped updates) optimizer (24). ClipUp makes gradient optimization more dynamic through gradient normalization and parameter update clipping. First, The normalization of the gradients is done by the population size:

$$step = \frac{\nabla f(\theta)}{||\nabla f(\theta)||}$$

Parameter update is based on heavy ball momentum (18):

$$v_{k+1} = m \cdot v_k + \alpha \cdot step$$

Where the momentum m and the step size α are fixed hyperparameters and $v_0 = 0$. Opposed to classical heavy ball momentum, ClipUp incorporates an additional v_{max} parameter that allows optima to be exploited in case v_{k+1} is higher than v_{max}:

$$v_{k+1} = v_{max} \cdot \frac{v_{k+1}}{||v_{k+1}||}$$

ClipUp has been used because it seems that adaptive optimizers perform significantly better when employed in distribution-based techniques such as PGPE (19).

2.3 Structured Control Net

The PGPE algorithm has been used to optimize the parameter of a Structured Control Net with sinusoidal waves (21). This network is made of two components: a linear component and a nonlinear component (see **Figure 2**).

The linear component consists of a state layer fully connected to a single output layer whose parameters are the weights and biases of the connections.

The non-linear component consists of an input-independent Fourier series comprising an arbitrary number of sinusoidal waves whose parameters are the amplitude, frequency, and phase of each. The varying input of each wave is the elapsed time, which increases by 1 for each call of the network:

$$Out_{sin} = A \cdot sin(\omega \cdot t + \phi)$$

The final output is the total sum:

$$Out_{total} = Out_{linear} + \sum_{i=1}^{n} Out_{sin_i}$$

The selection of sinusoidal waves is motivated by the objective of ensuring periodicity, which aligns well with locomotion tasks in Markovian environments (16) inspired by natural human locomotion (7).

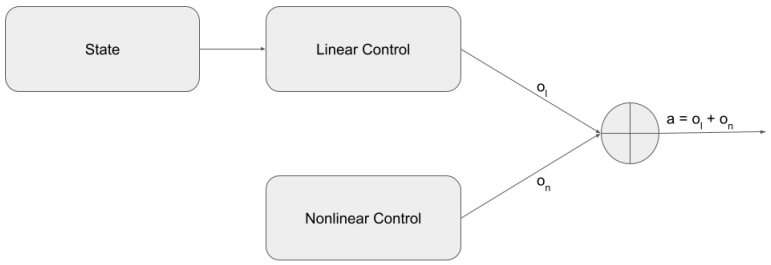


Fig. 2: **Structured control net**
A structure control net as proposed by Srouji et al. (21). Here the encoding is the state of the single joint concatenated to the state of its neighbors and the Nonlinear control is the Fourier series.

2.4 Robot structure

Every robot shape is formed of these elements:

- *Core*: initial element. Every robot body has to have one.
- *Joint*: connection between two rigid bodies.
- *Brick*: solid block of fixed size.
- *Active hinge*: thin block always connected to a joint
- *Rigid body*: solid part of the body. A rigid body is made of zero o more bricks and at least an active hinge or the core.

These distinctions allow the creation of robot shapes of various complexities, although each controller will only be interfaced with joints and rigid bodies. The final body is encoded in a tree shape, where the rigid body corresponding to the core serves as the root node. The root node is then connected to joints, which in turn connect to other rigid bodies. This structure continues until reaching the leaf nodes, which will always be rigid bodies.

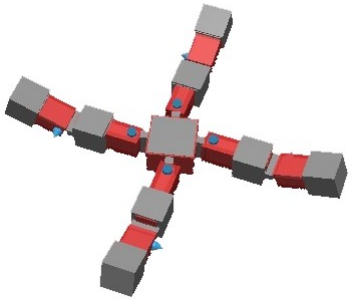


Fig. 3: **Modular robot body structure**
Image of a Spider robot body. The blue arrows represent the joint and their allowed direction, the parts in grey are rigid bodies corresponding to one or more bricks, the parts highlighted in red are rigid bodies corresponding to active hinges, and the core is the bigger body in the center.

2.5 Decentralized controller architecture

The proposed architecture maps a state vector into a single output. Each joint receives state information about itself and its corresponding lower-connected body. The state information includes the position vector $[x, y, z]$ (relative to the starting point) and the orientation quaternion $[x, y, z, w]$. In addition, the state vector of n neighbors is concatenated (see **Figure 4)**. The final input vector of each controller thus looks like this:

$$S_{total} = S_{own} + \sum_{i=1}^{n} S_i$$

The neighbors are scouted by first climbing down the tree and then back up. In case the robot body has fewer joints than the established number of neighbors, the closest neighbors' values will be duplicated.

