Noname manuscript No.
(will be inserted by the editor)

Application of Reinforcement Learning Methods

Asynchronous Advantage Actor-Critic (A3C)
Probabilistic Inference for Learning COntrol (PILCO)

Fabian Otto · Johannes Czech

Received: date / Accepted: date

Abstract In this work we apply two reinforcement learning algorithms Asynchronous Advantage Actor-Critic and Probabilistic Inference for Learning COntrol on three different Quanser robot environments: *CartpoleStabShort-v0*, *Cartpole-SwingShort-v0* and *Qube-v0*. The reinforcement learning agents are trained and evaluated in simulation as well as on the real systems. We show that A3C can solve all tasks in simulation, but has poor sample efficiency and transferability to the real systems. PILCO is highly sample efficient but is only able to solve the stabilization task and fails to learn the other tasks due to strong smoothing in its dynamics model and too high frequency of the environments. The source code including all pre-trained models are available at: `https://github.com/BoboDance/RL-Project`.

Keywords Reinforcement Learning · Quanser Robots · A3C · A2C · PILCO

1 Introduction

The application of reinforcement learning algorithms on robotic systems has gained in interest in recent years [9]. Besides solving a learning task, the application on robotic systems raises additional challenges and requirements [16]. Some of these are sample efficiency, joint, sensor and motor preservation as well as transferability of a policy which was learned in simulation. This paper gives an extensive summary about using Asynchronous Advantage Actor-Critic (A3C) and Probabilistic Inference for Learning COntrol (PILCO) to solve three unique Quanser robot tasks in simulation and on real systems. We shortly describe the algorithms and environments followed by an evaluation including different hyper-parameter settings and a detailed analysis of the major challenges and problems.

Fabian Otto
TU Darmstadt
E-mail: fabian.otto@stud.tu-darmstadt.de

Johannes Czech
TU Darmstadt
E-mail: johannes.czech@stud.tu-darmstadt.de

2 Algorithm Description

A3C [10] considers the infinite horizon *Markov Decision Process* (MDP) with continuous or discrete state and action spaces. It is a model free, actor critic method, which represents the actor and the critic with neural networks. A3C utilizes multiple workers in parallel, each with their individual clone of the environment and optimizes globally shared actor and critic networks. This asynchronous approach allows for better hardware utilization and exploration. Besides A3C, a synchronous and batched version, called *Advantage Actor-Critic* (A2C) maintains only one worker while running multiple environments in parallel. This yields empirically to more stability, faster convergence and improved GPU utilization [15, 11].
PILCO [2, 4, 3] considers the finite horizon MDP with continuous state and action spaces. The main goal is to enable data-efficient policy learning for real systems. This is achieved by a model based policy search method, which utilizes analytic policy gradients in order to find the optimal policy parameters. For the probabilistic dynamics model, a Gaussian process representation is applied. It allows to describe uncertainties, limiting the effect of model errors during planning. For the policy improvement, the cost of a trajectory rollout is evaluated analytically based on the dynamics model. Afterwards, the parameters can be updated with numerical methods, e. g. L-BFGS or Conjugate Gradients. However, the rollout requires the marginal probability of each state along the trajectory, which cannot be calculated exactly. Instead, PILCO approximates this distribution with a Gaussian based on approximate inference via robust moment matching [1, 5].

