# Inverse Reinforcement Learning with PI$^2$

Mrinal Kalakrishnan, Evangelos Theodorou, and Stefan Schaal
Department of Computer Science, University of Southern California
Email: {kalakris, etheodor, sschaal}@usc.edu

## Introduction

We present an algorithm that recovers an unknown cost function from expert-demonstrated trajectories in continuous space. We assume that the cost function is a weighted linear combination of features, and we are able to learn weights that result in a cost function under which the expert demonstrated trajectories are optimal. Unlike previous approaches [1], [2], our algorithm does not require repeated solving of the forward problem (i.e., finding optimal trajectories under a candidate cost function). At the core of our approach is the PI$^2$ (**P**olicy **I**mprovement with **P**ath **I**ntegrals) reinforcement learning algorithm [3], which optimizes a parameterized policy in continuous space and high dimensions. PI$^2$ boasts convergence that is an order of magnitude faster than previous trajectory-based reinforcement learning algorithms on typical problems. We solve for the unknown cost function by enforcing the constraint that the expert-demonstrated trajectory does not change under the PI$^2$ update rule, and hence is locally optimal.

## PI$^2$: Policy Improvement with Path Integrals

We represent all trajectories as Dynamic Movement Primitives (DMPs) [4], which are parameterized policies expressed by a set of differential equations. The DMP representation is advantageous as it guarantees attractor properties towards the goal while remaining linear in its parameters $\boldsymbol{\theta}$, thus facilitating learning [5].

The PI$^2$ algorithm arises from a solution of the stochastic Hamilton-Jacobi-Bellman (HJB) equation using the Feynman-Kac theorem, which transforms the stochastic optimal control problem into the approximation problem of a path integral. Detailed derivations are omitted due to lack of space; we refer the reader to a previous publication [3]. We consider cost functions of the following form:

$$Q(\boldsymbol{\tau}) = \phi_{t_N} + \sum_{i=0}^{N-1} (q_t + \mathbf{u}_t^{\mathrm{T}} \mathbf{R} \mathbf{u}_t) dt \tag{1}$$

where $Q(\boldsymbol{\tau})$ is the cost of a trajectory $\boldsymbol{\tau}$, $\phi_{t_N}$ is the terminal cost, $N$ is the number of time steps, $q_t$ is an arbitrary state-dependent cost function, $\mathbf{u}_t$ are the control commands at time $t$, and $\mathbf{R}$ is the positive definite weight matrix of the quadratic control cost. PI$^2$ updates the parameters of a DMP with parameters $\boldsymbol{\theta}$, using $K$ trajectories sampled from the same start state, with stochastic parameters $\boldsymbol{\theta} + \boldsymbol{\epsilon}_t$ at every time step. The update equations are as follows:

$$P(\boldsymbol{\tau}_{i,k}) = \frac{e^{-\frac{1}{\lambda} S(\boldsymbol{\tau}_{i,k})}}{\sum_{m=1}^{K} e^{-\frac{1}{\lambda} S(\boldsymbol{\tau}_{i,m})}} \tag{2}$$

$$\delta \boldsymbol{\theta}_{t_i} = \sum_{k=1}^{K} P(\boldsymbol{\tau}_{i,k}) \mathbf{M}_{t_i,k} \boldsymbol{\epsilon}_{t_i,k} \tag{3}$$

$$S(\boldsymbol{\tau}_{i,k}) = \phi_{t_N,k} + \sum_{j=i}^{N-1} q_{t_j,k} dt + \frac{1}{2} \sum_{j=i+1}^{N-1} (\boldsymbol{\theta} + \mathbf{M}_{t_j,k} \boldsymbol{\epsilon}_{t_j,k})^{\mathrm{T}} \mathbf{R} (\boldsymbol{\theta} + \mathbf{M}_{t_j,k} \boldsymbol{\epsilon}_{t_j,k}) dt \tag{4}$$

$$\mathbf{M}_{t_j,k} = \frac{\mathbf{R}^{-1} \mathbf{g}_{t_j,k} \mathbf{g}_{t_j,k}^{\mathrm{T}}}{\mathbf{g}_{t_j,k}^{\mathrm{T}} \mathbf{R}^{-1} \mathbf{g}_{t_j,k}} \tag{5}$$

