

# Comparison of Model-Based and Model-Free Reinforcement Learning and Optimal Control

Julia Ströbel, Sebastian Hügler, and Jan Brüdigam

**Abstract**—The control of dynamical systems can be achieved by a variety of approaches with different advantages and drawbacks. In this paper, implementations of model-based reinforcement learning (RL), model-free RL, and optimal control for linear and non-linear dynamical systems are compared and discussed.

## I. INTRODUCTION

The increase of computational power in recent years has allowed for the successful implementation of learning algorithms to control a wide range of dynamical systems. Nonetheless, classical approaches to control problems also provide useful solutions for this class of systems. Therefore, the aim of this paper is to present and discuss advantages and drawbacks of the algorithms model-based reinforcement learning (RL), model-free RL, and optimal control. These methods are applied to a spring-mass system, a pendulum, and a cart-pole.

## II. SYSTEMS

This section shows the three models that will be controlled by the different algorithms:

### A. Spring-Mass-Damper

The spring-mass system can be described by a system of differential equations

$$\ddot{x} + \frac{D}{m}\dot{x} + \frac{c}{m}x = \frac{F}{m} \quad (1)$$

with the parameters  $k = 1 \text{ N/m}$  (spring) and  $m = 1 \text{ kg}$  (mass).

### B. Pendulum

The dynamics of the pendulum are described in (2).

$$\dot{x} = \begin{pmatrix} x_2 \\ \frac{u - b x_2 - m g s \sin(x_1)}{m s^2} \end{pmatrix}, \quad (2)$$

with the input  $u$ , the parameters  $b = 0.2 \text{ sNm/rad}$  (friction),  $m = 1 \text{ kg}$  (mass),  $g = 9.82 \text{ m/s}^2$  (gravity), and  $s = 1 \text{ m}$  (length of the pendulum).

### C. Cart Pole

The cart-pole dynamics are as follows:

$$\dot{x} = \begin{pmatrix} x_2 \\ \frac{2 m l x_4^2 s_3 + 3 m g s_3 c_3 + 4(u - c x_2)}{4(M + m) - 3 m c_3^2} \\ x_4 \\ \frac{-3 m l x_4^2 s_3 c_3 - 6(M + m) g s_3 - 6(u - c x_2) c_3}{l(4(M + m) - 3 m c_3^2)} \end{pmatrix}, \quad (3)$$

with the parameters  $m = 0.2 \text{ kg}$  (mass of pendulum),  $M = 1 \text{ kg}$  (mass of cart),  $l = 0.5 \text{ m}$  (length of the pole),  $g =$

$9.82 \text{ m/s}^2$  (gravity),  $c = 0.2 \text{ sN/m}$  (friction), and where  $s_3$  and  $c_3$  stand for  $\sin(x_3)$  and  $\cos(x_3)$ , respectively.

### D. Simulations

For the comparison of the following algorithms the simulation horizon for all models is  $2.5 \text{ s}$ , the maximum executable force for the spring-mass and cart pole is  $10 \text{ N}$ , and the maximum torque for the pendulum is  $10 \text{ Nm}$ . The initial state of spring-mass (pendulum) is  $0 \text{ m}$  ( $0^\circ$ ) and  $0 \frac{\text{m}}{\text{s}}$  ( $0 \frac{\text{rad}}{\text{s}}$ ). The cart pole starts at the initial angles  $10^\circ$ ,  $90^\circ$ , or  $180^\circ$ , and zero position of the cart.

The goals are  $\underline{x} = (1.5 \ 0)^T$  for the spring-mass and upright positions for the pendulum and cart pole.

## III. MODEL-BASED RL

In model based reinforcement learning a model of the system to be controlled is learned using measurement data. Secondly a policy is optimized using the learned model in order to control the real world system afterwards.

### A. Algorithm

In this work the PILCO approach for model based reinforcement learning presented in [1] is investigated.

