Robust Central Pattern Generators for Embodied Hierarchical Reinforcement Learning

Matthijs Snel
Intelligent Systems Lab Amsterdam
University of Amsterdam
Amsterdam, The Netherlands
Email: m.snel@uva.nl

Shimon Whiteson
Intelligent Systems Lab Amsterdam
University of Amsterdam
Amsterdam, The Netherlands
Email: s.a.whiteson@uva.nl

Yasuo Kuniyoshi
Intelligent Systems and Informatics Lab
The University of Tokyo
Tokyo, Japan
Email: kuniyosh@isi.imi.i.u-tokyo.ac.jp

***Abstract*—Hierarchical organization of behavior and learning is widespread in animals and robots, among others to facilitate dealing with multiple tasks. In hierarchical reinforcement learning, agents usually have to learn to recombine or modulate low-level behaviors when facing a new task, which costs time that could potentially be saved by employing intrinsically adaptive low-level controllers. At the same time, although there exists extensive research on the use of pattern generators as low-level controllers for robot motion, the effect of their potential adaptivity on high-level performance on multiple tasks has not been explicitly studied. This paper investigates this effect using a dynamically simulated hexapod robot that needs to complete a high-level learning task on terrains of varying complexity. Results show that as terrain difficulty increases and adaptivity to environmental disturbances becomes more important, low-level controllers with a degree of instability have a positive impact on high-level performance. In particular, these controllers provide an initial performance boost that is maintained throughout learning, showing that their instability does not negatively affect their predictability, which is important for learning.**

I. Introduction

Hierarchical organization of behavior and learning is widespread in animals [1], [2], and has also been adopted in robotics [3] and machine learning [4]. Generally, layers near the bottom of the hierarchy are responsible for low-level, "subconscious" tasks, such as generating neural patterns for locomotion, while higher layers carry out cognitive tasks, such as finding food, by chaining and modulating low-level behaviors. For example, an animal may adapt direction or speed of locomotion in order to find food.

In principle, adaptation to *changes* in environment could be organized in a similar hierarchical fashion. For example, in the context of locomotion, subtle adaptation of gait to low-level terrain variations (e.g. flat, rocky, debris) could be at least partly handled by lower layers in order to reduce the burden on cognitive processing.

In general, we hypothesize that low-level robustness in a hierarchical system can reduce the need for high-level learning. For example, a walking animal or robot would have to spend less effort to learn to adapt its gait to different terrains. Moreover, while the robot could eventually learn to avoid difficult patches of terrain, having an adaptive low-level gait controller would reduce the need to do so.

We investigate this hypothesis in an embodied setting: goal-directed locomotion of a dynamically simulated hexapod robot on several different terrain types. Cognitive processing is represented by a reinforcement learning (RL) module. In particular, we use temporal-difference learning [5], a popular method for learning optimal control policies that has been linked to learning in the brain (e.g. [6]).

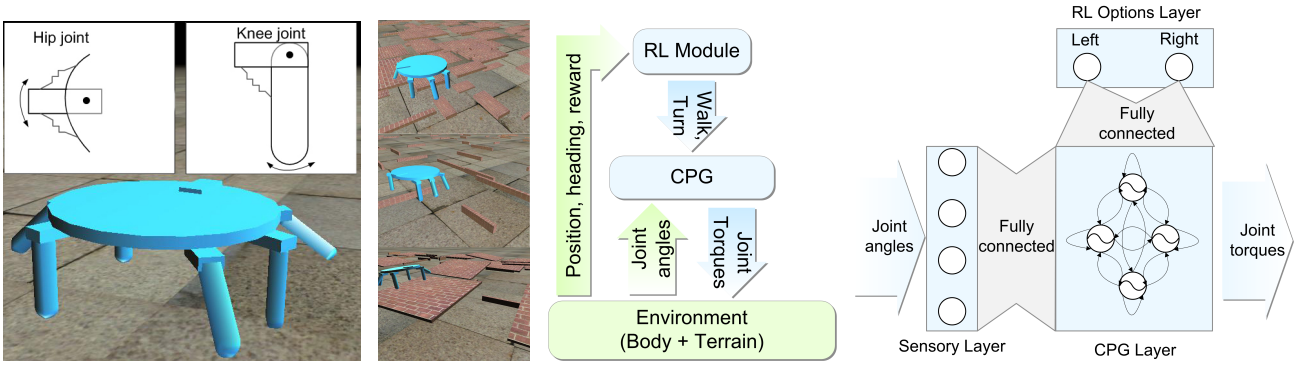
In hierarchical RL, high-level behaviors are typically learned while low-level behaviors are either pre-designed and fixed [7], [8] or also learned [9], [10], [11], [12]. Both approaches can be problematic when adaptation to environmental changes is required. If low-level behaviors are fixed, then the burden of adaption falls entirely on the high-level behaviors, which must learn how to use the same low-level behaviors in a new environment. If low-level behaviors are learned, then they can adapt to the new environment but doing so may be time consuming, yielding poor performance in the meantime.