The output is the single degree of freedom of the joint I.E. the new desired angle. The outputs of all the joints are then concatenated into the new degree of freedom state.

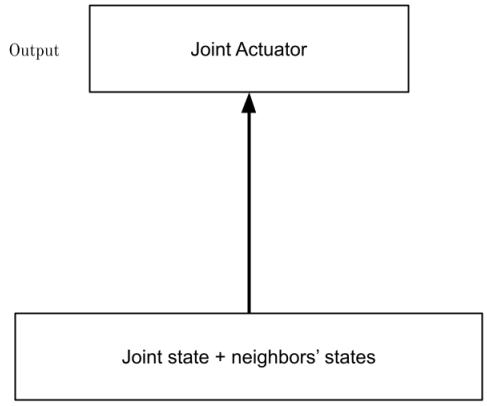


Fig. 4: **Architecture of decentralized controller**
An overview of the architecture of each single joint. It is possible to see that the joint's own state information is concatenated to that of the neighbors, but only one output is produced.

3 Experimental setup

We will now present the key components and procedures involved in the experiment, including the programming and simulation environment, the fitness function used for the evaluation of each generation, the parameter tuning, and the evaluation metrics.

3.1 Environment

The entire program has been developed in Python 3.8[1]. Different software has been used for the simulation of the physics environment and the implementation of the optimization algorithms.

Revolve2 Revolve2 (5) is an optimization and simulation library that provides a unified interface for different physics simulators and physical controllers. In this study, the library has been used to easily interface the MuJoCo physics simulator (22) and to gather state information about every single simulation to provide results and statistics.

Revolve2 also provides several standard body shapes that have been used in this study. Some of these shapes have been employed only in the testing stage to verify how well the architecture can zero-shot generalize to body shapes that have never been used during the optimization stage. An overview of the body shapes used can be seen in **Table 1**, and a visualization in **Figure 5**.

[1] The repository is publicly available at `https://github.com/theFosu/Decentralized_Controller`

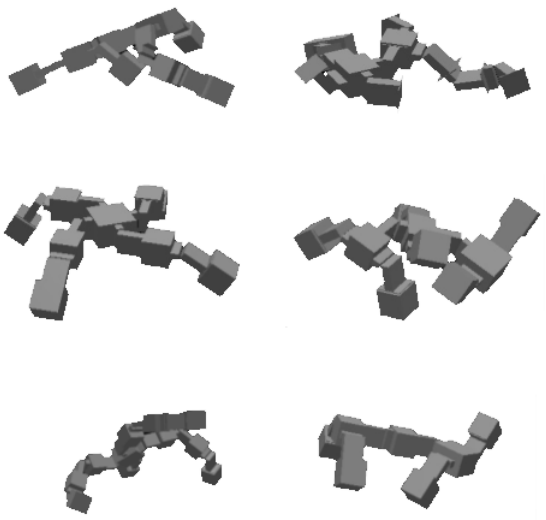


Fig. 5: **Robot bodies**
From left to right and top to bottom: Babya, Insect, Spider, Ant, Babyb, and gecko

Table 1: **Robot bodies**
Robot bodies used with their corresponding numbers of active hinges and bricks, and their description. The word "limb" is used here to describe a set of bodies and joints that extend from a "torso" made of the core and, optionally, other bodies and joints.

Body	**Active Hinges**	**Bricks**	**Description**
Train shapes			
Babya	8	7	2 asymmetric frontal limbs with respectively 1 and 3 joints each and 2 shorter rear limbs with 1 joint each.
Insect	9	2	Complex shape with 2 longer limbs and 3 active hinges with no Brick at the bottom.
Spider	8	8	Fully symmetrical body with 4 limbs.
Ant	8	8	3 pairs of limbs with 2 joints each.
Test shapes			
Babyb	10	10	3 limbs with 3 joints each and a limb with a single joint.
Gecko	6	6	2 pairs of limbs with 1 joint each.

Evotorch Evotorch (23) is a comprehensive library that builds on top of PyTorch (17) which features an implementation of several evolutionary optimization algorithms. PGPE and ClipUp are adopted as-is from the library.