In PILCO a probabilistic dynamic model for long term policy planning is trained. The unknown system dynamics are assumed with

$$x_t = f(x_{t-1}, u_{t-1}) \quad (4)$$

where  $x \in \mathbb{R}^D$  are the real valued system states,  $u \in \mathbb{R}^U$  are the control inputs and  $f$  is the unknown transition dynamics. The aim of PILCO is to find a policy  $\pi$  for the unknown system, i.e. the optimal parameters  $\theta$  of a controller, to minimize a cumulative cost  $J^\pi(\theta)$ .

a) *Learning Model Dynamics:* The learning of the model dynamics described in equation (4) is done using a gaussian process (GP), where  $(x_{t-1}, u_{t-1}) \in \mathbb{R}^{D+U}$  serve as inputs and  $\Delta_t = x_t - x_{t-1} + \epsilon \in \mathbb{R}^D$  with  $\epsilon \sim \mathcal{N}(0, \Sigma_\epsilon)$  serve as the GP's outputs. For the GP the squared exponential kernel and a prior mean  $m \equiv 0$  is chosen. The parameters (length scales, signal/ noise variance) are trained using evidence maximization. Finally the one step predictions based on the GP are:

$$p(x_t | x_{t-1}, u_{t-1}) = \mathcal{N}(x_t | \mu_t, \Sigma_t), \quad (5)$$

$$\mu_t = x_{t-1} + \mathbb{E}_f[\Delta_t], \Sigma_t = \text{var}_f[\Delta_t] \quad (6)$$

Here the mean  $\mathbb{E}_f[\Delta']$  and the variance  $\text{var}_f[\Delta']$  of a test input  $\Delta'$  can be calculated using the GP's hyperparameters[1].

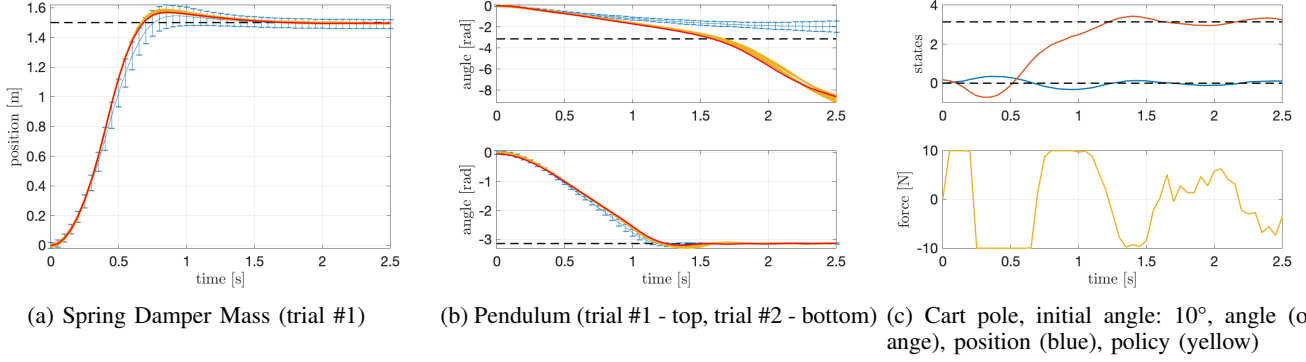


Fig. 1: Figures (a)-(b) show trajectories used for training of GP (red), additional rollouts not used for GP training (yellow), predicted system trajectory with confidence interval of 95% (blue) and target states (black).

*b) Policy Evaluation:* In order to predict a successor state  $x_t$  based on the policy input  $u_{t-1} = \pi(x_{t-1}, \theta)$  a joint distribution  $p(\tilde{x}_{t-1}) = p(x_{t-1}, u_{t-1})$  has to be determined. This is done by first integrating out the state  $x_{t-1}$  where  $\mu_u$  and  $\Sigma_u$  are obtained. Secondly by additionally calculating the cross-covariance  $\text{cov}_f = [x_{t-1}, u_{t-1}]$  the joint state-policy distribution can be approximated by a Gaussian. This result is then used for predicting  $p(\Delta_t)$  with

$$p(\Delta_t) = \int p(f(\tilde{x}_{t-1}) | \tilde{x}_{t-1}) p(\tilde{x}_{t-1}) d\tilde{x}_{t-1} \quad (7)$$

