Imitation Learning with DMP and PI^2

Urbaniak, Dominik

May 2020

1 Introduction

This report summarizes my achievements during the "Praktikum" where I reproduced the throwing task by Stulp and Raiola [4] in MATLAB and introduced new advances in DMP research [2]. First, I give some theoretical insights to the trajectory generation with DMP and the applied optimization method PI^2 and continue by explaining the way I implemented the throwing task.

2 DMP

"The idea of dynamic movement primitives (DMP) is to exploit well-known simple formulations", like a spring-damper system, "to code the basic behavioral pattern, and to use statistical learning to adjust the attractor landscape of the DMP to the detailed needs of the task" [3]. This generic approach allows modeling complex movements in real time [1]. Ijspeert et al. start with a spring-damper system,

$$\hat{\tau}\ddot{y} = \alpha_z(\beta_z(g - y) - \dot{y}) + f, \tag{1}$$

which is critically damped with $\beta_z = \alpha_z/4$ and with f defined as the learnable forcing term

$$f(t) = \frac{\sum_{i=1}^{N} \Psi_i(t) w_i}{\sum_{i=1}^{N} \Psi_i(t)}. \tag{2}$$

Here, Ψ represent N exponential basis functions with fixed centers and widths, scaled with adjustable weights w_i. The explicit time dependency complicates "the coordination of multiple degree-of-freedom in one dynamical system" [1]. Therefore, the time t is replaced by a canonical system

$$\hat{\tau}\dot{x} = -\alpha_x x, \tag{3}$$

where x is the phase variable and $\hat{\tau}$ the duration of the trajectory. During the approach of the goal g, x converges monotonically to zero. Time t and x can be

related by an exponential function (see equation 7). Finally, the forcing term is reformulated as

$$f(x) = \frac{\sum_{i=1}^{N} \Psi_i(x) w_i}{\sum_{i=1}^{N} \Psi_i(x)} x(g - y_0). \tag{4}$$

This formulation features invariance properties that allows to modify the learned trajectory (e.g the goal position) while preserving the general shape [1]. Recently, Ginesi, Sansonetto and Fiorini suggested improvements on DMP [2]. Mollifier-like basis functions are compactly supported, compared to Gaussian basis functions, and "allow the forcing term to be zero outside the support, thus guaranteeing the convergence to the goal position":

$$\varphi_i(x) = \begin{cases} \exp\left(-\frac{1}{1-|a_i(x-c_i)|^2}\right) & |a_i(x-c_i)| < 1 \\ 0 & otherwise \end{cases} \tag{5}$$

Here, a_i and c_i are the constant widths and centers of the N basis functions.

3 PI^2

Policy Improvement with Path Integrals (PI^2) is a policy-based RL algorithm that is based on stochastic optimal control [5]. The DMP provide the policies that are probabilisticly created with one exploration parameter ϵ applied on the weights w_i from equation (2). The desired behavior is encoded in a cost function $S(\tau_j)$ over M sample policies τ_j, where $j = 1, ..., M$ and the explored policies are assessed by their performance. PI^2 then calculates weights

$$W(\tau_j) = \exp\left(-c \frac{S(\tau_j) - \min S(\tau_j)}{\max S(\tau_j) - \min S(\tau_j)}\right) \tag{6}$$

with c=10, according to their performance.

4 Application

4.1 Description

The trajectory creation with DMP and the optimization via PI^2 is demonstrated in a throwing task. In a two dimensional space, a robot performs a trajectory in $\hat{\tau} = 1.43s$ holding a ball. After $t = 0.6s$ the ball is released. The desired goal position of the ball is at $G = (-0.7, -0.3)$. Figure 1 shows the given trajectory of the robot and the resulting trajectory of the ball in red. Hence, to reach G the robot needs to throw the ball further.

4.2 DMP Initialization

Given the initial trajectory with time steps t_s and $y, \dot{y}, \ddot{y} \in \mathbb{R}^2$, one first computes the inputs $x_0 \in \mathbb{R}^1$ and the outputs $f_0 \in \mathbb{R}^2$ for the basis function network.

$$x_0 = \exp \frac{-\frac{3}{2}\alpha_e t_s}{\hat{\tau}}, \tag{7}$$

where $\alpha_e = 4$. The output f_0 is derived by solving equation (1) for f. I use a mollifier-like basis function network for each of the two dimensions, both with the same input x_0 and respective $f_0^{(d)}$ where $d = 1, 2$. Calculating the weights of each network

$$w_i^{(d)} = \left((\varphi_i^{(d)})^T \varphi_i^{(d)} \right)^{-1} (\varphi_i^{(d)})^T f_0^{(d)}, \tag{8}$$

where w_i is an input for equation (2) and φ_i derived from equation (5).

