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.

I. Performance after Learning on Flat Terrain

In the first experiment, controllers were trained on flat terrain for 5000 epochs. We trained 15 controllers of each type.

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 fifteen 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	26.01	2.
Seed 2, Decentralized	707.283	33.84	6.
Seed 3, Decentralized	570.693	39.56	21.
Seed 4, Decentralized	681.352	26.10	8.
Seed 5, Decentralized	756.096	27.50	3.
Seed 6, Decentralized	695.860	24.94	7.
Seed 7, Decentralized	733.194	56.82	4.
Seed 8, Decentralized	575.666	48.79	20.
Seed 9, Decentralized	598.666	30.16	18.
Seed 10, Decentralized	617.380	23.71	15.
Seed 11, Decentralized	638.920	38.65	11.
Seed 12, Decentralized	721.685	37.84	5.
Seed 13, Decentralized	551.884	58.02	25.
Seed 14, Decentralized	640.512	30.23	10.
Seed 15, Decentralized	603.263	29.08	16.
Mean performance	657.111		
Std. dev. over controller	68.072		

For statistical analysis, we compared the two architectures using a two-tailed Welch t-test (see [?]), applying the function from the scipy-stats package (`ttest_ind`) which returned a p-value of 0.011. The relative effect size was 1.02 which indicates a large effect size.

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 which was confirmed by a rank-sum test showing a significant difference (Wilcoxon–Mann–Whitney test, $p = .041$).

II. Learning on Flat Terrain

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

[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 fifteen 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	26.81	27.
Seed 2, Central/Baseline	770.158	21.52	1.
Seed 3, Central/Baseline	225.315	86.06	30.
Seed 4, Central/Baseline	566.485	13.29	22.
Seed 5, Central/Baseline	652.852	18.28	9.
Seed 6, Central/Baseline	302.733	5.48	29.
Seed 7, Central/Baseline	565.406	25.68	23.
Seed 8, Central/Baseline	600.676	23.58	17.
Seed 9, Central/Baseline	592.450	21.06	19.
Seed 10, Central/Baseline	557.175	27.35	24.
Seed 11, Central/Baseline	470.521	16.68	28.
Seed 12, Central/Baseline	621.663	26.89	14.
Seed 13, Central/Baseline	629.747	16.32	13.
Seed 14, Central/Baseline	630.497	14.97	12.
Seed 15, Central/Baseline	510.738	14.48	26.
Mean performance	546.701		
Std. dev. over controller	131.229		

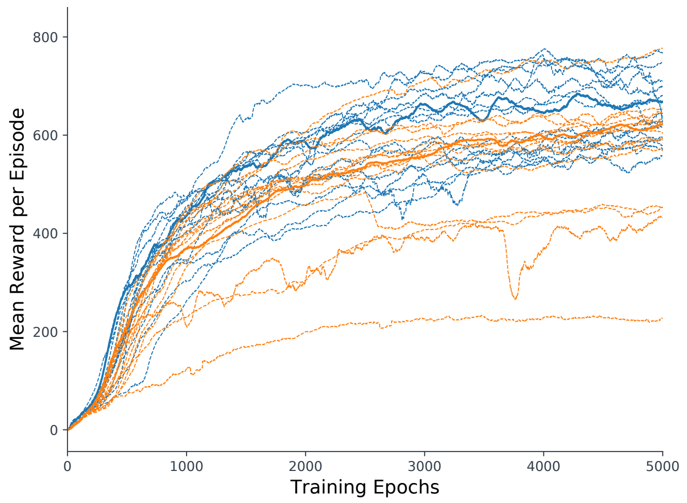

Fig. 1: Showing performance during training on flat terrain for individual seeds for 5000 epochs. Performance is measured as mean reward per episode (calculated as a running average over the last 100 episode in order to smoothen results). Results are given for the fifteen decentralized controllers (blue)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

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 fifteen 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 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 2187 epochs already).

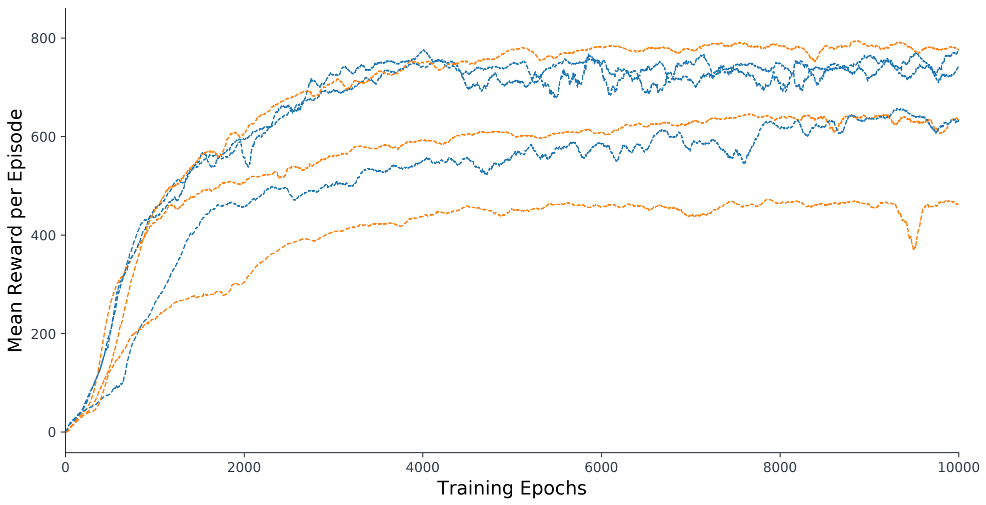

Fig. 2: Showing performance during training on flat terrain for individual seeds for 10000 epochs. Performance is measured as mean reward per episode (calculated as a running average over the last 100 episode in order to smoothen results). Results are given for selected decentralized controllers (blue)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