Often there is no analytically feasible solution for equation (7). Therefore the Gaussian of  $p(\Delta_t)$  is approximated using moment matching or linearization [2]. With the Gaussian of  $p(\Delta_t)$  an approximate Gaussian is obtained for  $p(x_t)$  by

$$\mu_t = \mu_{t-1} + \mu_{\Delta}, \quad (8)$$

$$\Sigma_t = \Sigma_{t-1} + \text{cov}[x_{t-1}, \Delta_t] + \text{cov}[\Delta_t, x_{t-1}], \quad (9)$$

$$\text{cov}[x_{t-1}, \Delta_t] = \text{cov}[x_{t-1}, u_{t-1}] \Sigma_u^{-1} \text{cov}[u_{t-1}, \Delta_t]. \quad (10)$$

Finally the the expected cost is calculated according to

$$\mathbb{E}[c(x_t)] = \int c(x_t) \mathcal{N}(x_t | \mu_t, \Sigma_t) dx_t. \quad (11)$$

The cost  $c(x)$  considering the target state  $x_{target}$  is a inverted squared exponential function for each state, where the width is determined by  $\sigma_c$ . Using this cost function the resulting partial derivatives over  $\mu_t$  and  $\Sigma_t$  for the gradients in the policy optimization are calculable in a closed form [2].

## B. Results

In this part the PILCO algorithm is applied to the three previous introduced systems. The simulations have been done using the online available PILCO framework for MATLAB [3]. For the policy a radial basis function (RBF) controller is used.

Figure 1a shows that the spring damper mass system is controlled on trail #1 after optimizing the GP with the data of random exploration in the beginning. This is due to the fact the system is a linear one and the complexity is quite low. The pendulum in figure 1b is controlled after the trial #2. At trail #1 the state space of the system is only explored

in the angular region from  $-\frac{\pi}{2}$  to  $\frac{\pi}{2}$ . The result is that the GP is only trained with data out of this region and the uncertainty of the GP model grows when the planned trajectory leaves this region (s. blue errorbars in fig. 1b). After trail #2 however, the system is stabilized at  $-\pi$  for all following trials. This shows that PILCO can handle unknown nonlinear systems quite effectively. This becomes even more obvious when considering results of the cart-pole trajectories. It takes 7 trials for the swing up from  $10^\circ$ , 3 trails from  $90^\circ$  and 2 trails from  $180^\circ$ . After the swing up the steady state is controlled stable with slight inaccuracies. Figure 1c shows the swing up from  $10^\circ$ . Remarkable for this swing up is that the controller uses gravity to let the pole first swing in the other direction and swing it to its target position afterwards. This is due to the restriction of the controller to 10 N maximum force. An linear controller would not be able to plan such a trajectory.

## IV. MODEL-FREE POLICY SEARCH

This section shows an approach of a model-free policy search using Policy learning by Weighting Exploration with Returns (PoWER).

### A. Algorithm

The PoWER-algorithm is based on applying varying policies to the unknown system and optimizing these policies according to their rewards. The policy  $\pi_\theta$  applied on the system in this work is based on Dynamic Movement Primitives (DMP)

$$\pi_\theta = \ddot{y} = \tau^2 \alpha_y \beta_y (g - y(t) - \tau^2 \alpha_y \dot{y}(t) + \tau^2 f(z(t))) \quad (12)$$

with parameters  $\tau, \alpha_y, \beta_y > 0$  [4]. The function  $f$  is represented by a radial basis function (RBF) network

$$f(z(t)) = \frac{\sum_{i=1}^N \phi_i \omega_i}{\sum_{i=1}^N \phi_i} z \quad (13)$$

where  $\omega_i$  corresponds to the weights of the RBFs  $\phi_i$ . Usually five to twenty radial basis functions  $\phi_i$  per system's degree of freedom (DoF) are used. The variable  $z(t)$  is a system state resulting from the dynamics