In this paper, we demonstrate that the performance of hierarchical RL can be improved by using intrinsically robust low-level controllers. Since these controllers adjust automatically to environmental changes, low-level learning is not required and the burden on high-level learning is reduced.

To achieve robust low-level behavior for the robot's legs, we employ several models for central pattern generators (CPGs), which are networks of neurons capable of autonomously generating oscillating output patterns [13]. There has been extensive research demonstrating that CPGs can achieve stable locomotion, with varying degrees of adaptivity in the face of environmental disturbances [14], [15], [16], [17], [18]. However, locomotion is merely a means to achieving a goal, not a goal in itself. In addition, flexibility can come at the cost of predictability, which is important for learning. Therefore, it is important to also evaluate these systems in the context of high-level learning. To the best of our knowledge, this paper presents the first results on the influence of various CPG types on the performance of high-level goal-directed behavior.

II. Problem Setting

In this section, we describe the specific simulated robot setting in which we evaluate our approach. There are three main properties that such a setting should possess in order to be suitable for evaluating our hypothesis that low-level



(a) Simulator snapshot (b) (c) Framework (d) Neural architecture

Fig. 1. Hexapod (a), terrains (b), learning framework (c), and neural architecture (d). The latter only depicts 4 of the 12 nodes in the sensory and CPG layer.

robustness can reduce the need for high-level learning in different environments.

First, while the body morphology should be simple so as to isolate the effects of each controller, it should also be complex enough to justify a hierarchical approach and to make stable locomotion non-trivial. To this end, we use a hexapod robot with a single disk-shaped body and 2-DOF legs, as shown in Fig. 1(a). As shown in the insets, a spring-damper system is connected to both the hip and knee joint, pulling back respectively the upper and lower leg to their resting positions. This adds important passive dynamics to the system that, among others, enable the robot to stand upright even if no torque is exerted on the joints.

Second, since our hypothesis concerns the robustness of low-level controllers, the problem setting should require those controllers to adapt to environmental changes. To this end, we consider four different terrains on which the robot must walk. In addition to the default *flat* terrain, we use the three terrains shown in fig. 1(b). From top to bottom, they are the *rubble* terrain, containing fixed rectangular obstacles of variable length, width, and height; he *bars* terrain, consisting of long, thin, and high rectangles; and the *wobbly* terrain, containing both fixed obstacles and loose planks. While final evaluation of the controllers takes place on these terrains, we optimize our models on terrains that are simpler, yet share properties with the final terrains. This tests the controllers' ability to deal with unforeseen, and more difficult, terrains that are nonetheless not completely new. Section IV-A will explain this in more detail.

Third, since our hypothesis concerns how low-level control affects high-level performance, each task should require not only locomotion but also achieving a high-level objective. Thus, the robot must not only walk successfully in each terrain but also travel to a specific goal area. We expect that low-level controllers that are better able to handle the various terrain types will help the high-level learner more quickly discover how to reach the goal.

III. Method and Related Work

Figure 1(c) shows the general hierarchical controller framework. At the bottom level, the robot's legs are controlled by a CPG network that receives as input a 12-dimensional vector of joint angles and outputs a vector of torques that are applied to the joints. The CPG makes the robot walk, while at the top level, the RL module decides whether to walk straight or turn left or right, based on its current position and heading.

The rest of this section describes the models used for each component and provides a comparison to related work. Discussion of the parameterization of the models is left to section IV, which explains the experimental set-up.