where $k$ indexes over the $K$ sampled trajectories used for the update, $i$ indexes over all time-steps, $P(\boldsymbol{\tau}_{i,k})$ is the discrete probability of trajectory $\boldsymbol{\tau}_k$ at time $t_i$, $S(\boldsymbol{\tau}_{i,k})$ is the cumulative cost of trajectory $\boldsymbol{\tau}_k$ starting from time $t_i$, $\delta \boldsymbol{\theta}_{t_i}$ is the update to the DMP parameter vector at time $t_i$, $\phi_{t_N,k}$ is the terminal cost incurred by trajectory $\boldsymbol{\tau}_k$, $q_{t_j,k}$ is the cost incurred by trajectory $\boldsymbol{\tau}_k$ at time $t_j$, $\boldsymbol{\epsilon}_{t_j,k}$ is the noise applied to the parameters of trajectory $\boldsymbol{\tau}_k$ at time $t_j$, and $\mathbf{g}_{t_j,k}$ are the basis functions of the DMP function approximator for trajectory $\boldsymbol{\tau}_k$ at time $t_j$.

## Inverse Reinforcement Learning with PI$^2$

We assume that the arbitrary state-dependent cost function $q_t$ is linearly parameterized as:

$$q_t = \boldsymbol{\beta}_q^{\mathrm{T}} \boldsymbol{\psi}_t \tag{6}$$

where $\boldsymbol{\psi}_t$ are the feature values at time $t$, and $\boldsymbol{\beta}_q$ are the weights associated with the features, which we are trying to learn. Due to the coupling of exploration noise and command cost in the path integral formalism, we assume that the the shape of the quadratic control cost matrix $\mathbf{R}$ is given, and we can only learn its scaling factor. We also need to learn the scaling factor on the terminal cost:

$$\mathbf{R} = \beta_{\mathbf{R}} \hat{\mathbf{R}}; \qquad \phi_{t_N} = \beta_{\phi_{t_N}} \psi_{t_N} \tag{7}$$

where $\beta_{\mathbf{R}}$ is the control cost scaling factor that needs to be learned, $\hat{\mathbf{R}}$ is the shape of the control cost, $\beta_{\phi_{t_N}}$ is the scaling of the terminal cost, and $\psi_{t_N}$ is a binary feature which is 1 when the terminal state is reached, and 0 otherwise.

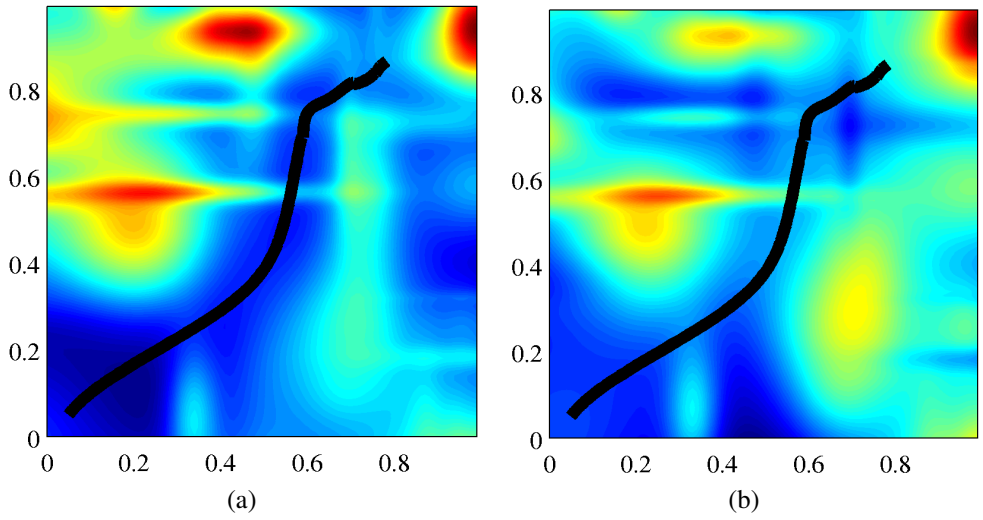


Fig. 1. (a) A cost function in a 2-D state space, along with a demonstrated low-cost trajectory between two points. The cost function is generated as a randomly weighted combination of some underlying features. (b) The cost function learnt from the single demonstrated trajectory.