$$\dot{z} = -\tau \alpha_z z \quad (14)$$

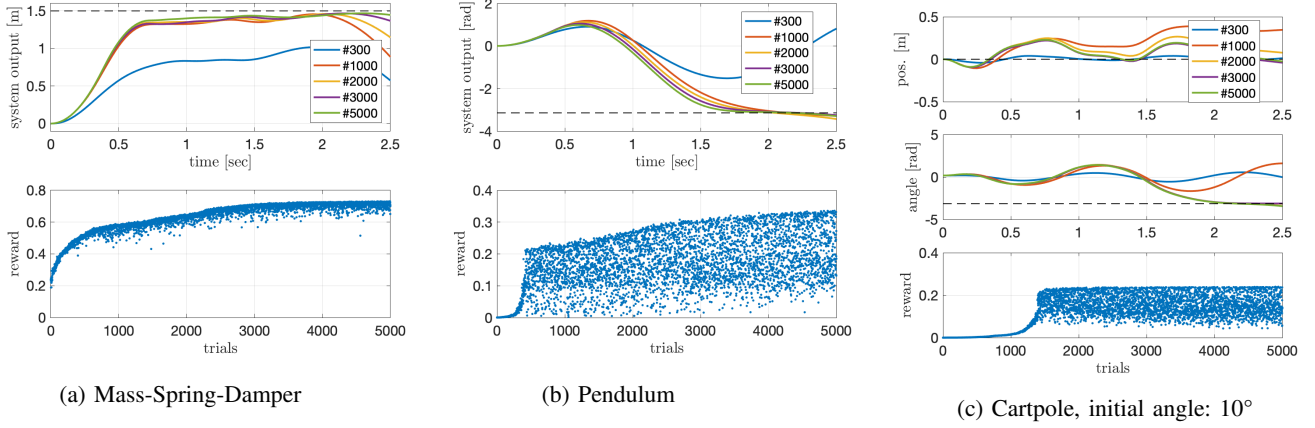


Fig. 2: Figures (a)-(c) show the system output over time as well as the reward at the increasing iteration steps.

with  $\alpha_z > 0$  [4]. To optimize the policy  $\pi_\theta$ , the policy parameter vector  $\theta$  (containing weights  $\omega_i$  of RBF  $\phi_i$ ) are explored in parameter space with adding a gaussian noise  $\varepsilon \sim N(0, \sigma_\varepsilon)$  to the parameters in iteration  $n$  [5]

$$\theta_{temp} = \theta_n + \varepsilon \quad (15)$$

The policy obtained by temporary parameters  $\theta_{temp}$  are rolled out on the system and the system trajectory  $x_t$  is measured. A cumulative reward  $R$  assesses the quality of the rollout using the trajectory  $x_t$ :

$$R(\theta_{temp}) = \sum_{t=0}^{T-1} r_t(x_t) \quad (16)$$

The reward  $r_t$  at time  $t$  is calculated using a squared exponential function. The procedure of applying different policies using different parameters  $\theta_{temp}$ , which are obtained by the gaussian noise added to the base parameter  $\theta_n$ , is repeated  $k$  times at each iteration  $n$ . The parameter update for the next iteration  $n+1$  is then done by ordering the temporary parameters  $\theta_{temp}$  by their rewards  $R(\theta_{temp})$  from the best to the worst. The update law is then

$$\theta_{n+1} = \theta_n + \frac{\Delta\theta_1 R(\theta_1) + \dots + \Delta\theta_{10} R(\theta_{10})}{R(\theta_1) + \dots + R(\theta_{10})} \quad (17)$$

with  $\Delta\theta_k = \theta_{temp,k} - \theta_n$ ,  $k = 1, 2, \dots, 10$ . The mentioned steps are repeated until the reward  $R(\theta_n)$  is greater than a threshold close to 1 or a maximum number of iterations is reached.

## B. Results

In the following the results of the three systems mass-spring-damper, pendulum and cartpole are shown. For each system it takes around 3.000 trials to reach the target positions with a sufficient quality (see figure 2). At first the three reward curves of the systems show a fast improvement. After about 500 trials the rewards of the mass-spring-damper and the pendulum show a moderated linear improvement; the reward of the cartpole has its significant point at about 1.500 trials. The appearance of the three graphs differ:

While the reward of the mass-spring-damper system shows comparatively little scattering and a maximum of about 0.75 (out of 1), the rewards of the two other systems have larger variations and lower maxima. This is due to the fact that leaving a point with higher rewards, entails a higher penalty in these two systems. Because the policy decreases with  $z$ , it is difficult for the algorithm to hold the target position in the end of the rollout horizon.

## V. OPTIMAL CONTROL

The overarching concept in optimal control is to optimize control inputs and system states according to a specific cost function.

Since there are many different forms of optimal control, in this section, LQR was applied to the linear spring-mass system, iLQR to the pendulum, and MPC to the cart-pole system. This highlights the different capabilities of these approaches.