A. CPG Controller

When dealing with multiple unknown terrains of varying difficulty, model-based and trajectory-planning approaches to locomotion become infeasible. Central pattern generators (CPGs) are widely deployed for locomotion in both animals and robots [16], [2]. While CPGs are capable of producing rhythmic output patterns autonomously, the addition of sensory feedback can modulate rhythmic patterns and synchronize neural and body dynamics [19], [16], [20].

In robotics, CPGs are usually modeled using dynamical systems, often based on coupled oscillators [16]. Such systems are attractive because of their potential for entering stable limit-cycle behavior, allowing the generation of oscillatory patterns. An alternative approach to CPG modeling that is receiving increasing interest is one based on chaotic systems. A key property of such systems is that, although their trajectory through state space remains confined to a bounded region (a *strange attractor*), their behavior within this region is erratic. This inherent bounded instability "facilitates the extraordinary ability of neural systems to adapt, make transitions from one pattern of behavior to another when the environment is altered, and consequently [to] create a rich variety of patterns" [20]. Thus, a potential advantage of chaotic systems over more conventional dynamical systems is their ability to incorporate various behavioral patterns, with different patterns emerging in different circumstances.

In fact, previous studies have demonstrated the inherent adaptivity of coupled chaotic systems. Kuniyoshi and Suzuki [17] showed that an insect-like robot controlled by a weakly coupled chaotic map can spontaneously generate movement

and adjust that movement when it encounters obstacles. Similarly, when terrain roughness increases, a robot controlled by a chaotic Rössler system has been shown to cover more distance than one using stable limit-cycle behavior [15].

These examples underscore the adaptive ability of chaotic systems. However, we do not claim that chaotic systems are required: other systems with a degree of instability could exhibit similar properties.

In this paper, we evaluate how one chaotic and three non-chaotic systems affect performance on a high-level task. Low-level instability can decrease predictability, which is important for high-level control. Hence, we investigate whether flexible low-level controllers retain enough predictability that their overall effect is to speed high-level learning on various terrains.

The framework for all CPG types that we compare is a two-layer recurrent neural net (RNN) (fig. 1(d)), consisting of a *sensory layer* that is fully connected to the *CPG layer*, which is in turn fully connected to itself. The network has one sensory node and one CPG node at each robot joint; each CPG node sends motor output only to its own joint. Thus, 6 CPG nodes output to hip joints and 6 to knee joints. The *RL options layer*, in which high-level learning interacts with the CPG, is discussed in the next section.

Input to the sensory layer consists of a vector of joint angles linearly transformed by a sensor gain and bias, and CPG output is similarly transformed by a motor gain and bias before it is applied as torque to the joints. The rest of this section describes how the four CPG models that we compare are implemented in this neural-network framework.

The first model is a discrete-time *chaotic* CPG in which the output $\mathbf{x}_n$ at the nth update is given by:

$$\mathbf{x}_n = \mathbf{f}(\mathbf{K}\mathbf{x}_{n-1} + \mathbf{W}\mathbf{s}_n), \tag{1}$$

where $\mathbf{K}$ is the matrix of CPG layer weights (couplings between nodes), $\mathbf{W}$ is the weights from sensory to CPG nodes, $\mathbf{s}_n$ is the sensory input at update n, and $\mathbf{f}$ is the logistic map:

$$\mathbf{f}(\mathbf{x}) = \mathbf{1} - \alpha \mathrm{diag}(\mathbf{x}\mathbf{x}^{\mathrm{T}}), \tag{2}$$

where α is a parameter that influences how chaotic the function is: for low values of α, the function's behavior is stable, either settling on a single value (fixed point) or oscillating regularly (limit cycle). For higher values, output oscillates irregularly and never exactly repeats itself, but remains confined to a bounded interval (chaotic).

The next two models are not usually seen as CPGs, but have the ability to exhibit oscillatory patterns. We include them here for comparison purposes, and because they represent standard RNN models. The first of these is a discrete-time RNN (*DTRNN*); it is the same as the chaotic model except that $\mathbf{f}(\cdot)$ is a sigmoid function:

$$\mathbf{f}(\cdot) = \tanh(\cdot). \tag{3}$$

The third model is a continuous-time RNN (*CTRNN*), of which output $\mathbf{x}$ is governed by:

$$\tau\dot{\mathbf{x}} = -\mathbf{x} + \sigma(\mathbf{K}\mathbf{x} + \mathbf{W}\mathbf{s}), \tag{4}$$

where $\dot{\mathbf{x}}$ is the derivative with respect to time, τ is a time constant and $\sigma(\cdot)$ the $\tanh(\cdot)$ sigmoid function. This is a fairly standard CTRNN model; the $-\mathbf{x}$ term drives the activation of each node to 0, which together with the sigmoid function ensures network output stays bounded.