4.3 DMP Optimization

Optimizing the DMP with PI^2 is an iterative process with P iterations. At the beginning, one chooses the number of time steps n and reproduces the initial trajectory. Then, one generates M sample trajectories from the current policy τ_p, calculates new weights $w_{i,p+1}$, hence creates an updated policy τ_{p+1} and repeats that procedure until the costs converge to their optimum. Samples are generated from a multivariate normal distribution $\mathcal{N}(w_i, \epsilon^2 I_N)$. The cost function takes into account the final distance y_n in dimension $d = 1$ to G and the acceleration $\ddot{y}$:

$$S_{dist}(\tau_j) = \begin{cases} |G_1 - y_{n,1}| - c_m & |G_1 - y_{n,1}| - c_m \geq 0 \\ 0 & otherwise, \end{cases} \tag{9}$$

where $c_m = 0.01$ is a small margin in which the landing site is optimal. Hence, the final cost function is defined as

$$S(\tau_j) = S_{dist}(\tau_j) + c_f \sum_{l=1}^{n} \ddot{y}_l, \tag{10}$$

where $c_f = 0.001$ is a weighting factor that assigns much higher priority to reaching the goal position. Equation (6) computes the weights associated with the costs to create the new policy at iteration step $p = 1, ..., P$, represented by the weights

$$w_{i,p+1} = \frac{\sum_{j=1}^{M} w_{i,p,j} W_p(\tau_j)}{\sum_{j=1}^{M} W_p(\tau_j)}. \tag{11}$$

4.4 Results

In ten consecutive runs, my implementation makes the cost converge to the optimum each time. Figure 1 illustrates the development of the trajectory

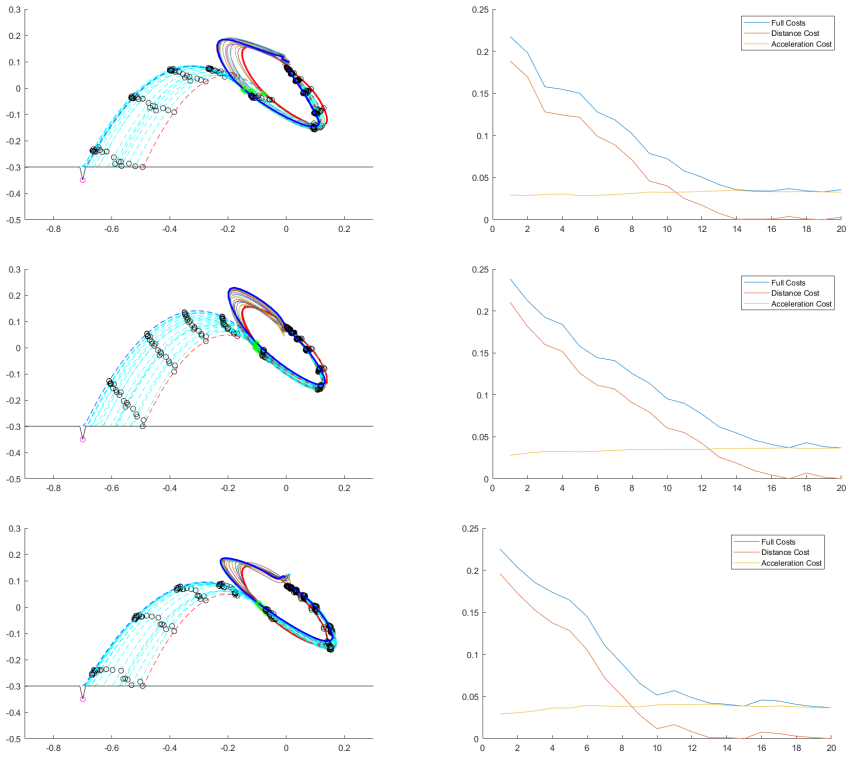

Figure 1: DMP optimization: Development of the trajectory and the cost in three runs

and its costs in three runs during $P = 20$ iterations. The thick red and blue lines show the initial and optimized trajectory of the spring-damper-system, respectively. The dashed lines represent the trajectory of the ball. One can see that costs of the distance converges to zero within twelve to seventeen iterations. The costs for the acceleration increase slightly, since throwing the ball further requires more effort. All trajectories are similar to the original, however, due to the random exploration with $\epsilon = 3.5$ the shape of the learned trajectory varies.

References

[1] Auke Jan Ijspeert, Jun Nakanishi, Heiko Hoffmann, Peter Pastor, and Stefan Schaal. Dynamical movement primitives: learning attractor models for motor behaviors. *Neural computation*, 25(2):328–373, 2013.

[2] Michele Ginesi, Nicola Sansonetto, Paolo Fiorini. DMP++: Overcoming Some Drawbacks of Dynamic Movement Primitives. https://arxiv.org/abs/1908.10608, 2019. Online; accessed 06 May 2020.

[3] Stefan Schaal, Jan Peters, J. Nakanishi, and A.J. Ijspeert. Control, planning, learning, and imitation with dynamic movement primitives. *Workshop on Bilateral Paradigms on Humans and Humanoids, IEEE Int. Conf. on Intelligent Robots and Systems, Las Vegas, NV*, 01 2003.

[4] Freek Stulp and Gennaro Raiola. Dmpbbo: A versatile python/c++ library for function approximation, dynamical movement primitives, and black-box optimization. *Journal of Open Source Software*, 2019.

[5] Evangelos Theodorou, Jonas Buchli, and Stefan Schaal. Learning policy improvements with path integrals. *Journal of Machine Learning Research - Proceedings Track*, 9:828–835, 01 2010.