We can rewrite Eqn. 4, the cumulative cost, as follows:

$$S(\boldsymbol{\tau}_{i,k}) = \boldsymbol{\beta}^{\mathrm{T}} \begin{bmatrix} \sum_{j=i}^{N-1} \boldsymbol{\psi}_{\boldsymbol{t_j},\boldsymbol{k}} \\ \frac{1}{2}\sum_{j=i+1}^{N-1} (\boldsymbol{\theta} + \mathbf{M}_{t_j,k}\boldsymbol{\epsilon}_{t_j,k})^{\mathrm{T}} \hat{\mathbf{R}} (\boldsymbol{\theta} + \mathbf{M}_{t_j,k}\boldsymbol{\epsilon}_{t_j,k}) \\ \psi_{t_N,k} \end{bmatrix} = \boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{\Phi} \quad (8)$$

where $\boldsymbol{\beta} = [\boldsymbol{\beta}_q \quad \beta_{\mathbf{R}} \quad \beta_{\phi_{t_N}}]^{\mathrm{T}}$ is the weight vector to be learned, and $\boldsymbol{\Phi}$ are sufficient statistics that can be computed for every time-step for each sample trajectory.

The expert-demonstrated trajectory can be considered locally optimal under a particular cost function if the updates to its parameters using the PI$^2$ update rule are 0. In order to achieve the inverse, i.e., solve for the cost function under which the expert-demonstrated trajectory is locally optimum, we minimize the squared magnitude of trajectory updates, with an additional regularization term on the weight vector:

$$\min_{\boldsymbol{\beta}} \sum_{i=0}^{N-1} w_i^2 (\delta\boldsymbol{\theta}_{t_i})^{\mathrm{T}} (\delta\boldsymbol{\theta}_{t_i}) + \gamma \boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{\beta} \quad (9)$$

$$w_i = \frac{N-i}{\sum_{i=0}^{N-1}(N-i)} \quad (10)$$

where $\gamma$ is a scalar regularization factor. The weights $w_i$ assign a weight to the update for each time step, equal to the number of time steps left in the trajectory. This gives early points in the trajectory a higher weight, which is useful since their parameters affect a larger time horizon. Other weighting schemes could be used, but this one was experimentally found to give good learning results [3].

The solution of this optimization problem yields a weight vector $\boldsymbol{\beta}$, which produces the cost function we seek. We optimize this objective function using the Matlab non-linear least squares solver, and are currently investigating alternate, more efficient ways of performing this optimization.

## EVALUATION

We generated 5 random features in a 2-D state space, each feature being a sum of many Gaussians with random centers and variances. A random weighting of these features results in a cost function, shown in Fig. 1(a). A low-cost trajectory between two points is then demonstrated in this state space, also seen in Fig. 1(a). We then sampled 50 trajectories around this expert-demonstrated trajectory, and solved the optimization problem in Eqn. 9 to obtain the weights $\boldsymbol{\beta}$, and the resulting cost function as seen in Fig. 1(b). There is an excellent match between the true cost function and the learned one, demonstrating the effectiveness of our approach.

These results are to be considered preliminary. Efficient algorithms for optimizing Eqn. 9, systematic evaluations of learning performance, comparisons with other inverse reinforcement learning algorithms, and applications to real-world problems are left for future work.

## REFERENCES

[1] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proceedings of the twenty-first international conference on Machine learning*, 2004.

[2] N. Ratliff, D. Silver, and J. Bagnell, "Learning to search: Functional gradient techniques for imitation learning," *Autonomous Robots*, vol. 27, pp. 25–53, July 2009.

[3] E. A. Theodorou, J. Buchli, and S. Schaal, "Reinforcement learning of motor skills in high dimensions: A path integral approach," in *International Conference of Robotics and Automation (accepted)*, 2010. Available at `http://www-clmc.usc.edu/publications/E/PI2.pdf`.

[4] A. J. Ijspeert, J. Nakanishi, and S. Schaal, "Learning attractor landscapes for learning motor primitives," *Advances in neural information processing systems*, pp. 1547–1554, 2003.

[5] J. Peters and S. Schaal, "Reinforcement learning of motor skills with policy gradients," *Neural Networks*, vol. 21, no. 4, pp. 682–697, 2008.

**Topic: learning algorithms; Preference: oral/poster**