The fourth model is a *coupled oscillator*. Coupled oscillators are widely employed in CPG modeling in robotics [16]. However, there is great variation in the kind of models used, and they are often tailored to a specific robot. For the sake of comparison, we employ a basic model that is a variation on the Kuramoto model [21]:

$$\tau\dot{\theta}_i = \omega_i + \sum_{j=1}^{N} k_{ji}\sin(\theta_j - \theta_i) + \mathbf{w}_i^{\mathrm{T}}\mathbf{s} \tag{5}$$

$$\mathbf{x} = \cos(\theta), \tag{6}$$

where θ_i and ω_i are the phase and intrinsic frequency of node i, k_{ji} is the coupling weight from node j to i, and $\mathbf{w}_i$ is the vector of incoming sensory weights to node i. Without coupling or sensory input, each node is driven to oscillate at its intrinsic frequency; coupling encourages phase synchronization of nodes while sensory input encourages synchronization to external dynamics.

B. RL Module

A reinforcement-learning agent interacts with an unknown environment by sequentially choosing actions based on sensory input, or state. For each action, the agent receives a *reward*, and its goal is to learn a *policy* – a mapping from states to actions – that maximizes *return*: the expected cumulative reward. In most realistic scenarios, it is also important to maximize reward accrued *during* learning.

Basic RL methods do not cope well with tasks with large or continuous state or action spaces. One tool for combating this scaling problem is temporally extended actions, in which the agent follows a sub-policy for multiple time steps. Such actions enable hierarchical approaches in which both the task and the state-action space are decomposed into sub-tasks and sub-spaces that are tackled separately, and have also been linked to psychological and neuroscientific constructs [28]. In our setting, one result of this decomposition is that the RL module at the high level need not consider the continuous action space addressed by the CPG at the low level.

In this paper we employ the *options* framework [4]. An option is a temporally extended action parameterized by a set of states in which the option can be selected, a policy to follow during option execution, and a function $\beta(s)$ that gives the probability of terminating in state s. Generally, the agent selects a new option once the currently selected option terminates according to β. However, once the agent has a good estimate of each option's expected return, it can learn to *interrupt* an executing option before termination if other options in that state yield higher expected return.

Our framework uses a CPG network with fixed parameters to implement three options: `walk`, `turn-left` and

`turn-right`. As shown in fig. 1(d), the RL module interacts with the CPG through two neural nodes that are fully connected to the CPG layer. For turning left, the RL module activates the left node (by clamping its value to 1), while for turning right, it activates the right node. When neither node is active, the robot walks into the direction of its current heading.

This approach differs substantially from existing work regarding options. Much research, e.g., [7], has investigated the use of fixed, pre-designed options for use in a single task. Fixed options have also been used in multi-task settings to facilitate *transfer learning* [22], by representing a fixed behavior that is useful in different tasks. Our option parameters are similarly fixed but, since their behavior is inherently adaptive, they are especially well suited to use across multiple tasks (e.g., terrains).

However, options are also commonly learned, often based on identification of subgoals [23], [24], [11], [12], [25], [26]. While options that learn can obviously adapt to new tasks, doing so is typically time-consuming. In contrast, the inherent robustness of our options allows them to adapt without additional low-level learning.

Furthermore, while dynamical systems have previously been used as policies in RL [9], [10], [27], the main focus was on exploiting the limit-cycle (rhythmic) or point-attractor (single-stroke movement) behavior of these systems. To the best of our knowledge, the capacity of options based on dynamical systems to automatically adapt to environmental changes has not been explicitly exploited to date.

IV. Experiments and Results

The goals of the following experiments are to 1) demonstrate that the robot can learn to reach a goal area on the four terrain types described in section II, 2) compare how the controllers fare on each terrain, and 3) qualitatively relate performance to the CPGs' adaptivity. However, we begin by describing experiments conducted to optimize the parameters of the CPG models described in section III-A and qualitatively assess adaptivity via a dynamical systems analysis.

