# An Example Application of Policy Improvement with Path Integrals ($PI^2$)

**Stefan Schaal, Evangelos Theodorou, Jonas Buchli, Freek Stulp**

Wednesday, June 9, 2010

http://www-clmc.usc.edu sschaal@usc.edu

## 1 The $PI^2$ Algorithm Applied to Dynamic Movement Primitives

This is a simple example of how to use $PI^2$ (Theodorou, Buchli, & Schaal 2010) to optimize a 2nd order system with basis functions. We use the Dynamic Movement Primitive (DMP) framework as a convenient example of such a system (Ijspeert, Nakanishi, Pastor, Hoffman, & Schaal submitted, Ijspeert, Nakanishi, & Schaal 2003). We consider the following general dynamical system:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) + \mathbf{g}(\mathbf{x})^T(\theta + \epsilon)$$
$$\text{or} \qquad (1)$$
$$\dot{\mathbf{x}}_t = \mathbf{f}_t + \mathbf{g}_t^T(\theta + \varepsilon_t)$$

where $\mathbf{g}(\mathbf{x})$ is a collumn vector, and the controls are interpreted as a constant parameter vector $\theta$. DMPs are exactly in this form:

$$\begin{aligned}
\tau\dot{z} &= \alpha_z(\beta_z(g-y)-z)+f \\
\tau\dot{y} &= z \\
f &= \frac{\sum_{i=1}^{N}\psi_i(x)\theta_i}{\sum_{i=1}^{N}\psi_i(x)}x(g-y_0) \qquad (2)\\
\psi_i(x) &= \exp\left(-\frac{1}{2\sigma_i}(x-c_i)^2\right) \\
\tau\dot{x} &= -\alpha_x x
\end{aligned}$$

More details on DMPs are in (Ijspeert et al. submitted, Ijspeert et al. 2003). For $PI^2$ learning, the linear parameterization in $\theta$ is among the most important features, i.e., from the viewpoint of Eqn. (1), we have:

$$\begin{aligned}
f_t &= \alpha_z(\beta_z(g-y)-z) \\
\mathbf{g}_t &= \frac{x(g-y_0)}{\sum_{i=1}^{N}\psi_i(x)}\begin{bmatrix}\psi_1(x)\\ \vdots \\ \psi_N(x)\end{bmatrix} \qquad (3)
\end{aligned}$$

and only the differential equation in $\dot{z}$ matters for PI$^2$ learning.

For such a system, the pseudo code for PI2 becomes:

- **Given**:
  - An immediate cost function $r_t = q_t + \frac{1}{2}\theta_t^T \mathbf{R}\theta_t$
  - A terminal cost $\phi_{t_N}$
  - A stochastic parameterized policy $\mathbf{a}_t = \mathbf{g}_t^T(\theta + \varepsilon_t)$
  - The basis function $\mathbf{g}_t$ are known
  - The variance $\Sigma_\varepsilon$ of the mean-zero noise $\varepsilon_t$
  - The initial parameter vector $\theta$
- **Repeat** until convergence of the trajectory cost $R$:
  - Create $K$ roll-outs of the system from the same start state $\mathbf{x}_0$ using stochastic parameters $\theta + \varepsilon_t$ at every time step
  - **For** all $K$ roll-outs, compute:
    - $P(\tau_{i,k}) = \dfrac{\exp\left(-\frac{1}{\lambda}S(\tau_{i,k})\right)}{\sum_{m=1}^{K}\exp\left(-\frac{1}{\lambda}S(\tau_{i,m})\right)}$
    - $S(\tau_{i,k}) = \phi_{t_N,k} + \sum_{j=1}^{N-1} q_{t_j,k} + \sum_{j=i+1}^{N-1}\left(\theta + \mathbf{M}_{t_j,k}\varepsilon_{t_j,k}\right)^T \mathbf{R}\left(\theta + \mathbf{M}_{t_j,k}\varepsilon_{t_j,k}\right)$
    - $\mathbf{M}_{t_j,k} = \dfrac{\mathbf{R}^{-1}\mathbf{g}_{t_j,k}\mathbf{g}_{t_j,k}^T}{\mathbf{g}_{t_j,k}^T\mathbf{R}^{-1}\mathbf{g}_{t_j,k}}$
  - **For** all $i$ time steps, compute:
    - $\delta\theta_{t_i} = \sum_{k=1}^{K} P(\tau_{i,k})\mathbf{M}_{t_i,k}\varepsilon_{t_i,k}$
  - $[\delta\theta]_j = \dfrac{\sum_{i=0}^{N-1}(N-i)w_{j,t_i}\left[\delta\theta_{t_i}\right]_j}{\sum_{i=0}^{N-1}(N-i)w_{j,t_i}}$
  - Update $\theta \leftarrow \theta + \delta\theta$
  - Create one noiseless rollout to check the trajectory cost $R = \phi_{t_N} + \sum_{i=0}^{N-1} r_{t_i}$. In case the noise cannot be turned off, i.e., the control system is inherently stochastic, multiple roll-outs need to be averaged to assess the cost improvement of the parameter update.