3.2 Fitness

Each solution undergoes evaluation across multiple body shapes, and the final fitness is determined by selecting the lowest fitness value achieved among all the corresponding body shapes. This approach ensures the selection of a solution that exhibits the best generalization across different body structures.

The fitness of each robot and solution pair is calculated At the end of each simulation. The position of the robot denoted as $coor_{t(n)}$ with *coor* representing x, y, or z, is recorded at each time-step $t(n)$ (with $t(-1)$ being the final time-step), along with the corresponding action vector A describing the amplitude of joint movement. It is possible to compute the fitness:

$$F = 3 \cdot (x_{t(0)} - x_{t(-1)})^2 - \sum_{i=1}^{m} \frac{a_i^2}{n_J} - \sigma_A$$

The formula is divided into three parts. The first part simply optimizes the maximum distance traveled on the x axis. Movement on the y and z axis are not used in order to favor straight walking gaits. The second part is a penalty for large action amplitude. The need for this part of the formula arises to discourage unrealistic exploitation of the simulator, where very quick and wide movements can be made to propel the robot forward. The value is normalized by the number of joints since the same formula has been used for all robots. In addition, in case this part of the formula results in 0, a penalty of -0.5 is given to prevent the robot from ever not performing any action. The third part penalized asymmetric behavior, that is to avoid the robots not exploiting each of their limbs to move forward.

3.3 Parameters

There are three sets of hyperparameters that need to be tuned: Evolutionary, Network, and Optimizer. A detailed overview of the parameters can be found in **Table 2**.

The evolutionary hyperparameters are parameters that involve the simulations and the iterations of the optimizer. A study by Lan et al. (13) on other techniques applied in the library preceding Revolve2 showed that it might be unnecessary to run simulations as long as 10 seconds. In addition, the sampling and control frequencies have been chosen to approximate the arbitrary time-step of the simulator. Finally, the number of generations and the population size are rather small due to the simplicity of the architecture and the complexity of the fitness function, as higher values pose the risk of overfitting towards a specific part of the formula.

The network hyperparameters are parameters that modulate the size and the topology of the controlling neural network. The number of sinusoids is based on the work of Srouji et al. (21) who first implemented SCNs. The number of neighbors has been chosen to involve approximately less than half of the average robot body with the assumption that a symmetric behavior could arise.

Finally, the Optimizer hyperparameters are the parameters required by PGPE and ClipUp. These parameters seem to be the most optimal as indicated by Gomez et al. (9).

Table 2: **Hyperparameters**
List of parameters. Evolution Hyperparameters involve the parameters common to all evolutionary algorithms, Network Hyperparameters are the parameters needed to instantiate the networks to optimize, and the Optimizer hyperparameters are the parameters required by PGPE and ClipUp. All the parameters not mentioned here are left as the default of the employed library.

Parameter	**Value**
Evolution Hyperparameters	
Simulation time (seconds)	6
Sampling Frequency	60
Control Frequency	60
Simulation time-step (seconds)	0.001
Population size	254
Generations	400
Network Hyperparameters	
Number of neighbors	4
Number of sinusoids	16
Optimizer hyperparameters	
σ_0	2.5
v_{max}	$\frac{\sigma_0}{15}$
α	$\frac{v_{max}}{2}$
m	0.9

3.4 Evaluation

The model designed will first be evaluated by its fitness improvements. Since the fitness formula is divided into three parts (see **Section 3.2**), the number of generations needed for convergence will be identified in order to prevent overfitting.

The generalization of the model will be tested by noting the difference in fitness between each of the training and test bodies. A short effect size of fitness between bodies will indicate high generalization. Additionally, Each body will be tested 60 times to assess whether the controller exhibits non-deterministic behavior.

Ablations Ablations will be performed in order to check whether all parts of the architecture are necessary to obtain the same results. The ablations will be confronted by their fitness curve.

The first ablation (SCN-0) will consist of a model with the same architecture but without the use of neighbors. At each time-step every network will thus only receive the information of the joint and corresponding rigid body controlled.

The other ablation (MLP-4) will consist of a simple FeedForward network with neighbors and 1 hidden layer. The number of nodes in the hidden layer is 16, the same as the number of sinusoids in the other architectures.

4 Results

In this research, we raised the question of what the point of convergence of fitness evaluation for our model would be. **Figure 6** shows that the fitness tends to converge rather quickly after 100 generations, as the same best solution is kept for more than 100 more generations. We thus consider the best solution of a model as the best solution found in the 150th generation, in order to let slower converging searches to also find an optimum.

