Reinforcement Learning with Dynamic Movement Primitives - DMPs

Gerhard Kniewasser, kniewasser@student.tugraz.at
and Supervisor: Elmar Rückert
Department of Theoretical Computer Science, University of Technology, Graz

***Abstract*—In this project we set up the AMARSi Oncilla Simulator[1] and used Dynamic movement primitivies (DMPs) as movement representation and optimized their parameters in a reinforcement learning framework to adapt the robot's behaviour to new problems. After some experiments on toy examples we applied an open-loop control scheme to the Oncilla Simulator. In the end we want to apply this approach to a real robot, the AMARSi[2] Oncilla quadroped and evaluate its performance.**

I. Introduction

Teaching a robot to learn a certain behaviour is a non-trivial task, since the dynamics of such systems are highly non-linear and living in a high-dimensional space (e. g. 10 to over 50 dimensions). The increasing performance in today's computer systems allow for real-time control of robots with many DoF (e.g. humanoid robots). Therefore, control approaches such as [3] are very interesting for engineers and scientists. Nethertheless, it is mandatory to find a good movement representation, to achieve good learning results. In the field of leged robots CPGs (central pattern generators) found to be a good choice. There are quite many approaches, how to implement these CPGs. The authors in [1] present a novel bio-inspired approach using CPGs with nonlinear phase-coupled Hopf oscillator with reflex feedback signal for preventing from stumbling and virtual model control (VMC) for posture control. They achieve very good results. One of the most impressive results of leged robots in the present is the "Big Dog" from Boston Dynamics [7], which is cabable of walking in rougth terrain. For very complex movement plans, a discretization into a sequence of much simpler movement primitives seems, so far, to be the only chance to cope with this high complexity.

In this project we implement the CPG movement representations with DMPs. Dynamic movement primitives are a quite interesting research field, since DMPs provide a very general and simple system representation (e.g. point attractor or rythmic limit cycle are possible) and they are by themselves stable, so one does not have to cope with stability issues [3], [5]. In the following sections we give a brief overview and description of DMPs and the implemented learning framework.

II. DMP - Model Description

The system in Equation 1 descripes a second order dynamical system and is called the transformation system.

$$\begin{aligned} \tau\dot{z} &= \alpha_z(\beta_z(g-y)-z)+f_{task}(\phi), \qquad (1)\\ \tau\dot{y} &= z. \end{aligned}$$

The variables α_z, β_z are time constants which corresponds to the spring- and damper- coefficients in dynamic mechanical system. τ defines the period of the movement and g is the unique attractor in this system, which represents the goal state. Since we are looking for rythmic walking behaviour, we describe in the following equations only the limit cycle case.

The simple canonical system in Equation 2 represents the time evolution in this description, where ϕ is a substitute for the time t.

$$\begin{aligned} \tau\dot{\phi} &= 1, \qquad (2)\\ \phi &= mod(\phi, 2\pi). \qquad (3) \end{aligned}$$

If f_{task} would be zero, we would get an homogenous second order dynamic system converging to g. By introducing the function f_{task} it is possible to force the dynamical system to follow an arbitrary trajectory.

The forcing term f_{task} is approximated by a set of normalized radial basis function $\psi_i(\phi)$ and weights ω_i

$$\begin{aligned} f_{task} &= \frac{\sum_i^K \psi_i(\phi) w_i}{\sum_j \psi_j(\phi)} A, \qquad (4)\\ \psi_i(\phi) &= exp(h_i cos(\phi - c_i) - 1). \qquad (5) \end{aligned}$$

$\psi_i(\phi)$ in Equation 5 is represented by von Mises basis functions, which are periodic in ϕ and distributed over the centers c_i with precision h_i.

III. Optimization Framework

The parameters of the movement representation are $\mathbf{\Theta} = \{\omega_i\}, i = 1 : K$. The goal of the optimization is to find an optimal policy vector

$$\mathbf{\Theta}^* = argmin_{\Theta} C(\mathbf{\Theta}),$$

which minimizes the expected costs

$$C(\mathbf{\Theta}) = \mathbb{E}[J(\boldsymbol{y}, \dot{\boldsymbol{y}}) \mid \mathbf{\Theta}].$$

[1] http://docs.cor-lab.de/oncilla-sim-manual/0.1/html/index.html
[2] http://amarsi-project.eu

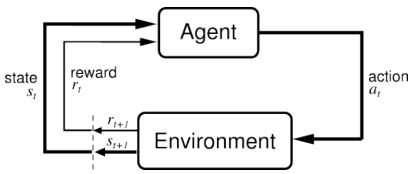


Figure 1: Reinforcement learning Environment [10]

DMPs are a model-free approach, where a model of the forward dynamics of a robot is not needed. We use the classic reinforcement learning framework [10] to optimize the gait of our Robots (see Figure 1). In this case the Agent generates a parameter vector $\boldsymbol{\omega}$ for each DoF and generates the desired trajectories (here: the actions a_t are the desired trajectories). The environment is represented by our simulator, or the real robot hardware, which generates a trajectory and a corresponding reward to evaluate the choice of the parameter vector.