## 2 Application Example

The example provided in the Matlab programs is a very simple 2D application of PI$^2$ to a two dimensional point mass control system:

$$\begin{aligned}
&\ddot{\mathbf{q}} = \frac{1}{m}\left(-b\dot{\mathbf{q}} + \mathbf{u}\right) \\
&\mathbf{u} = m\ddot{\mathbf{q}}_d + b\dot{\mathbf{q}} + k_P\left(\mathbf{q}_d - \mathbf{q}\right) + k_D\left(\dot{\mathbf{q}}_d - \dot{\mathbf{q}}\right) \\
&\text{where} \\
&\mathbf{q}_d = \mathbf{y}, \dot{\mathbf{q}}_d = \dot{\mathbf{y}}, \ddot{\mathbf{q}}_d = \ddot{\mathbf{y}} \text{ from the DMPs}
\end{aligned} \tag{4}$$

Thus, we use a PD controller with inverse dynamics control term for control, and the motor primitives realize the desired trajectory. Note that this is a perfect tracking controller, i.e., the controller will accurately track the desired trajectory. Learning the optimized motor primitives is therefore the only interesting component. But one may consider replacing this control system with more complicated and/or stochastic systems, or using a less proficient controller. All these changes would make the control component more interesting.

The Matlab code provides two optimization examples. In both cases, the nominal movement is from $\mathbf{q} = [0,0]^T$ to $\mathbf{q} = [1,1]^T$. In the acc2.m cost function, the cost is:

$$\begin{aligned}
&r_t = 0.5\ddot{\mathbf{q}}_t^T\ddot{\mathbf{q}}_t Q + 0.5\theta^T\theta R \\
&\text{with} \\
&Q = 1000,\ R = 1
\end{aligned} \tag{5}$$

I.e., the main objective is the minimize the squared acceleration of the point mass, while keeping the norm of the parameter vector short.

In the viapoint.m cost function, the cost is:

$$\begin{aligned}
&r_t = 0.5\ddot{\mathbf{q}}_t^T\ddot{\mathbf{q}}_t Q + 0.5\theta^T\theta R \\
&\text{at t=0.25s: } r_t \leftarrow r_t + 10000000000\left(\mathbf{q} - [0.5\ 0.2]^T\right)^T\left(\mathbf{q} - [0.5\ 0.2]^T\right) \\
&\text{with} \\
&Q = 1000,\ R = 1
\end{aligned} \tag{6}$$

This is the cost function as before, but with a very strong discontinuous penalty for passing through the point [0.5 0.2] at time t=0.25 seconds.

## 3 Remarks on Using PI$^2$

We put the most important parameters of PI$^2$ learning in the protocol_xxx.txt files, such that experimentation with our Matlab code can be done with minimal programming. Here is the example for the protocol_acc2.txt file:

```
% The protocol file has the following columns:
%
% start_x start_y goal_x goal_y duration std repetitions cost_function updates
%
% where
% start_x, start_y: the start position of the DMP
% goal_x, goal_y  : the goal position of the DMP
% duration        : the duration of the DMP
% std             : standard deviation of added noise for parameters
% repetitions     : repetitions of movement before learning update
% cost_function   : name of the cost function to use
% updates         : number of PI2 updates to perform
% basis_noise     : only add noise to the max active basis function
% n_reuse         : number of re-used trials per update

0  0  1  1  0.5  20  10  acc2  100  1  5
```

