Supplement for 'Decentralized Deep Reinforcement Learning for a Distributed and Adaptive Locomotion Controller of a Hexapod Robot'

Malte Schilling[1], Kai Konen[1], Frank W. Ohl[2], and Timo Korthals[3]

Overview

This report provides supplementary information for the article 'Decentralized Deep Reinforcement Learning for a Distributed and Adaptive Locomotion Controller of a Hexapod Robot'. It contains detailed results for the different experiments described in the paper and gives information on where to find the source code as well as how to run the provided docker image and the experiments.

Correction: Note, in the original submission there was one typo in the evaluation of the performance for one seed. Therefore the numbers and figures changed slightly. In detail: p-value for comparison was written as 0.053 but is corrected to 0.051, point in time when the decentralized approach reaches the level of the centralized approach was given as epoch 2229, but is corrected to already epoch 1998. Furthermore, figure five and six from the original paper were updated.

I. Performance after Learning on Flat Terrain

In the first experiment, controllers were trained on flat terrain for 5000 epochs. Initially, we trained 10 controllers of each type. As the result showed a trend towards better performance of the decentralized approach, we are currently increasing the number towards 15 controllers for each approach (training takes about three days per controller; we will update the results later-on for the fifteen controller, an eleventh controller is added in the list as this already was sufficient to come up with a significant result).

Detailed results for all training runs are given in table I (for the decentralized architecture) and table II (for the centralized/baseline approach)

TABLE I: Detailed results after learning for 5000 epochs. The mean rewards were evaluated for each of the ten learned controllers for the decentralized architectures in 100 simulation runs after training. Note that standard deviation is taken for the single controller over the 100 repetitions which showed to be low.

Seed (Architecture)	Avg. Reward	Std. Dev. (btw. runs)	Overall Rank
Seed 1, Decentralized	764.209	26.013	2.
Seed 2, Decentralized	707.283	33.836	5.
Seed 3, Decentralized	570.693	39.562	13.
Seed 4, Decentralized	681.352	26.103	7.
Seed 5, Decentralized	756.096	27.497	3.
Seed 6, Decentralized	695.860	24.942	6.
Seed 7, Decentralized	733.194	56.818	4.
Seed 8, Decentralized	575.666	48.789	12.
Seed 9, Decentralized	598.666	30.161	10.
Seed 10, Decentralized	446.231	102.808	18.
Mean performance	652.925		
Std. dev. over controller	96.683		
Seed 11, Decentralized	638.920	38.650	/

For statistical analysis, we compared the two architectures using a two-tailed Welch t-test (see [1]), applying the function from the scipy-stats package (`ttest_ind_from_stats`) which returned a p-value of 0.051. Including one further seed for both architectures, mean values and standard deviation didn't change too much (decentralized architecture mean: 651.652 (std. dev. 92.271); centralized architecture mean: 527.988 (std. dev. 145.738)). Welch t-test produced a significant result, showing that the decentralized architecture performed significantly better ($p - value = 0.028$). The relative effect size was 0.94 which indicates a large effect size.

[1]Malte Schilling and Kai Konen are with the Neuroinformatics Group, Bielefeld University, 33501 Bielefeld, Germany mschilli@techfak.uni-bielefeld.de

[2]Frank W. Ohl is with the Department of Systems Physiology of Learning, Leibniz Institute for Neurobiology and with the Institute of Biology, Otto-von-Guericke University, Magdeburg, Germany.

[3]Timo Korthals is with the Kognitronik and Sensorik Group, Bielefeld University, 33501 Bielefeld, Germany.

TABLE II: Detailed results after learning for 5000 epochs. The mean rewards were evaluated for each of the ten learned controllers for the centralized (baseline) architectures in 100 simulation runs after training. Note that standard deviation is taken for the single controller over the 100 repetitions which showed to be low.

Seed (Architecture)	Avg. Reward	Std. Dev. (btw. runs)	Overall Rank
Seed 1, Central/Baseline	504.095	26.807	17.
Seed 2, Central/Baseline	770.158	21.520	1.
Seed 3, Central/Baseline	225.315	86.057	20.
Seed 4, Central/Baseline	566.485	13.287	14.
Seed 5, Central/Baseline	652.852	18.281	8.
Seed 6, Central/Baseline	302.733	5.476	19.
Seed 7, Central/Baseline	565.406	25.679	15.
Seed 8, Central/Baseline	600.676	23.579	9.
Seed 9, Central/Baseline	592.450	21.059	11.
Seed 10, Central/Baseline	557.175	27.348	16.
Mean performance	533.734		
Std. dev. over controller	151.658		
Seed 11,	470.521	16.683	/