To generate and optimize our parameters, we use a stochastic optimization algorithm called CMA-ES (Convolution Matrix Adaptation - Evolutionary Strategy) [2], which in short words samples the parameter space, picks the best samples and use it to update the mean μ and covarianz matrix Σ of the policy distribution to get closer to the (local) optimal solution.

A. Course of dimensionality

In the following experiments we use a biped walker model and the AMARSi oncilla robot. The size of the parameter vectors is shown in Tab I.

	DoF	K	dim of parameter vector
biped walker	4	4	16
oncilla	12	8	96

Table I: parameter vector dimensions for each robot. The **walker** impl. 2x Hip, 2x Knee. The **Oncilla** impl. 4xHip-Pro-/Retraction, 4xHip-Ab-/Adduction, 4xKnee. K is the number of parameters needed for the representation.

Imaging, we would have a discretized space, where each dimension is encoded by 10 possible values. We would get, in case of the walker, 10^{16} or 10^{96} possible values for the oncilla robot, where the optimal values live in a small subspace.

Due to the high dimensional contineous parameter space parameter learning is hardly feasible without a good initial solution, e.g. from demonstration or tuning by hand.

In the walker task we made use of the hand tuned values from Table II as initial state. The vector $\mathbf{g}$ sets the attractor point in Equation 1, $\mathbf{k}_{pos}$ and $\mathbf{k}_{vel}$ are the gains for the PD-feedbackcontroller.

Variable	Value
$\mathbf{q}_1$	$[3.5, 4.4, -0.07, -0.5, -0.7, -1.5, -0.6, -1.0, 0.3, -0.4]$
$\mathbf{g}$	$[2.8, 4.5, -0.3, -1.8]$
$\mathbf{k}_{pos}$	$[632.5, 885.6, 463.4, 643.7]$
$\mathbf{k}_{vel}$	$[14.5, 13.2, 38.6, 40.6]$
$\mathbf{q}_{min}$	$[2.8, 2.8, -2.6, -2.6, -1.04]$
$\mathbf{q}_{max}$	$[4.7, 4.7, 0, 0, 1.0]$

Table II: Biped walker setting of pre-optimized quantities.

IV. Experiments and Results

A. Toy Example

To get used to the DMPs we start with an easy task and try to learn / fit a one-dimensional function, which is a superposition of sine and cosine oscillations.

$$y^*(t) = sin(2\pi t) + 0.25cos(4\pi t + 0.77) + 0.1sin(6\pi t + 3.0)$$

We want to learn the forcing function $f_{task}(\phi)$ for our model in Equation 1. In this simple scenario we can use a simple supervised learning approach (e. g. locally weighted regression) to fit the desired trajectory [3] or reinforcement learning. We briefly discuss both strategies.

1) Imitation learning/ curve fitting: The paramter vector $\boldsymbol{\omega}$ for each DMP is found by locally weighted regression [9]:

We want to minimize the weighted sum J_i for every dimension of $\boldsymbol{\omega}$, where our kernels $\psi_i(t)$ are used as weights too:

$$J_i = \sum_{t=1}^{T} \psi_i(t)(f_{task}(t) - \hat{f}(t;\omega_i))^2, \tag{6}$$

$$\omega_i = \frac{\boldsymbol{s}^T \Gamma_i \boldsymbol{f}_{task}}{\boldsymbol{s}^T \Gamma_i \boldsymbol{s}}, \tag{7}$$

where

$$\mathbf{s} = \begin{pmatrix} A \\ A \\ \dots \\ A \end{pmatrix} \quad \Gamma_i = \begin{pmatrix} \psi_i(1) & & & \\ & \psi_i(2) & & \\ & & \dots & \\ & & & \psi_i(T) \end{pmatrix},$$

$$\boldsymbol{f}_{task} = \begin{pmatrix} f_{task}(1) \\ f_{task}(2) \\ \dots \\ f_{task}(T) \end{pmatrix},$$

where A is the amplitude of our oscillator from Equ. 4.