### A. Algorithms

The continuous LQR aims at minimizing the cost function depicted in (18).

$$J_1 = \int_0^\infty (x^T Q x + u^T R u) dt, \quad (18)$$

with weight matrices  $Q$  and  $R$ . Since there was no specifications for input and states, the weights were simply set to  $Q = \text{diag}(1, 1)$  and  $R = 0.1$ . This results in the well known control law  $u = -Kx$  which transforms (1) into

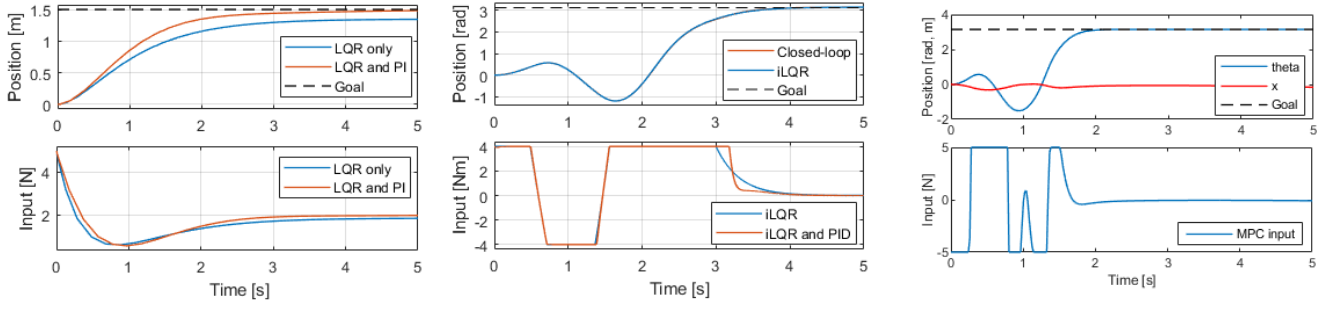
$$\dot{x} = (A - BK)x. \quad (19)$$

$K$  is the optimal state feedback gain.

With a goal position different from the origin, a prefilter matrix  $L$  as shown in (20) is necessary.

$$L = (C(BK - A)^{-1}B)^{-1} \quad (20)$$

Since LQR is lacking an integral component, a simple PI controller ( $K_P = K_I = 1$ ) was added to drive the system into the desired goal position even under



(a) Comparison of LQR and LQR+PI. Position  $x_1$  of the spring-mass system (top) and control input  $u$  (bottom). (b) Comparison of iLQR result and closed-loop simulation. Angular position  $x_1$  of the pendulum (top) and control input  $u$  (bottom). (c) MPC of the cart-pole. Angle  $\theta = x_3$  of the pendulum and position  $x = x_1$  of the car (top) and control input  $u$  (bottom).

Fig. 3: Plots (a) - (b) show the state trajectories for the three systems. Mass-spring (a), Pendulum (b), Cart pole (c).

disturbances. Note that the system would also be stable for any other controller gain but experience higher overshooting.

The iLQR approach for the pendulum was based on [6] and their Matlab toolbox.

The input was limited to  $|u| < 4$  Nm, so that the goal can only be reached by swinging back and forth.

The cost function for the pendulum problem is defined as

$$J_2 = \frac{1}{2} \bar{x}_N^T P \bar{x}_N + \frac{1}{2} \sum_{k=0}^{N-1} (\bar{x}_k^T Q \bar{x}_k + u_k^T R u_k), \quad (21)$$

with the time horizon  $N$ , the deviation of the current state from the desired state  $\bar{x}_k = (x_k - x_{des})$ , the final weight matrix  $P$ , and the familiar  $Q$  and  $R$  matrices.

Without any other specifications, the input weight was set to a rather low value of  $R = 10^{-3}$  and  $Q = \text{diag}(10^{-3}, 0)$  to allow for necessary swinging and full speed during the run. The final cost was set to  $P = \text{diag}(100, 0.1)$  to ensure the upright position.

The iLQR algorithm was set to a time horizon of  $N = 5000$  at a  $t = 1$  ms sample time, which equals a runtime of 5 seconds.

iLQR is an open-loop controller. Therefore, in the implementation, the resulting input trajectory  $\underline{u}$  from the optimization was used as feed forward control, while a PID controller minimized the difference between the estimated trajectory  $\underline{x}$  (from the optimization) and the actual state. Without this setup, even the slightest perturbation of the system (e.g. rounding errors) leads to a failed upswing since the goal position is an unstable equilibrium point.