- The *std* variable is the standard deviation of the exploration noise. This is the only open algorithmic parameter in $PI^2$. One reasonable way of setting it is to look at several noisy realizations of roll-outs, and assess graphically whether the noise is sufficient to explore around the noiseless trajectory. Other factors may be important, e.g., for robotics, the noise should be kept small to avoid wear-and-tear on the robot. The plots generated by the Matlab programs are helpful to assess the exploration noise.
- The *repetitions* parameter stands for the number of roll-outs before an update of the learning parameters is performed. Note that the minimum is "2" for $PI^2$ – otherwise the algorithm cannot create reasonable updates. But there is no danger of numerical instabilities when using a small number of roll-outs per update.
- The *cost_function* is for a Matlab function within the path of Matlab. This function implements how the cost is calculated. It is really the design of the cost function that one spends most of the time on in RL implementations with $PI^2$ — this design process is beyond the $PI^2$ algorithm, which assumes that the cost function is given.
- The *updates* parameter simply specifies after how may updates the program should terminate.
- Setting *basis_noise* to "1" means that noise is only added to the most active kernel of the DMP nonlinearity, and that the noise is kept constant as long as this kernel is active. This procedure is equivalent that noise is only added to one of the DMP parameters at a time – the other parameters have zero noise perturbations. From our experience, this procedure speeds up convergence of $PI^2$ significantly — this trick has been used in man policy gradient methods before.
- The *n_reuse* parameter specifies how many roll-outs from the previous update should be kept in memory and re-used for the next update. This re-use is not needed in $PI^2$, but it is helpful if one wishes to operate with a minimal number of roll-outs per update. For instance, assume we only perform two roll-outs to compute the next parameter update. One can easily imagine that the

random noise exploration creates two new roll-outs that are both much worse than the previously computed best parameter setting. Thus, the it may happen that the cost of a trajectory actually increases in this way. Keeping a few of the best trajectories from the previous roll-outs around helps assuring that the cost of trajectories rarely goes up. These "re-used" trajectories are re-assessed in terms of the their cost and probabilities according to the latest parameter values — this is essentially an importance sampling method.

The attractor dynamics of DMPs reduce the magnitude of the basis functions $\mathbf{g}_t$ as the system approaches the goal state. This fact implies that exploration noise at the end of a trajectory has a reduced effect on the trajectory variations. Most of the time, this property is useful. If exploration noise is supposed to be more effective through the trajectory, try to initialize the DMPs with `dcp('init',i,n_rfs,sprintf('pi2_dmp_%d',i),1)` (see line 29 of our Matlab code, and note that the last parameter is "1" instead of "0"). This change uses a 2nd order canonical system in the DMPs, which creates stronger noise perturbations as the basis functions $\mathbf{g}_t$ decay more slowly towards the end of a trajectory. For this alternative DMP, the parameters $\theta$ will be of smaller magnitude, which will potentially require rescaling the cost function.

## 4 Summary

PI$^2$ is really easy to apply to RL problems — after the exploration noise is set, the user normally entirely focuses on cost function design.

It should be noted that the DMP representation for RL with parameterized policies has most likely several advantages, which facilitates PI$^2$ learning, but also the application of other RL algorithms. Of course, PI$^2$ can be applied to many other parameterized systems. But these alternative parameterization may need to be studied carefully in how far they facilitate learning or make it more complicated. Good presentations matter tremendously, a statement that is true for all machine learning problems.

If you have problems with PI$^2$ and the give Matlab code, please contact Stefan at sschaal@usc.edu. Some other fun PI$^2$ applications can be found in papers on our website http://www-clmc.usc.edu/publications .

## 5 References


Ijspeert, A., Nakanishi, J., Pastor, P., Hoffman, H. & Schaal, S. (submitted) Learning nonlinear dynamical systems. *Neural Computation*.

Ijspeert, A., Nakanishi, J. & Schaal, S. (2003). *Learning attractor landscapes for learning motor primitives.* Paper presented at the Advances in Neural Information Processing Systems 15.

Theodorou, E., Buchli, J. & Schaal, S. (2010) Reinforcement learning in high dimensional state spaces: A path integral approach. *Journal of Machine Learning Research 2010*(11): 3137-3181. http://www-clmc.usc.edu/publications/T/theodorou-JMLR2010.pdf