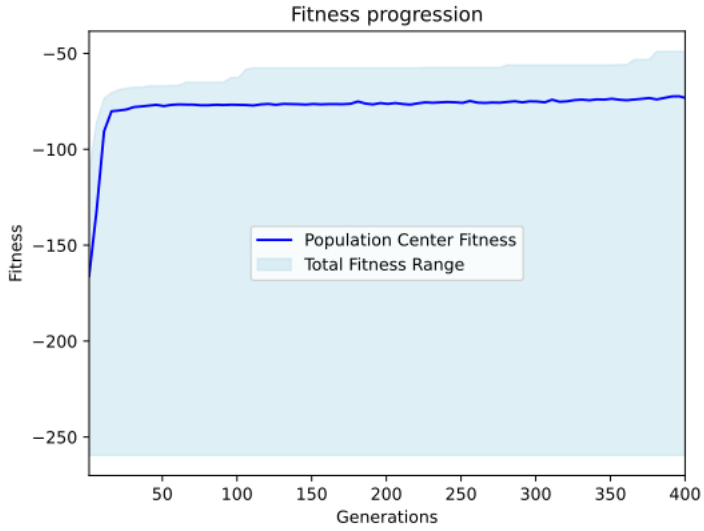


Fig. 6: **DecLoco fitness kinetics**
Fitness kinetics of our proposed architecture DecLoco. The main curve indicates the median fitness of the generation, while the range indicates the best and worst solutions ever found.

4.1 Generalization and Determinism

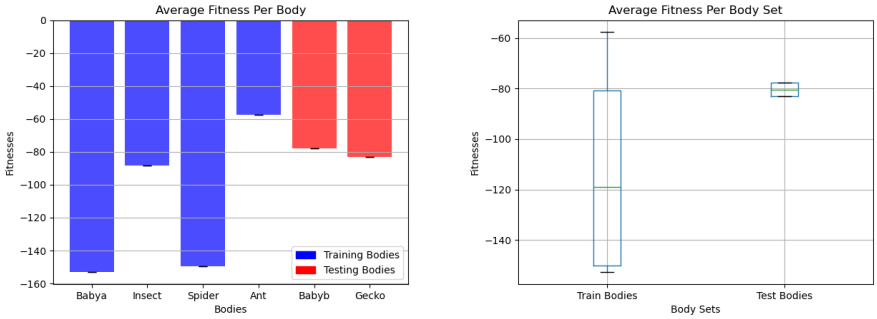
We decided to test the generalization of the solution by comparing the fitness value returned when tested over each single body, and when splitting the results between bodies used during training and new unseen bodies. Furthermore, the same test can give additional insights into whether the solutions exhibit deterministic or stochastic behavior by running each simulation multiple times and computing the deviation.

Figure 7(a) shows that the deviation between bodies is rather large. In addition, both the test bodies seem to perform better than Insect and Spider. In particular, a comparison of the boxplots (see **Figure 7(b)**) shows that the solution does not always perform better with bodies used during training.

An additional note on **Figure 7(a)** is that the deviation of fitness among simulations over the same body is 0. This indicates that the solution is stable and deterministic.

4.2 Ablations

We will now assess the importance of different elements of the architecture through the comparison with ablations over the complete architecture. **Figure 8** shows the fitness curves of the various models. Our main model DecLoco seems to perform equally as well as SCN-0, albeit with faster convergence. MLP-4 performs largely worse than the other model but converges rather quickly, suggesting a relatively small solution space. The difference in sizes of the solutions



(a) Single Body Comparison (b) Test/Train Body set Comparison

Fig. 7: **Comparison of fitness evaluations**
(a) A bar chart indicating the performance of DecLoco on each individual model. (b) A comparison of boxplots representing respectively the bodies used to optimize the model and new bodies excluded from the optimization process. It is possible to see that the deviation between bodies is rather large and seems to be smaller on the test set. On the other hand, the deviation within each set of simulations using the same body (the error bars) is 0.

spaces can also be noted by the reduced extension of the total range of fitnesses. However, it is noteworthy that SCN-0 has a larger fitness range than DecLoco, even though the parameter space to optimize is smaller.

