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 primitives (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 quadropered 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. Nevertheless, 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. 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 rough terrain. For very complex movement plans, a discretization into a sequence of much simpler movement primitivies seems, so far, to be the only chance to cope with this high complexity.

In this project we implement the CPG movement represen-tations 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 describes a second order dynamical system and is called the transformation system.

\[
\begin{align*}
\tau \ddot{z} &= \alpha_z (\beta_z (g - y) - z) + f_{\text{task}}(\phi), \\
\tau \dot{y} &= z.
\end{align*}
\]

The variables \(\alpha_z, \beta_z\) are time constants which corresponds to the spring- and damper- coefficients in dynamic mechanical system. \(\tau\) 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 \(\phi\) is a substitute for the time \(t\).

\[
\begin{align*}
\tau \dot{\phi} &= 1, \\
\phi &= \text{mod}(\phi, 2\pi).
\end{align*}
\]

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

The forcing term \(f_{\text{task}}\) is approximatly by a set of normalized radial basis function \(\psi_i(\phi)\) and weights \(\omega_i\)

\[
f_{\text{task}} = \frac{\sum_i^K \psi_i(\phi) w_i}{\sum_j^\infty \psi_j(\phi) A_i},
\]

\[
\psi_i(\phi) = \exp(h_i \cos(\phi - c_i) - 1).
\]

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

III. OPTIMIZATION FRAMEWORK

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

\[
\Theta^* = \arg \min_{\Theta} C(\Theta),
\]

which minimizes the expected costs

\[
C(\Theta) = \mathbb{E}[J(y, \dot{y}) | \Theta].
\]
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 \( \omega \) 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 \( \mu \) and covariance matrix \( \Sigma \) 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 Table I.

<table>
<thead>
<tr>
<th></th>
<th>DoF</th>
<th>( K )</th>
<th>dim of parameter vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>biped walker</td>
<td>4</td>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>oncilla</td>
<td>12</td>
<td>8</td>
<td>96</td>
</tr>
</tbody>
</table>

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

Due to the high dimensional continuous 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 \( g \) sets the attractor point in Equation 1, \( k_{pos} \) and \( k_{vel} \) are the gains for the PD-feedbackcontroller.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_1 )</td>
<td>(-0.5, 4.5, 3.7, -0.5, -0.5, 0.7, -1.0, -0.6, 0.5, 0.3, -0.1)</td>
</tr>
<tr>
<td>( g )</td>
<td>(2.8, 4.5, -0.3, -1.8)</td>
</tr>
<tr>
<td>( k_{pos} )</td>
<td>(632.5, 885.6, 463.4, 643.7)</td>
</tr>
<tr>
<td>( k_{vel} )</td>
<td>(14.5, 13.2, 38.6, 40.6)</td>
</tr>
<tr>
<td>( q_{pos} )</td>
<td>(2.8, 2.8, -2.6, -2.6, -1.04)</td>
</tr>
<tr>
<td>( q_{vel} )</td>
<td>(4.7, 4.7, 0.0, 0.1)</td>
</tr>
</tbody>
</table>

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

In the walker task we made use of the hand tuned values 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 \( \mu \) and covariance matrix \( \Sigma \) of the policy distribution to get closer to the (local) optimal solution.

### 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.

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

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 parameter vector \( \omega \) for each DMP is found by locally weighted regression [9]:

\[
J_i = \sum_{t=1}^{T} \psi_i(t)(f_{task}(t) - \tilde{f}(t; \omega_i))^2, \quad (6)
\]

\[
\omega_i = s^T \Gamma_i f_{task} / s^T \Gamma_i s, \quad (7)
\]

where

\[
s = \begin{pmatrix} A \\ A \\ \ldots \\ A \end{pmatrix}, \quad \Gamma_i = \begin{pmatrix} \psi_i(1) & \psi_i(2) & \ldots & \psi_i(T) \end{pmatrix}, \quad \tilde{f}(t; \omega_i) = \begin{pmatrix} f_{task}(1) \\ f_{task}(2) \\ \ldots \\ f_{task}(T) \end{pmatrix},
\]

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, although 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:

\[ J(\alpha) = \sum_{n=1}^{N} \alpha_0 (y^*(n) - \hat{y}(n; \omega))^2 + \alpha_1 (dy^*(n) - d\hat{y}(n; \omega))^2, \]

with \( \alpha = \begin{bmatrix} \alpha_0 \\ \alpha_1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0.1 \end{bmatrix} \) \( (9) \)

\( y^* \) 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 exploration rates.

\[ J(\alpha) = \sum_{n=1}^{N} \alpha_0 \cdot \text{steplength} + \alpha_1 \cdot (\text{maxTime} - \text{stableTime}) + \alpha_2 \cdot \text{distance}. \]

\( \alpha = \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \end{bmatrix} = \begin{bmatrix} 0.6 \\ 0.2 \\ 0.1 \end{bmatrix}. \)

Figure 6 shows, that learning converges after about 600 rollouts 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 3](image-url)  
(a) mean learning performance, 5 runs, \( \lambda = 5 \)  
(b) mean learning performance, 5 runs, \( \lambda = 5 \)

![Figure 4](image-url)  
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 \( J : \)

![Figure 6](image-url)  
performance chart of walker with exploration rate 0.04 and 100 Iterations

![Figure 7](image-url)  
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.
1) Simple trotting gait: If we look at dogs or other biology quadruped, we can see that in the case of trotting the movement of the diagonal legs is equal,

\[ \theta_{\text{left fore}} = \theta_{\text{right hind}}, \quad (10) \]
\[ \theta_{\text{left hind}} = \theta_{\text{right fore}}. \quad (11) \]

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 Equation 11. Additionally, we can assume that the left_fore leg and the right_fore leg have a phase shift of \( \pi \). Considering these assumptions we provide an initial solution with simple sinusoidal oscillators.

As reward function we implemented the following:

\[
J(\alpha) = \alpha_0 \cdot \text{simulationTime} + \alpha_1 \cdot \sum_{n=1}^{N} (\text{CoM}_{\text{height}}(n) - \text{desiredHeight})^2 + \alpha_2 \cdot x_{\text{distance}},
\]

\[ \alpha = \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \end{bmatrix} = \begin{bmatrix} -0.25e - 4 \\ 0.4 \\ 1e - 3 \end{bmatrix}. \]

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 m/s.
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 $0.18 \text{ m/s}$ at a step frequency of $0.5 \text{ Hz}$. These are nice results for the beginning 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