A. CPG Weights and Properties

Due to the complex nonlinear dynamics of some of the models, particularly the chaotic map, weight learning is infeasible. In addition, it is likely that in nature low-level controllers are at least partly preset by evolution, selected for the ability to cope with environmental variation. Therefore, we employ an *evolutionary algorithm* (EA) [29] to optimize the weight matrices $\mathbf{W}$ and $\mathbf{K}$ and the time constant of the models. For the coupled oscillator model, we also evolve a single intrinsic frequency ω used for all nodes. We first evolve parameters for walking straight and fix them, then separately evolve the coupling between the RL options and CPG layers for turning.

For all experiments, sensor gain $g_s = 0.625$, sensor bias $\mathbf{b}_s = \mathbf{0.5}$, motor gain $g_m = 70$, and motor bias is -26 for hip nodes and 0 for knee nodes. Instead of evolving the full weight matrices, we evolve a genetic template of 48 weights for a single leg: 12 sensory connections for the hip and knee joint, and 12 couplings for the hip and knee CPG node. Thus, each hip CPG node is connected in the same way to its own leg and neighboring legs, and the same holds for each knee CPG node. This corresponds to mechanisms employed in a developing embryo, where genetic templates are re-used to form for example limbs and segmental structures [30].

Fitness evaluation consists of simulating the robot for 1000 time steps, twice on each of four terrains: the flat terrain; a terrain containing a single 6 degree slope; a terrain with wobbly properties, consisting of a collection of loose planks; and the rubble terrain. As explained in section II, these terrains share basic properties with the terrains on which the robot will be evaluated. Thus, the controllers are allowed to incorporate the basic patterns necessary for dealing with these terrain types, though in testing they must synthesize these patterns and extend them to more difficult situations.

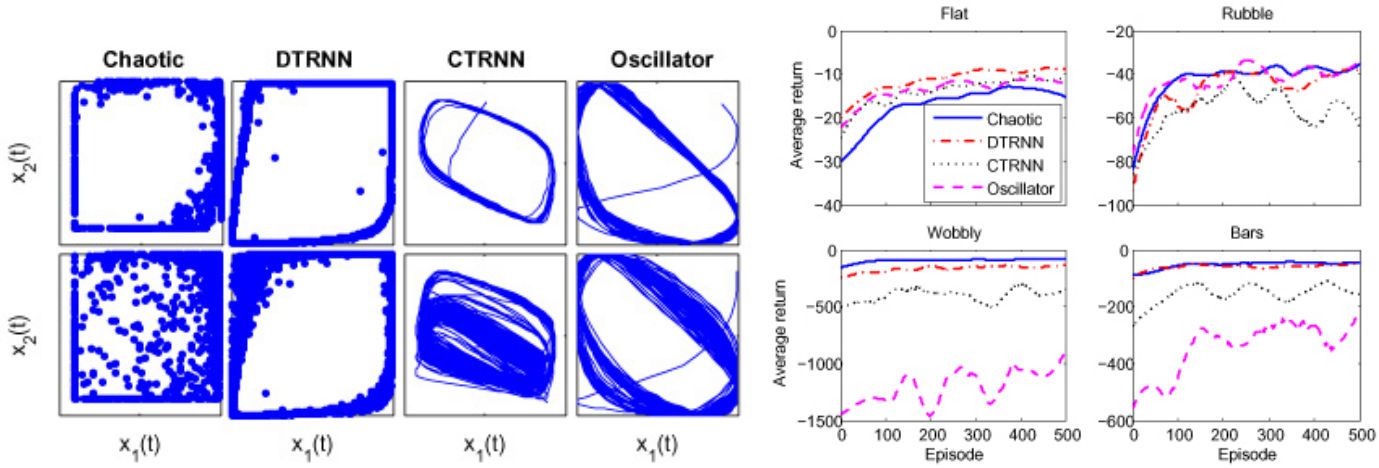
The fitness function determines the direction the robot heads in during the first 200 steps; the score for each fitness evaluation is the distance traveled in this direction after 1000 steps. The final fitness score is the average of the 8 measured fitness evaluations. Before starting the EA, we initialize parameters randomly close to 0, but thereafter do not restrict them, except for the time constant τ of the continuous-time models, which we sample from a normal distribution $\mathcal{N}(0.5, 0.2)$, and the intrinsic frequency of the oscillator model, $\omega \sim \mathcal{N}(6, 0.5)$.

