# Gaussian Processes for Data-Efficient Learning in Robotics and Control

by Deisenroth, Fox, Rasmussen; pres. by Emmanuel Sales

2020-12-02

## Context

We want to **learn complex dynamics models** to achieve an optimal policy $\pi(x) = u$ for reaching a target state.

- Reinforcement learning: based on repeated trials, exploration, reward evaluations.
  - Problem: RL is very slow, with a lot of data that needs to be collected for even simple problems.
- Model-based methods could help with efficiency problems by providing additional information about the system.
  - Problem: model-based methods are subject to uncertainty, and assuming that uncertainty away isn't going to produce good results.

## Enter PILCO

PILCO (Probabilistic Inference for Learning Control) is a model-based policy search method.

- Based on Gaussian processes and gradient-based policy optimization. GPs allow modelling of uncertainty for each possible input point.
- Computationally efficient deterministic inference that's more **data-efficient** than the existing RL methods.

## PILCO framework and Gaussian process overview

PILCO is based on **taking training input data** and **learning optimal GP models based on them**.

Very general dynamical system model

$$x_{t+1} = f(x_t, u_t) + w, \quad w \sim \mathcal{N}(0, \Sigma_w)$$

## Gaussian process introduction

A GP is specified by a *mean function* $m(z)$ and a *covariance kernel* $k(z_1, z_2)$. For every $z$ in the domain of our function we model our function by

$$f(z) \sim \mathcal{N}(m(z), \Sigma_k(z, Z))$$

In PILCO our input space is both state and control space $\tilde{x} = (x, u) \in \mathbb{R}^{D+F}$ and they use the kernel

$$k(\tilde{x}_p, \tilde{x}_q) = \sigma_f^2 \exp\left(-\frac{1}{2}(\tilde{x}_p - \tilde{x}_q)^T \Lambda^{-1}(\tilde{x}_p - \tilde{x}_q)\right) + \delta_{pq}\sigma_w^2$$

GPs are trained in a Bayesian manner: the mean function and covariance matrix are updated as you feed the model training data $(Z)$.

## Gaussian processes visualized

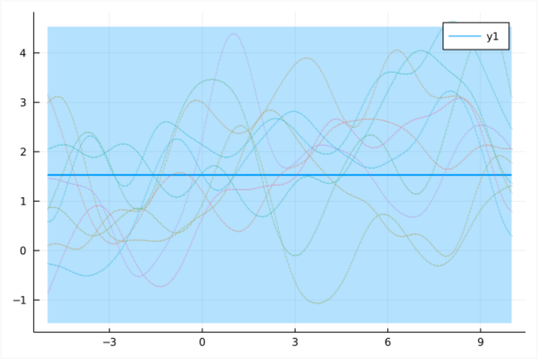


**Figure 1:** example of a "prior" Gaussian process (before training). This includes 10 samples of the function.

## Gaussian processes visualized

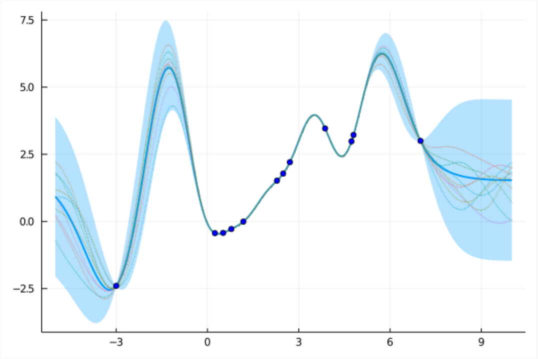


**Figure 2:** example of a "posterior" Gaussian process (after training). Notice that areas far away from training points have high variance, function at and near training points has very low variance

## Gaussian Processes in PILCO prediction model

We form a general one-step GP-based prediction model between $x_t, u_t$ and $x_{t+1}$.

For PILCO we have the following dynamics model:

$$\mu_{t+1} = x_t + \mathbb{E}[\Delta_t], \quad \Sigma_{t+1} = \mathrm{Var}_f[\Delta_t]$$

Values of the GP at $\tilde{x}_t = (x_t, u_t)$ are inferred via conditional Gaussian laws and training data $\tilde{X}, y$:

$$\mathbb{E}_f[\Delta_t] = m_f(\tilde{x}_t) = k(\tilde{X}, \tilde{x}_t)^T (K(\tilde{X}) + \sigma_w^2 I)^{-1} y$$

$$\mathrm{Var}_f[\Delta_t] = k(\tilde{x}_t, \tilde{x}_t) - k(\tilde{X}, \tilde{x}_t)^T (K(\tilde{X}) + \sigma_w^2 I)^{-1} k(\tilde{X}, \tilde{x}_t)$$

## PILCO: policy iteration

Given our one-step prediction model, PILCO then needs to search for a parameterized *policy* $\pi(x, \theta)$ that minimizes the expected long term cost

$$J^{\pi}(\theta) = \sum_{t=0}^{T} \mathbb{E}_{x_t}[c(x_t)], \quad x_0 \sim \mathcal{N}(\mu_0, \Sigma_0)$$

$\theta$ are parameters of the policy, for a simple example take PID coefficients.