The MPC implementation was based on [7]. Here, the Matlab function “fmincon” is used to solve the optimization.

The input was constrained to  $|u| < 5$  N, and the cart position was limited to  $|x_1| < 4$  m. The cost function is the same as in (21), with  $P = \text{diag}(0, 10, 10, 10)$  (end position of the cart is irrelevant as long as it stays within the bounds),  $Q = \text{diag}(20, 10, 10, 0)$  (cart should remain in the center during upswing, while angular speed is irrelevant), and  $R = 0.01$ .

The prediction horizon was set to  $N = 20$  at a sample time of  $t = 20$  ms, and the simulation ran for 4 seconds.

Since MPC is a closed-loop control strategy, no additional controller was necessary.

## B. Results

For the spring-mass system without disturbances, the goal was reached by the LQR controller. However, with an added constant disturbance of  $d = 0.5$  N, the PI controller was necessary as shown in figure (3a). Note that using the PI controller instead of LQR would cause the system to become unstable.

For the pendulum, the iLQR algorithm converged after 52 iterations. The resulting trajectory is one initial swing to the left, followed by a complete upswing to the right. Figure (3b) displays the stand-alone iLQR trajectory and one with an added PID controller.

Both trajectories are almost identical, and there is only a small deviation in the control input at the end of the upswing (compare figure (3b) bottom at 3 seconds). However, this slight change is crucial to keep the pendulum in the upright position. Without the added PID controller, the system remains unstable as shown in figure (??), since iLQR is not a closed-loop controller.

The MPC controller managed to stabilize the pendulum of the cart-pole in the upright position for all initial angles. The successful swing-up for an initial position of 10 is presented in figure (3c).

## VI. COMPARISON AND DISCUSSION

As the results show, optimal control is an effective method as it does not need any interactions with the real system and still delivers high accuracy with reasonable computational effort. However, the system model has to be known or derived by linearization, and the parameterization of the weights requires some work.

For unknown or highly non-linear models this method might fail, which is when the presented reinforcement learning approaches appear to be valid options. The advantage of

PILCO is that model learning and simulative parameter optimizations require only few system interactions for stable closed loop control. The optimization algorithm is, however, computationally expensive.

By contrast, the calculation of DMP-policies is faster, but requires several system interactions to yield an (open loop) policy. Both RL algorithms show slight deviations around the goal position due to their inherent structure (see figures. 1 (c); 2 (b), (c)).

## VII. CONCLUSION

Summarizing the results, optimal control presents a good starting point for controlling a system when the dynamics are (partially) known. More complex sub-dynamics (e.g. friction) can then be tackled with reinforcement learning. Here, DMP policies appear to be a good choice for repetitive tasks, while PILCO is a strong algorithm for a general control scheme.

## REFERENCES

- [1] Deisenroth, Marc, and Carl E. Rasmussen. "PILCO: A model-based and data-efficient approach to policy search." *Proceedings of the 28th International Conference on machine learning (ICML-11)*. 2011.
- [2] Deisenroth, Marc Peter, Dieter Fox, and Carl Edward Rasmussen. "Gaussian processes for data-efficient learning in robotics and control." *IEEE transactions on pattern analysis and machine intelligence* 37.2 (2015): 408-423.
- [3] Deisenroth, Marc Peter, Dieter Fox, and Carl Edward Rasmussen. "PILCO website", 2013. [Online]. Available: <http://mlg.eng.cam.ac.uk/pilco/>. [Accessed: 27- Jan- 2019]
- [4] Studywolf. "Dynamic Movement Primitives Part 1: The Basics", 2013. [Online]. Available: <https://studywolf.wordpress.com/2013/11/16/dynamic-movement-primitives-part-1-the-basics/>. [Accessed: 28-Jan- 2019]
- [5] Jens Kober, Jan Peters. "Policy search for motor primitives in robotics". *Advances in neural information processing systems* (2009): 849-856.
- [6] Tassa, Mansard, and Todorov. "Control-limited differential dynamic programming". *IEEE International Conference on Robotics and Automation (ICRA)*. 2014.
- [7] Gruene, and Pannek. "Nonlinear Model Predictive Control". Springer Verlag. 2017.