Figure 2(a) shows phase portraits for each evolved controller, on the flat and rubble terrain. This figure illustrates a key difference between the controllers: on the flat terrain, without any disturbances, cyclic behavior is evident for all controllers but the chaotic controller is clearly less stable. On the rubble terrain, its behavior becomes even more erratic, exploring the state space for viable behavioral patterns. Although a difference can also be seen for the other controllers, they stay closer to their limit cycles.

B. Learning

As explained in section III-B, RL options for each model consist of the CPG network with evolved parameters. The goal of learning is to reach a fixed cylinder-shaped goal area with the same radius as the robot's body; a learning episode terminates when the robot's center enters the cylinder. The robot's body has diameter 1, and the environment is 5x5, in arbitrary units. Option parameters are as follows. The `walk` option terminates whenever the robot is about to leave the environment, and is ineligible while the robot is near the edge of the world and facing outward. The `turn` options are always eligible, and terminate if the robot looks into the direction of the goal area, in order to keep episodes short. Note that in spite of this, the robot still has to learn whether to turn left or right and when to interrupt options, since just walking when facing the goal usually does not get the robot to the goal, especially on difficult terrains.

Reward is -0.01 per simulator time step (10ms), and 1 upon reaching the goal area. To speed learning, we assume the robot has an innate sense of direction and distance to the goal (e.g., through smell), and give it an additional positive (negative)



(a) Phase portraits

(b) High-level performance

Fig. 2. (a) Phase portraits of the two nodes controlling the joints of the right front leg for each model, measured on the flat terrain (top row) and rubble terrain (bottom row). Range of each variable is [-1,1]. (b) Learning curves for each controller on each terrain. Results are strongly smoothed averages over 20 runs. For an indication of variance, see fig. 3. The wavy pattern of the graphs is caused by the noisiness of the results.

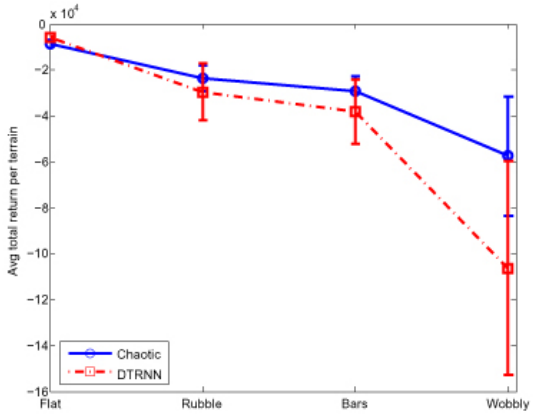


Fig. 3. Average total return per run (i.e., summed over all episodes in a run) per terrain, together with the standard deviation. This represents the area between the (unsmoothed) learning curves of fig. 2(b) and the line return=0. (CTRNN and oscillator not shown, see main text.)

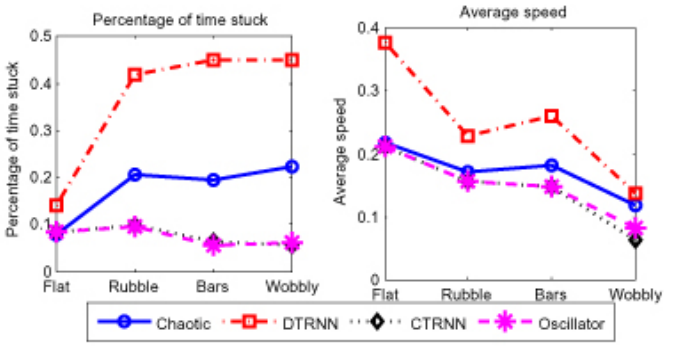


Fig. 4. Average speed and percentage of time steps in which a controller is not moving in each terrain.

reward when distance to the goal is decreased (increased). Evaluation on each terrain consists of 500 learning episodes; at the start of each episode, the robot is placed at a random location facing a random direction.

Figure 3 presents the average total return per terrain accumulated by each controller, while 2(b) displays learning curves. All results are smoothed averages over 20 runs.

On flat terrain, the DTRNN provides the best performance, while the chaotic controller does worst. This is in accordance with previous results [15], which showed that on easy terrain, controllers with stable limit-cycle behavior cover the largest distance. The right panel of fig. 4 confirms this, showing that the DTRNN has highest average speed, while the chaotic controller is slowest.