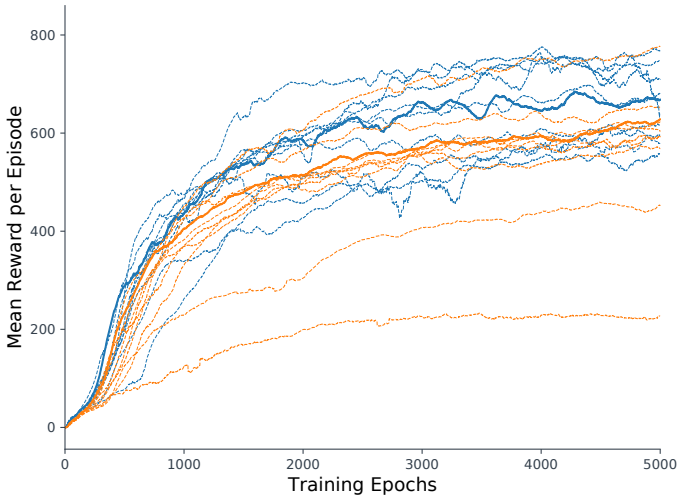

Fig. 1: Showing performance during training on flat terrain for individual seeds for 5000 epochs. Performance is measured as mean reward per episode (calculated as a running average over the last 100 episode in order to smoothen results). Results are given for the ten decentralized controllers (blue) and for the baseline centralized approach is shown in orange. Dashed lines show individual seeds and medians are shown as continuous line.

As the results showed a large variance, we compared as well the best performing controllers. Overall, the standard deviation in both conditions were in a similar range, but the distribution of the centralized approaches appeared larger. While the best performing approach was a centralized approach, there, first, seems to be an upper limit for maximum mean velocity and multiple approaches converged towards this performance. Overall, we found more decentralized architectures in the best performing approaches, but a rank-sum test could only show a trend and was not significant (Wilcoxon–Mann–Whitney test, $p = .063$).

II. Learning on Flat Terrain

The learned architectures were trained for 5000 epochs. Learning curves for the individual seeds are shown in Fig. 1.

Seeds were only measured up to 5000 epochs points as learning appears to have converged by then: Initially, we run simulations for up to 10000 epochs, but these only showed minor further improvements. There appears to be a maximum reachable velocity, Fig. 2.

Mean performance (shown in Fig. 3) was calculated over the ten seeds for each of the two architectures over training. Looking at individual runs showed a high variance during learning. The decentralized architecture learned significantly faster

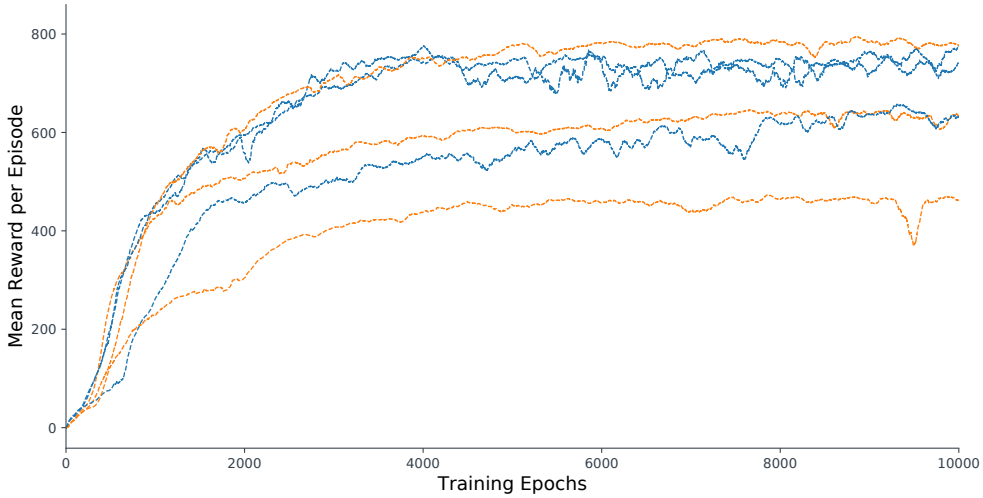

Fig. 2: Showing performance during training on flat terrain for individual seeds for 10000 epochs.. Performance is measured as mean reward per episode (calculated as a running average over the last 100 episode in order to smoothen results). Results are given for the ten decentralized controllers (blue) and for the baseline centralized approach is shown in orange.