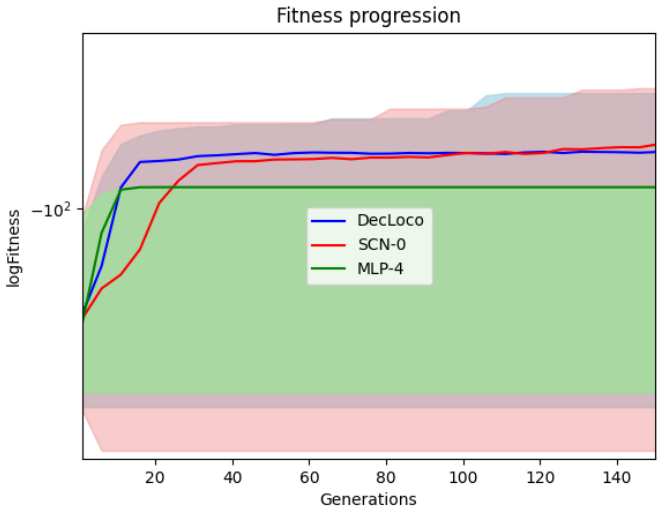


Fig. 8: **Fitness Comparisons**
Fitness kinetics of DecLoco compared to SCN-0 and MLP-4. While DecLoco and SCN-0 have similar curves, MLP-4 seems to converge more steeply into a worse solution. log scale is used to better visualize the differences.

5 Discussion and Conclusion

In this research, we developed a decentralized controller called DecLoco using nature-inspired methods for robot locomotion. This controller was influenced by previous implementations of nature-inspired robot controllers (1; 20), but with an eye to the need of developing a decentralized solution that would generalize across different robot topologies. The core assumption was that rhythmic movement is not based on distributed coordination, but rather on decentralized periodic gaits. The use of a Structured control net with a Fourier series that controls single joints has been assumed to be the most indicative representation of local periodicity.

As shown earlier, the proposed solution showed a rather quick and stable convergence. This result hints at the possibility to use the optimizer and the environment for higher-scale architectures and problems.

We developed a solution that showed very positive results regarding generalization, as illustrated in **Section 4.1**. Both the test bodies performed better than some of the training bodies employed. Concretely, this result might indicate the suitability of a fully decentralized solution when nature-inspired methods are used.

The suitability of fully decentralized solutions is further suggested by the results illustrated in **Section 4.2**. We could here notice that the inclusion of neighboring information had nearly no impact on the final performance of the solution. The negligible results of the addition of neighbors suggest that message-passing mechanisms might not be suitable for nature-inspired solutions influenced by CPGs such as ours. We speculate that the best performance could be obtained by a controller receiving both local information about the single joint, and fully global information such as the position of the center of mass of the whole body, the number of joints, the total weight, and so on.

The aim of this paper is to initiate a new line of research. Similar solutions could be applied to a more vast array of robot shapes, such as wheeled or biped shapes. Additionally, As mentioned in the introduction, a system of sinusoidal waves is only one of many techniques used to simulate CPGs in distributed solutions. Further research could explore decentralized implementations of others of these techniques. One of these implementations could make use of the Fehlberg method (4). This method could be applied to the locomotion of robots whose joints have several degrees of freedom. Another proposal could involve the use of time recurrent neural networks (6). This method could be applied in more complex non-Markovian environments, where the following action can dramatically change when small variations in the input state happen.

6 Acknowledgements

I thank Anil Yaman for his time dedicated to supervising this project on behalf of the university and for his extensive help with technical difficulties. I also want to thank Aart Stuurman for his elucidations regarding the use of Revolve2. I finally want to thank the entirety of the Computational Intelligence group for their availability in providing the computational resources to make this project possible and Karen Chiang for her help with the setup.

Bibliography

[1] Aguilar, J., Zhang, T., Qian, F., Kingsbury, M., McInroe, B., Mazouchova, N., Li, C., Maladen, R., Gong, C., Travers, M., et al.: A review on locomotion robophysics: the study of movement at the intersection of robotics, soft matter and dynamical systems. Reports on Progress in Physics **79**(11), 110001 (2016)

[2] Bäck, T., Schwefel, H.P.: An overview of evolutionary algorithms for parameter optimization. Evolutionary computation **1**(1), 1–23 (1993)

[3] Brogårdh, T.: Present and future robot control development—an industrial perspective. Annual Reviews in Control **31**(1), 69–79 (2007)

[4] Chen, R.T., Rubanova, Y., Bettencourt, J., Duvenaud, D.K.: Neural ordinary differential equations. Advances in neural information processing systems **31** (2018)