On the rubble terrain, the situation starts to reverse, with the chaotic controller outperforming the other controllers by a small margin, and the CTRNN doing worst. Continuous-time controller performance significantly deteriorate with increasing terrain difficulty, while the chaotic controller is less affected. The left panel of fig. 4 shows the percentage of time steps on which controllers get stuck, i.e., have moved less than 0.01 distance units in one second. In light of their bad performance, it is surprising to see that the CTRNN and oscillator controllers get stuck the least often, although their average speed does decrease. Further path analysis shows that this is due to the robot wandering off beyond the edge of the world imposed by the option termination conditions. Since turn options are always eligible, this is probably due to premature option interruption, which causes the robot to keep turning left and right. It is not clear, though, why this happens only for these controllers on difficult terrains. Unfortunately, we did not have time to further investigate this issue, so we will focus on the difference between the chaotic and DTRNN controller.

Notice the relation between the performance results and the phase portraits of fig. 2(a). The chaotic controller's phase portrait clearly deviates from that of the flat terrain, which is reflected in its greater adaptivity and performance on difficult terrains.

From the learning curves in fig. 2(b), it is not entirely clear

whether the more adaptive low-level controller (i.e., chaotic) speeds high-level learning. If it did, we would expect the chaotic controller's performance to asymptote soonest, with the DTRNN achieving similar performance only after more learning. Nevertheless, these results demonstrate the positive effect of low-level adaptivity on high-level performance in a hierarchical system: especially on difficult terrains such as the wobbly terrain, initial performance is much higher than for the DTRNN controller, and remains higher throughout learning.

V. Discussion and Future Work

This paper presents a study of the effect of adaptivity of various low-level controllers in a hierarchical reinforcement learning controller. Controllers are evaluated using a dynamically simulated hexapod robot that needs to complete a high-level learning task on terrains of varying complexity.

Results show that on flat terrain, where speed is more important than adaptivity, controllers with stable limit-cycle behavior achieve the highest return by reaching the goal area in the shortest time. On the other hand, when terrains become more difficult and adaptivity to environmental disturbances becomes more important, the chaotic controller, which has the most unstable, and thus most flexible, phase-space behavior, significantly outperforms other controllers. This confirms the findings of previous studies [15], [17] and extends them to hierarchical learning on high-level tasks.

Analysis of the chaotic and DTRNN controller's behavior on each terrain showed a correlation between the amount of time a controller got stuck and its high-level performance: the less a controller got stuck, the better its performance. This is a clear indication that low-level adaptivity is an important factor in achieving high-level return. Unfortunately, we were not able to accurately investigate the relation between adaptivity and performance of the other two controllers. We are currently addressing this issue.

From a hierarchical reinforcement-learning perspective, the relation between low-level adaptivity and high-level performance shows that adaptive dynamical systems are useful for dealing with multiple tasks (terrains, in this case). Even though controllers were optimized on a simple set of terrains, the adaptivity of the chaotic controller allowed it to achieve high return on much more difficult terrains, without re-learning. This can present a performance boost that can be further fine-tuned through low-level learning. Such an approach would be an interesting extension for future work.

Although the chaotic controller's behavior is more erratic and hence unpredictable, it is not so unpredictable as to hinder learning. However, there must be a point beyond which more chaotic behavior negatively affects learning performance. Future studies should further investigate and quantify this relation between instability, adaptivity, and learning performance.

Acknowledgments

We thank the anonymous reviewers for useful comments. Special thanks to Alex Pitti for many discussions. M. Snel was partly supported by JSPS Fellowship grant PE09053.