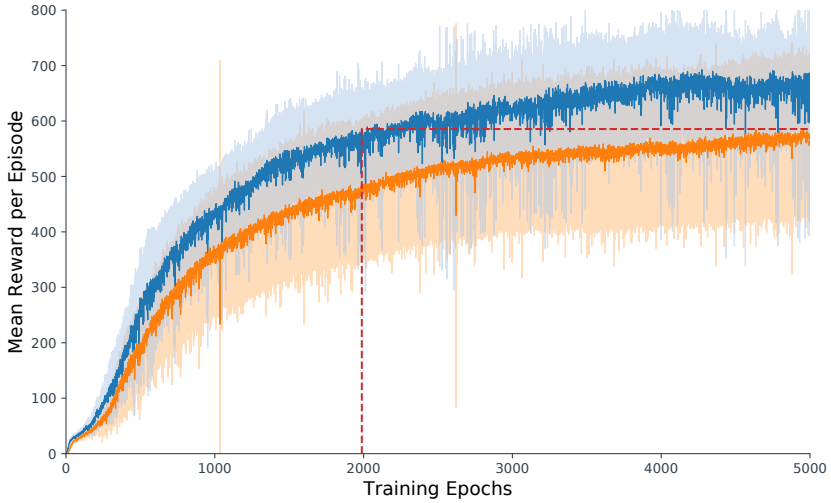

Fig. 3: Comparison of the mean reward during training: mean performance over the ten decentralized controllers is shown in blue and mean reward for the baseline centralized approach is shown in orange (shaded areas show standard deviation). Performance is measured as reward per episode. Seeds were only measured up to 5000 epochs points as learning appears to have converged by then. The horizontal red dashed line visualizes the maximum of the centralized approach at the end of training. The vertical red dashed line indicates when this performance level was reached by the decentralized approach (after 1988 epochs already).

and reached the maximum performance of the centralized controller much earlier. The horizontal red dashed line visualizes the maximum of the centralized approach after 5000 epochs. The vertical red dashed line indicates when this performance level was reached by the decentralized approach (after 1988 epochs already).

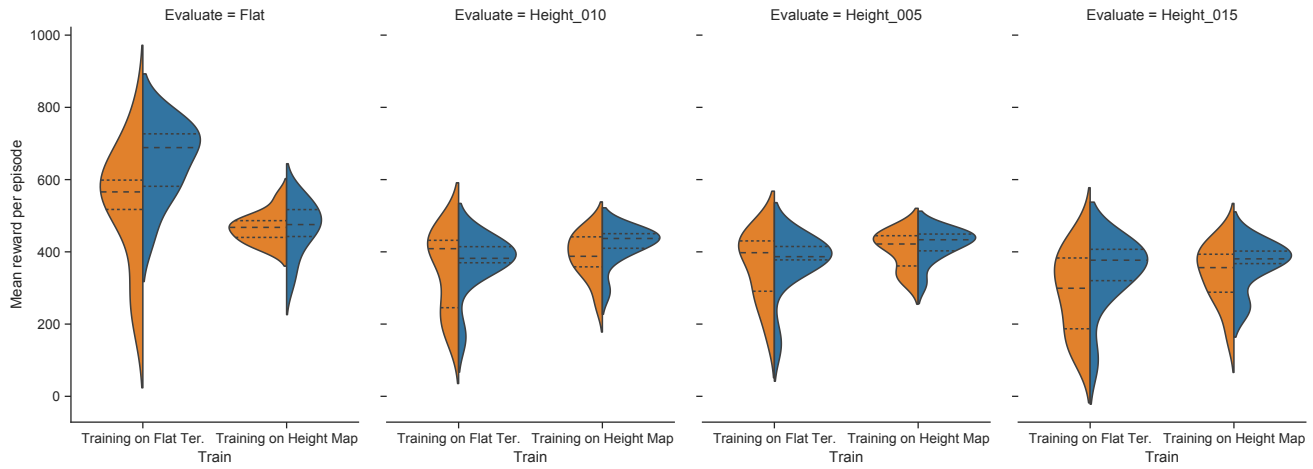

Fig. 4: Comparison of the (approximated) performance distribution for the controllers. Shown are violin plots for four evaluation conditions, from left to right: evaluation of walking on flat terrain; evaluation on three different height maps of maximum height of $0.05\,\mathrm{m}$, $0.10\,\mathrm{m}$, and $0.15\,\mathrm{m}$. Inside the four evaluations, it is further distinguished between controllers trained on the flat terrain (left side) and controllers trained on the height map (right side). The violin plots show to approximated distributions (and the quartiles are given inside): orange shows the centralized baseline approach and blue shows the decentralized architecture. Performance for the ten different seeds for each condition was measured as mean reward per episode (indicates mean velocity over simulation time).