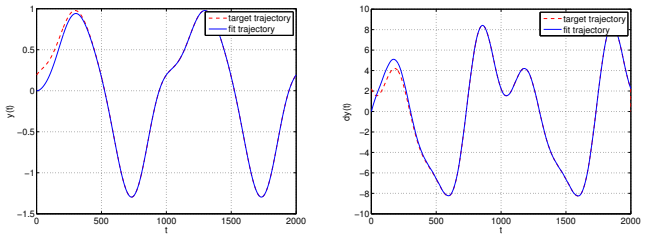


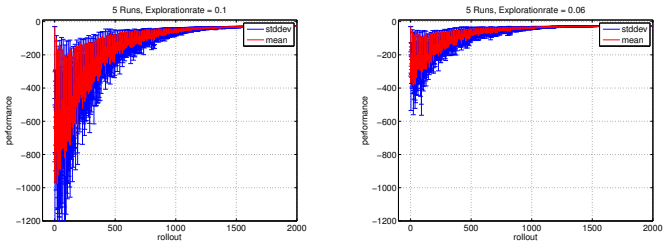
Figure 2: learned trajectory, $K = 20, \alpha_z = 4, \beta_z = 1, \tau = 0.1952s$. (left) y-position, (right) velocity

Figure 2 shows the result for imitation learning. The DMPs fit the trajectories very well, allthough the DMPs start at a different point ($y_{DMP}(0) = 0$).

2) reward based Learning: For the reinforcement learning approach we used the following cost function:

$$J(\boldsymbol{\alpha}) = \sum_{n=1}^{N} \alpha_0 (y^*(n) - \hat{y}(n;\boldsymbol{\omega}))^2 + \quad (8)$$
$$+\alpha_1 (dy^*(n) - d\hat{y}(n;\boldsymbol{\omega}))^2, \; with$$
$$\boldsymbol{\alpha} = \begin{bmatrix} \alpha_0 \\ \alpha_1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0.1 \end{bmatrix} \quad (9)$$

y^* represents the desired trajectory and $\hat{y}$ is the generated output from our DMP. N is the number of trajectory samples. Figure 4 and 3 show the different learning performances, if using the result of imitation learning or random initialization and using different explorationrates.



(a) (b)

Figure 3: with initial fit a)mean learning performance, 5 runs, $\lambda = 5$, b) mean learning performance, 5 runs, $\lambda = 5$

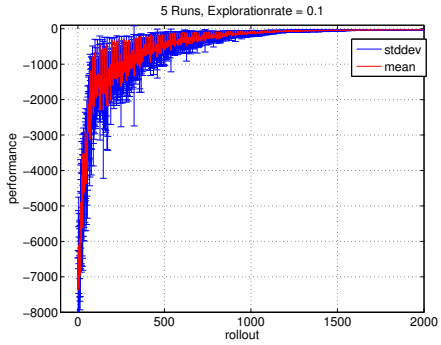


Figure 4: random initial parameter set

B. Five Link Walker

In this section we investigate a more difficult model. This model simulates a biped walker and consists of 5 links and 4 Joints, namely 2 hip joints and 2 knee joints. Each joint is represented by a single DMP, which all are sharing the same canonical system.

The parameters are optimized according to:

- progress in forward direction,
- stable time,

using the following cost function J :

$$J(\boldsymbol{\alpha}) = \sum_{n=1}^{N} \alpha_0 \cdot steplength + \quad (10)$$
$$+\alpha_1 \cdot (maxTime - stableTime)$$
$$+\alpha_2 \cdot x_{distance},$$
$$\boldsymbol{\alpha} = \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \end{bmatrix} = \begin{bmatrix} 0.6 \\ 0.2 \\ 0.1 \end{bmatrix}. \quad (11)$$

Figure 6 shows, that learning converges after about 600 roll-outs to the trajectories depicted in Figure 5 and Figure 7. Since the simulator switches the legs on every step, discontinuities occur in the joint angle trajectories, which are not very natural from a biological point of view. A Phase resetting mechanism on foot impact supports the walker in learning [4].