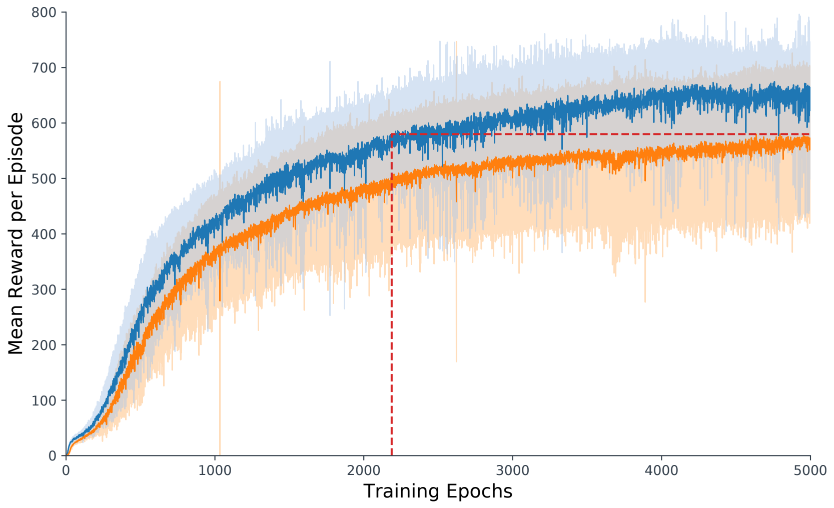

Fig. 3: Comparison of the mean reward during training: mean performance over the fifteen decentralized controllers is shown in blue and mean reward for the baseline centralized approach is shown in orange (shaded areas show standard deviation). Performance is measured as reward per episode. Seeds were 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 2187 epochs already).

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 [**?**]. Detailed results are given in table III and see Fig. 4.

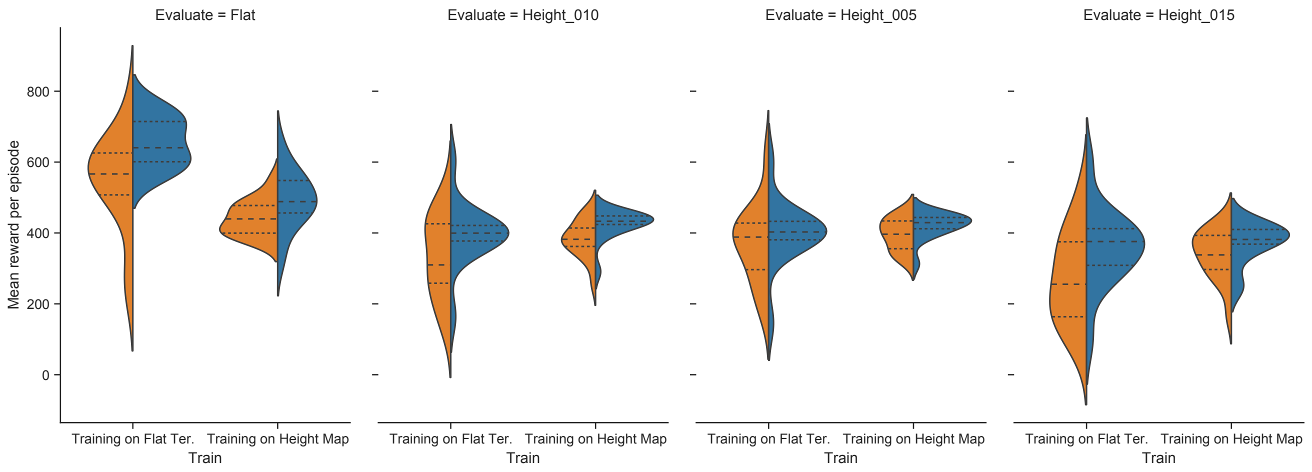

Fig. 4: Comparison of the (approximated) performance distribution for the controllers. Shown are violin plots for four evaluation conditions, from left to right: evaluation of walking on flat terrain; evaluation on three different height maps of maximum height of $0.05\,\mathrm{m}$, $0.10\,\mathrm{m}$,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

TABLE III: Comparison of rewards between the different control architectures during evaluation. Data was collected during evaluation for 100 episodes for each controller. Given are mean rewards (and standard deviation in brackets) for each group of controllers (each group consisted of fifteen 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	*657.11(68.07)*	492.69(79.22)	*546.70(131.23)*	441.50(48.00)
Evaluation on height map (0.05 m)	400.86(87.15)	421.06(35.53)	369.14(105.84)	404.95(43.93)
Evaluation on height map (0.10 m)	397.91(83.87)	*424.30(40.09)*	334.06(112.94)	*382.42(51.23)*
Evaluation on height map (0.15 m)	357.68(104.93)	371.10(55.83)	271.09(125.96)	339.41(67.63)

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

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

This allowed to use the standard baseline implementations of DRL algorithms [**?**]. In this approach Proximal Policy Optimization (PPO) [**?**] 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.