III. Generalization and Learning on Uneven Terrain

Controllers were further evaluated on novel, uneven terrain that weren't experienced during learning. All controllers trained on the flat terrain were again tested for 100 simulation runs each on different uneven terrains. The terrains were generated from height-maps, using three different maximum heights of $0.05\,\mathrm{m}$, $0.10\,\mathrm{m}$, and $0.15\,\mathrm{m}$. Height-maps were generated using the diamond-square algorithm [2]. Detailed results are given in table III and see Fig. 4.

TABLE III: Comparison of rewards between the different control architectures during evaluation. Data was collected during evaluation for 100 episodes for each of controller. Given are mean rewards (and standard deviation in brackets) for each group of controllers (each group consisted of ten individually trained controllers). There were two different controller architectures, decentralized and centralized approaches, and two different training conditions, trained on flat terrain or on uneven terrain (using a height map with maximum height of 0.10 m). Values in italic reflect controllers that were evaluated on the same terrain they were trained on.

Condition	Decentralized Arch.		Centralized Arch.	
Trained on …	Flat Terrain	Uneven Terrain	Flat Terrain	Uneven Terrain
Evaluation on flat terrain	*652.93(96.68)*	466.09(69.27)	*533.73(151.66)*	466.56 (39.12)
Evaluation on height map (0.05 m)	371.20(78.05)	419.55(42.36)	356.83(90.15)	404.95(48.66)
Evaluation on height map (0.10 m)	371.46(73.40)	419.53(47.89)	346.56(103.59)	388.00(59.91)
Evaluation on height map (0.15 m)	342.66(91.38)	362.98(60.94)	287.26(105.46)	336.94(77.18)

IV. Implementation

The code of the implementation is open-sourced and available online. The easiest way to reproduce the results is to use the existing docker-image (from `kkonen/ros2_phantomx:master_thesis_final`). Detailed information on how to run the simulation and evaluation can be found in the repository accompanying the supplemental material at `https://github.com/malteschilling/ddrl_hexapod`.

In the experiments, we used a simulated robot[1] in the ROS simulation environment *Gazebo* [3] as it is open source and because of the good support of ROS (here ROS 2, version Dashing Diademata [4]). As a physics-engine *Bullet* was used [5]. The simulation required several custom modules (e.g., publishing joint-effort controllers in ROS 2) that have been

[1]robot definitions: `https://github.com/kkonen/PhantomX`

implemented now for ROS 2[2]. One current difference between simulated robot and the real robot was the implementation of ground contact sensors using data from the simulation environment, but solutions for such sensors have been proposed for the robot as well [6]. Simulator and ROS were run with a fixed publishing-rate of $25\,\mathrm{Hz}$ in order to align sensor and actuator data. This guaranteed a stable and constant step-size.

As a framework for Deep Reinforcement Learning (DRL) we employed OpenAI's gym. The connection to the simulator through the Robotic Operation System (ROS) was wrapped as a new environment through a connector using *ROS2Learn* [7]. This allowed to use the standard baseline implementations of DRL algorithms [8]. In this approach Proximal Policy Optimization (PPO) [9] was used as it has shown to work well on continuous tasks without the need of intensive hyperparameter tuning. As a further advantage, it has a comparably low sampling complexity.

For details on the architectures and hyperparameters see original paper.

REFERENCES

[1] C. Colas, O. Sigaud, and P.-Y. Oudeyer, "A Hitchhiker's Guide to Statistical Comparisons of Reinforcement Learning Algorithms," *arXiv:1904.06979 [cs, stat]*, Apr. 2019. [Online]. Available: http://arxiv.org/abs/1904.06979

[2] G. S. P. Miller, "The definition and rendering of terrain maps," *SIGGRAPH Comput. Graph.*, vol. 20, no. 4, p. 39–48, Aug. 1986. [Online]. Available: https://doi.org/10.1145/15886.15890

[3] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, Sep. 2004, pp. 2149–2154 vol.3. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/1389727

[4] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[5] E. Coumans, "Bullet physics simulation," in *ACM SIGGRAPH 2015 Courses*, ser. SIGGRAPH '15. New York, NY, USA: Association for Computing Machinery, 2015. [Online]. Available: https://doi.org/10.1145/2776880.2792704

[6] M. Tikam, "Posture control of a low-cost commercially available hexapod robot for uneven terrain locomotion," Ph.D. dissertation, University of Pretoria, 2018.

[7] Y. L. E. Nuin, N. G. Lopez, E. B. Moral, L. U. S. Juan, A. S. Rueda, V. M. Vilches, and R. Kojcev, "Ros2learn: a reinforcement learning framework for ros 2," 2019.

[8] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov, "Openai baselines," https://github.com/openai/baselines, 2017.

[9] 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

[2]for details see the open repository `https://github.com/kkonen/ros2learn`