References

[1] N. Tinbergen, "The hierarchical organization of nervous mechanisms underlying instinctive behaviour," *Soc. Exp. Biol.*, vol. 4, 1967.
[2] F. Delcomyn, "Walking robots and the central and peripheral control of locomotion in insects," *Autonomous Robots*, vol. 7, pp. 259–270, 1999.
[3] R. A. Brooks, "A robust layered control system for a mobile robot," *IEEE Journal of Robotics and Automation*, vol. RA-2, no. 1, 1986.
[4] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artificial Intelligence*, vol. 112, pp. 181–211, 1999.
[5] R. Sutton, "Learning to predict by the method of temporal differences," *Machine Learning*, vol. 3, pp. 9–44, 1988.
[6] W. Schultz, P. Dayan, and P. R. Montague, "A neural substrate of prediction and reward," *Science*, vol. 275, pp. 1593–1599, 1997.
[7] P. Stone, R. S. Sutton, and G. Kuhlmann, "Reinforcement learning for Robocup soccer keepaway," *Adaptive Behavior*, vol. 13, no. 3, 2005.
[8] M. Huber and R. A. Grupen, "Learning to coordinate controllers - reinforcement learning on a control basis," in *IJCAI*, 1997.
[9] J. Kober and J. Peters, "Policy search for motor primitives in robotics," *Machine Learning*, pp. 1–33, 2010.
[10] S. Schaal, P. Mohajerian, and A. Ijspeert, "Dynamics systems vs. optimal control: a unifying view," *Progress in Brain Research*, vol. 165, 2007.
[11] P.-Y. Oudeyer, F. Kaplan, and V. Hafner, "Intrinsic motivation systems for autonomous mental development," *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 2, pp. 265–286, 2007.
[12] A. Barto, S. Singh, and N. Chentanez, "Intrinsically motivated learning of hierarchical collections of skills," in *ICDL*, 2004.
[13] F. Delcomyn, "Neural basis of rhythmic behavior in animals," *Science*, vol. 210, pp. 492–498, 1980.
[14] J. Nassour, P. Hénaff, F. B. Ouezdou, and G. Cheng, "A study of adaptive locomotive behaviors of a biped robot: patterns generation and classification," in *SAB*, 2010, pp. 313–324.
[15] L. Matthey, L. Righetti, and A. J. Ijspeert, "Experimental study of limit cycle and chaotic controllers for the locomotion of centipede robots," in *IROS*, 2008, pp. 1860–1865.
[16] A. J. Ijspeert, "Central pattern generators for locomotion control in animals and robots: A review," *Neural Networks*, vol. 21, 2008.
[17] Y. Kuniyoshi and S. Suzuki, "Dynamic emergence and adaptation of behavior through embodiment as coupled chaotic field," in *IROS*, 2004.
[18] G. Taga, Y. Yamaguchi, and H. Shimizu, "Self-organized control of bipedal locomotion by neural oscillators in unpredictable environment," *Biological Cybernetics*, vol. 65, pp. 147–159, 1991.
[19] H. J. Chiel, L. H. Ting, O. Ekeberg, and M. J. Z. Hartmann, "The brain in its body: Motor control and sensing in a biomechanical context," *The Journal of Neuroscience*, vol. 29, no. 41, pp. 12 807–12 814, 2009.
[20] M. I. Rabinovich, P. Varona, A. I. Selverston, and H. D. I. Abarbanel, "Dynamical principles in neuroscience," *Reviews of Modern Physics*, vol. 78, no. 4, pp. 1213–1265, 2006.
[21] S. Strogatz, "From Kuramoto to Crawford: Exploring the onset of synchronization in populations of coupled oscillators," *Physica D*, vol. 143, pp. 1–20, 2000.
[22] M. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey." *Journal of Machine Learning Research*, vol. 10, no. 1, pp. 1633–1685, 2009.
[23] G. Comanici and D. Precup, "Optimal policy switching algorithms for reinforcement learning," in *AAMAS*, 2010.
[24] G. Konidaris and A. Barto, "Skill discovery in continuous reinforcement learning domains using skill chaining," in *NIPS*, 2009.
[25] J. Morimoto and K. Doya, "Acquisition of stand-up behavior by a real robot using hierarchical reinforcement learning," *Robotics and Autonomous Systems*, vol. 36, pp. 37–51, 2001.
[26] A. McGovern and A. G. Barto, "Automatic discovery of subgoals in reinforcement learning using diverse density," in *ICML*, 2001.
[27] 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.
[28] M. M. Botvinick, Y. Niv, and A. G. Barto, "Hierarchically organized behavior and its neural foundations: A reinforcement learning perspective," *Cognition*, vol. 113, pp. 262–280, 2009.
[29] D. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.
[30] L. Wolpert, , T. Jessell, P. Lawrence, E. Meyerowitz, E. Robertson, and J. Smith, *Principles of Development*. OUP; 3rd ed., 2006.