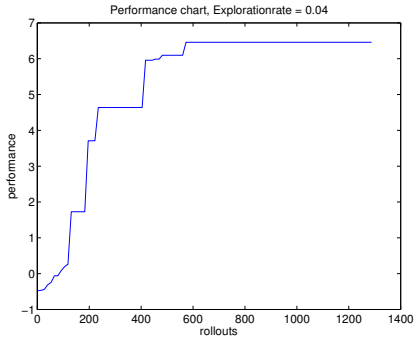


Figure 6: performance chart of walker with explorationrate 0.04 and 100 Iterations

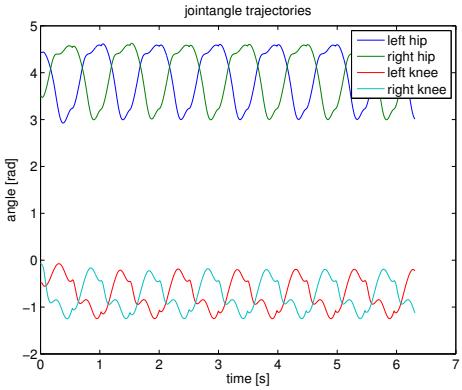
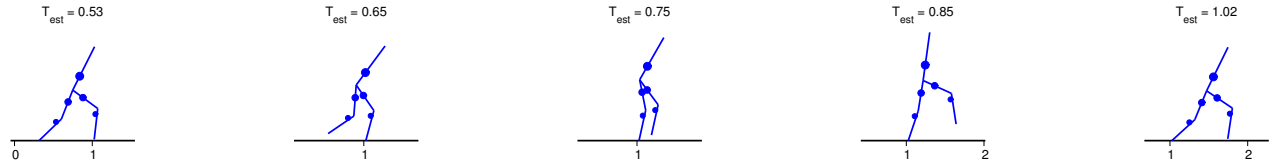


Figure 7: simulated trajectories of the biped walker

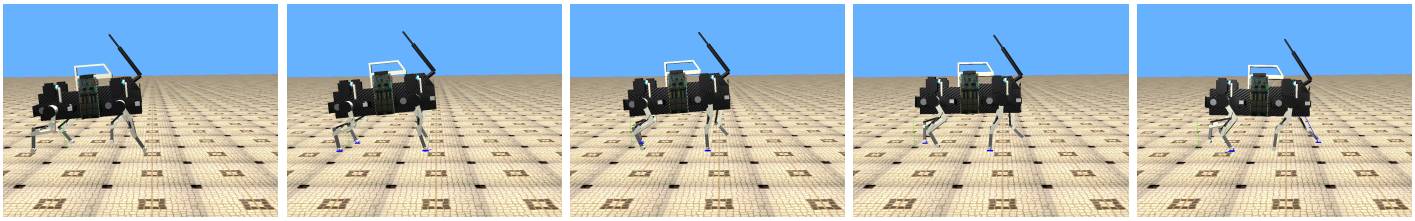
C. Oncilla Quadruped

The oncilla quadroped is similar to a house cat in size and weight. It can actuate hip ad-/abduction, pro-/retraction and knee flexion/extension. In the following experiments we are neglecting the actuation of ad-/abduction servos for simplicity, resulting in a 8-DoF robot platform. Due to the early development stage of the AMARSi Oncilla simulator, we could only use position control instead of torque control. Thus we had to use the already implemented gain's of the feedback controller.



(a) t=0.53 s (b) t=0.65 s (c) t=0.75 s (d) t=0.85 s (e) t=1.694 s

Figure 5: snapshots of walker sequence



(a) t=1.440 s (b) t=1.548 s (c) t=1.614 s (d) t=1.660 s (e) t=1.694 s

Figure 8: snapshots of trotting oncilla

1) simple trotting gait: If we look at dogs or other biologic quadroped, we can see that in the case of trotting the movement of the diagonal legs is equal,

$$\theta_{leftfore} = \theta_{righthind}, \quad (10)$$
$$\theta_{lefthind} = \theta_{rightfore}. \quad (11)$$

Therefore we can simplify our learning and restrict our parameter vector to learn only the parameters for the left_fore and right_fore legs and assign these values to the other legs according to Equation 10 and Equation11. Additionally, we can assume that the left_fore leg and the right_fore leg have a phase shift of π. Considering this assumptions we provide an initial solution with simple sinusoidal oscillators.

As reward function we implemented the following:

$$\begin{aligned} J(\boldsymbol{\alpha}) &= \alpha_0 \cdot simulationTime + \\ &\quad +\alpha_1 \cdot \sum_{n=1}^{N} (CoM_{height}(n) - desiredHeight)^2 \\ &\quad +\alpha_2 \cdot x_{distance}, \\ \boldsymbol{\alpha} &= \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \end{bmatrix} = \begin{bmatrix} -0.25e-4 \\ 0.4 \\ 1e-3 \end{bmatrix}. \end{aligned}$$

To speed up learning, each iteration step is aborted, if the robot is falling on the ground. Thus, the simulation time is a good measure for the stability of the gait. The CMA-ES learning method is stopped after 60 iterations due to time issues, but achieves already good results after about 40 iterations (~600 rollouts) shown in Fig. 9. Figure 10 shows the learned trajectories simulated on the robot. The oncilla robot achieves with this learned gait an average speed of $0.18\frac{m}{s}$.