## Long-term predictions

To calculate $J^\pi$ we need to iteratively compute $p(x_1|\theta), \dots, p(x_T|\theta)$.

$$p(x_{t+1}|\theta) = \iiint p(x_{t+1}|x_t, u_t) p(x_t, u_t|\theta) df dx_t du_t$$

Problem: especially over longer time horizons, this distribution is very hard to compute.

## Gaussian approximations

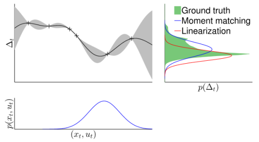


Fig. 2. GP prediction at an uncertain input. The input distribution $p(\boldsymbol{x}_t, \boldsymbol{u}_t)$ is assumed Gaussian (lower left panel). When propagating it through the GP model (upper left panel), we obtain the shaded distribution $p(\boldsymbol{\Delta}_t)$, upper right panel. We approximate $p(\boldsymbol{\Delta}_t)$ by a Gaussian (upper right panel), which is computed by means of either moment matching (blue) or linearization of the posterior GP mean (red). Using linearization for approximate inference can lead to predictive distributions that are too tight.

We use Gaussian approximation on each time step in order to make the computation tractable. Two approaches for doing this:

- Moment matching - assume joint Gaussianity between input and prediction, compute mean and covariance of predictive distribution in closed form
- Linearization - linearize the nonlinear GP function at the mean of the input distribution

## Policy improvement

We now need to improve the policy. Calculate gradients $\frac{\partial J^\pi(\theta)}{\partial \theta}$ of the policy.

$$\frac{dJ^\pi}{d\theta} = \sum_{t=1}^{T} \frac{d\mathbb{E}[c_t]}{d\theta}$$

and so forth... (very long chain)

Afterwards, use a method like gradient descent or BFGS to find the optimum $\theta$.

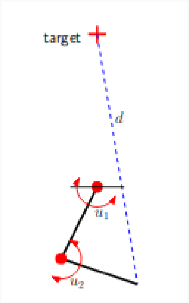
## PILCO summary

High level version of the algorithm:

```
1: procedure PILCO(initial parameter $\theta_0$, input data $x$)
2:     repeat
3:         learn GP dynamics model using all data
4:         repeat
5:             Calculate approximated $J^\pi(\theta)$
6:             Improve policy using a gradient method based on $dJ(\theta)/d\theta$
7:         until convergence; return $\theta^*$
8:         $\pi^* \leftarrow \pi(\theta^*)$
9:         Apply $\pi^*$ to system and record data
10:    until Task learned
```

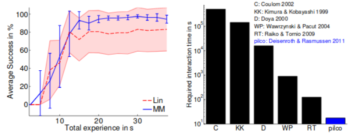
## Experimental results

- Two tasks explored in small dimensions:
  - Double pendulum swing-up (control space dimension 2, state space 4): after 50s, 95% success rate
  - Cart pole swing-up (control space 1, state space 4): after 15-20s, 95% success rate
- Tested both moment matching ($O(n^2E^2D)$) and linearization ($O(n^2ED)$) approximation techniques (MM was found to be more accurate)



## Comparison with other methods

- Compared PILCO learning speed (data efficiency with other RL methods) including some that utilized learned dynamics models.
- PILCO outperforms any then-existing RL algorithm by at least one order of magnitude on cart-pole.



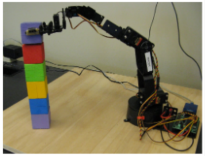
(a) Average learning curves with 95% standard errors: moment matching (MM) and posterior GP linearization (Lin).

(b) Required interaction time of different RL algorithms for learning the cart-pole swing-up from scratch, shown on a log scale.

Fig. 7. Results for the cart-pole swing-up task. (a) Learning curves for moment matching and linearization (simulation task), (b) required interaction time for solving the cart-pole swing-up task compared with other algorithms.

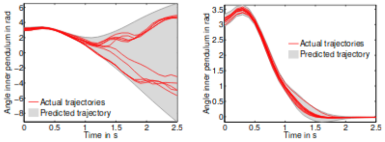
## Expanded experiments

- Expanded to a higher dimension unicycle upright task (control space 2, state space 12), successful after 20 trials (30s)
- Expanded to hardware tasks for cart-pole swing up and robotic manipulator arm (no pose feedback, just visual feedback).



https://www.youtube.com/user/PilcoLearner

## Deeper discussion



(a) Early stage of learning. (b) After successful learning.

Fig. 9. Long-term predictive (Gaussian) distributions during planning (shaded) and sample rollouts (red). (a) In the early stages of learning, the Gaussian approximation is a suboptimal choice. (b) PILCO learned a controller such that the Gaussian approximations of the predictive states are good. Note the different scales in (a) and (b).

- Cost function (section 6) - unorthodox cost function that encourages exploration.
- Quality of Gaussian approximation (section 7) - how good is our assumption of Gaussian approximation?

## Conclusion

- PILCO is a model-based learning framework that uses Gaussian processes to account for model uncertainty. It is a reinforcement learning approach that is able to learn in a data-efficient manner.
- Final line of the paper: "nonparametric Bayesian models can play a fundamental role in classical control setups, while avoiding the typically excessive reliance on explicit models"