[5] CI-Group: Revolve2

[6] Dzeladini, F., Ait-Bouziad, N., Ijspeert, A.: Cpg-based control of humanoid robot locomotion. Humanoid Robotics: A Reference pp. 1–35 (2018)

[7] Fujii, K., Takeishi, N., Kibushi, B., Kouzaki, M., Kawahara, Y.: Data-driven spectral analysis for coordinative structures in periodic human locomotion. Scientific reports **9**(1), 16755 (2019)

[8] Geijtenbeek, T., Van De Panne, M., Van Der Stappen, A.F.: Flexible muscle-based locomotion for bipedal creatures. ACM Transactions on Graphics (TOG) **32**(6), 1–11 (2013)

[9] Gomez, F., Schmidhuber, J., Miikkulainen, R., Mitchell, M.: Accelerated neural evolution through cooperatively coevolved synapses. Journal of Machine Learning Research **9**(5) (2008)

[10] Haarnoja, T., Ha, S., Zhou, A., Tan, J., Tucker, G., Levine, S.: Learning to walk via deep reinforcement learning. arXiv preprint arXiv:1812.11103 (2018)

[11] Huang, W., Mordatch, I., Pathak, D.: One policy to control them all: Shared modular policies for agent-agnostic control. In: International Conference on Machine Learning. pp. 4455–4464. PMLR (2020)

[12] Ijspeert, A.J.: Central pattern generators for locomotion control in animals and robots: a review. Neural networks **21**(4), 642–653 (2008)

[13] Lan, G., Tomczak, J.M., Roijers, D.M., Eiben, A.: Time efficiency in optimization with a bayesian-evolutionary algorithm. Swarm and Evolutionary Computation **69**, 100970 (2022)

[14] McGhee, R.B.: Robot locomotion. Neural control of locomotion pp. 237–264 (1976)

[15] Moriarty, D.E., Mikkulainen, R.: Efficient reinforcement learning through symbiotic evolution. Machine learning **22**(1-3), 11–32 (1996)

[16] Morris, B., Grizzle, J.W.: Hybrid invariant manifolds in systems with impulse effects with application to periodic locomotion in bipedal robots. IEEE Transactions on Automatic Control **54**(8), 1751–1764 (2009)

[17] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al.: Pytorch: An imperative style, high-performance deep learning library. Advances in neural information processing systems **32** (2019)
[18] Polyak, B.T.: Some methods of speeding up the convergence of iteration methods. Ussr computational mathematics and mathematical physics **4**(5), 1–17 (1964)
[19] Salimans, T., Ho, J., Chen, X., Sidor, S., Sutskever, I.: Evolution strategies as a scalable alternative to reinforcement learning. arXiv preprint arXiv:1703.03864 (2017)
[20] Sehnke, F., Osendorfer, C., Rückstieß, T., Graves, A., Peters, J., Schmidhuber, J.: Parameter-exploring policy gradients. Neural Networks **23**(4), 551–559 (2010)
[21] Srouji, M., Zhang, J., Salakhutdinov, R.: Structured control nets for deep reinforcement learning. In: International Conference on Machine Learning. pp. 4742–4751. PMLR (2018)
[22] Todorov, E., Erez, T., Tassa, Y.: Mujoco: A physics engine for model-based control. In: 2012 IEEE/RSJ international conference on intelligent robots and systems. pp. 5026–5033. IEEE (2012)
[23] Toklu, N.E., Atkinson, T., Micka, V., Liskowski, P., Srivastava, R.K.: EvoTorch: Scalable evolutionary computation in Python. arXiv preprint (2023), https://arxiv.org/abs/2302.12600
[24] Toklu, N.E., Liskowski, P., Srivastava, R.K.: Clipup: a simple and powerful optimizer for distribution-based policy evolution. In: Parallel Problem Solving from Nature–PPSN XVI: 16th International Conference, PPSN 2020, Leiden, The Netherlands, September 5-9, 2020, Proceedings, Part II. pp. 515–527. Springer (2020)
[25] Wiering, M.A., Van Otterlo, M.: Reinforcement learning. Adaptation, learning, and optimization **12**(3), 729 (2012)
[26] Wierstra, D., Schaul, T., Glasmachers, T., Sun, Y., Peters, J., Schmidhuber, J.: Natural evolution strategies. The Journal of Machine Learning Research **15**(1), 949–980 (2014)