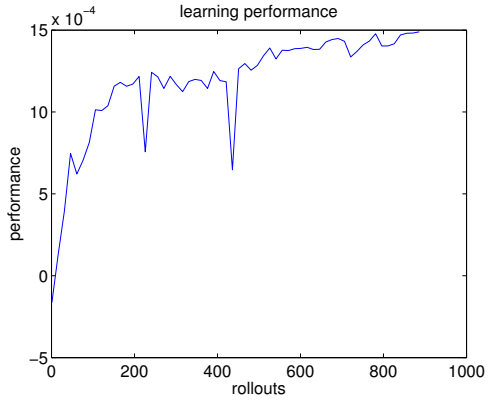


Figure 9: learning performance with $\lambda = 14$, explorationrate = 0.05

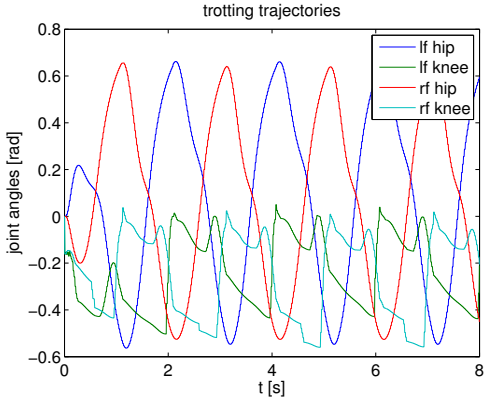


Figure 10: performed trajectories of the left and right for legs of the oncilla robot in a trotting gait

V. Conclusion and future work

In this project we used DMPs to perform gaits on the biped walker and the AMARSi oncilla robot. All experiments were performed on an Intel Core 2 Duo @ 2x2.8GHz. The experiments showed, that DMPs are quite useful. The walker was parameterized with 16 parameters and performed quite well after about 600 episodes.

The oncilla uses a parameter vector of 32 parameters, representing the left and right fore legs hip and knee joints, and using those for the two other legs as well. Due to a lack of time and the limited computation power on our system we restricted the CMA-ES learner to 60 iterations per run. The robot achieves with the learned trotting gait an average speed of $0.18\frac{m}{s}$ at a step frequency of 0.5 Hz. These are nice results for the begining and further steps will be, introducing feedback signals form the robot investigating perturbations and different environments (e.g. slopes, rough terrain, etc.), and to set up a new computer system to speed up learning. Due to some issues on the Oncilla hardware, which could not be fixed til the end of this project, we were not able to execute the learned gaits on the real robot, which is part of ongoing research.

Acknowledgements

I would like to thank my supervisor DI Elmar Rückert for supporting me in this project.

References

[1] Mostafa Ajallooeian, Soha Pouya, Alexander Sproewitz, and Auke Jan Ijspeert. Central pattern generators augmented with virtual model control for quadruped rough terrain locomotion. 2013.

[2] N. Hansen, S.D. Muller, and P. Koumoutsakos. Reducing the Time Complexity of the Derandomized Evolution Strategy with Covariance Matrix Adaptation (CMA-ES). *Evolutionary Computation*, 11(1):1–18, 2003.

[3] A. Ijspeert, J. Nakanishi, P Pastor, H. Hoffmann, and S. Schaal. dynamical movement primitives: learning attractor models formotor behaviors. (25):328–373, 2013.

[4] Jun Nakanishi, Jun Morimoto, Gen Endo, Gordon Cheng, Stefan Schaal, and Mitsuo Kawato. Learning from demonstration and adaptation of biped locomotion. *Robotics and Autonomous Systems*, 47:79–91, 2004.

[5] P. Pastor, H. Hoffmann, T. Asfour, and S. Schaal. learning and generalization of motor skills by learning from demonstration. In *international conference on robotics and automation (icra2009)*, 2009.

[6] J. Peters and S. Schaal. Reinforcement Learning of Motor Skills with Policy Gradients. *Neural Networks*, 21(4):682–697, 2008.

[7] Marc Raibert. BigDog, the Rough-Terrain Quadruped Robot. In Myung J. Chung, editor, *Proceedings of the 17th IFAC World Congress, 2008*, volume 17.

[8] S Schaal, P. Mohajerian, and A." Ijspeert. *dynamics systems vs. optimal control - a unifying view*, pages 425–445. Number 165. 2007.

[9] Stefan Schaal and Christopher G. Atkeson. Constructive incremental learning from only local information. *Neural Computation*, 10:2047–2084, 1997.

[10] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.