3 Platform Description

All three platforms require to solve a stabilization task, while *CartpoleSwingShort-v0* and *Qube-v0* combine this with a swing-up. The simulations and real systems have continuous state and action spaces. The goal of the *CartpoleStabShort-v0* environment is to balance a pole on a cart which can move horizontally on a fixed linear track. The initial position of the pole is upright with slight variations of the angle. The state is specified by $[x, \sin(\theta), \cos(\theta), \dot{x}, \dot{\theta}]$, where x represents the horizontal position, and θ the angle of the pole. $\dot{x}$ and $\dot{\theta}$ represent the position and angle derivatives respectively. The legal action range is $\pm 24V$. However, stabilizing the pole is possible with a reduced action range of $\pm 5V$ which is consequently used for the experiments. One episode is terminated after successfully balancing the pole over 10000 steps, i.e. not violating the angle or position thresholds.
CartpoleSwingShort-v0 uses the same environment specifications as *CartpoleStabShort-v0* and provides a swing-up task with subsequent stabilization. Therefore, the angle has no constraints and the pole's position is initialized pointing downwards. As for *CartpoleStabShort-v0*, the maximum action range of $\pm 24V$ is not necessary in order to solve this task, it is sufficient to use $\pm 5V$.
Qube-v0, which is also known as Furuta Pendulum, can be solved by performing a pendulum swing-up in the vertical plane by controlling the attached driven arm rotating in the horizontal plane. The state observations are represented as $[\cos(\theta), \sin(\theta), \cos(\alpha), \sin(\alpha), \dot{\theta}, \dot{\alpha}]$, where θ describes the angle of the driven arm and α the pendulum angle. $\dot{\theta}$ and $\dot{\alpha}$ represent the respective derivatives. The legal action range is $\pm 5V$.

4 Experiments and Results for A3C

We use two fixed network architectures throughout all experiments. The actor has one hidden layer with 200 nodes and the critic two hidden layers with 100 nodes, both use ReLU activation functions. The actor returns μ, Σ of a Gaussian and the critic a single output for the state value. We found smaller architectures are often not sufficient. The original implementation [10] uses a shared network architecture. However, when comparing distinct and shared networks, the performance and convergence is better for the former. Further, including an LSTM layer for the Quanser environments decreases performance and was as a result removed from the final version. The difference in performance can be explained based on the observation space, in contrast to the Atari games, the Quanser environments return a lower dimensional and fully observed observation. Moreover, A2C is always outperforming the asynchronous version A3C on all three environments under comparable conditions. Additional changes to the original A3C version include the use of *General Advantage Estimation* (GAE) [12] in order to trade-off bias and variance for the n-step returns.
We define a baseline hyper-parameter setting for A2C in order to conduct systematic evaluations. The learning rate is set to 10^{-4} and 10^{-3} for actor and critic respectively. The discount $\gamma = 0.99$, the GAE parameter $\tau = 0.99$ and the entropy loss weight $\beta = 10^{-4}$ with 50 rollout steps in 5 parallel environments using the ADAM [8] optimizer. Furthermore, all evaluation entries are averaged over ten test runs. Figure 1a shows the mean total test reward using 5 different seeds as well as the minimal and maximal performance. For the stabilization, A2C is improving over time but is not able to maintain a good policy. Even for this simple task, A2C quickly forgets about its good policy and drops in performance after some additional steps. Moreover, the sample efficiency is poor, A2C requires in the best case scenario 600000 steps, on average approximately 1.5 million. Our best policy achieves on 100 consecutive test runs a total reward of 19999.95 ± 0.04 with an episode length of 10000.0 ± 0.0. When applying this policy to the real system, the cartpole could not be stabilized, due to strong overfitting to the simulation. However, policies, which perform slightly worse in simulation, can stabilize the cartpole without additional training. On the real system, we achieve a total reward of 4102.34 ± 460.38 with an episode length of 1995.07 ± 335.05 evaluated over 15 test runs. The best run attains a total reward of 5436.31 with an episode length of 2720.0. When analyzing different hyper-parameter settings, only slight improvements in sample efficiency can be achieved. Figure 1b compares the performance for different learning rates. In general, sample efficiency is increased by using lower learning rates for the critic, whereas reducing the actor learning rate mostly results in a decrease. This is consistent for different network architecture settings. The combined reduction performed best, however the changes for the actor do not result in consistent improvements. Figure 1c visualizes the performance of A2C for different discount factors. Reducing γ leads to longer training times, hence worse sample efficiency. The low complexity of the stabilization task allows to use a higher $\gamma = 0.999$ for the best performance, as the agent does not benefit from higher immediate rewards. When altering the trade-off between bias and variance, the baseline value $\tau = 0.99$ performs poorly, even n-step Monte Carlo ($\tau = 1$) converges faster. Picking a trade-off around $\tau = 0.95$, allowing for stronger bias reduction, achieves the best performance in our experiments. This

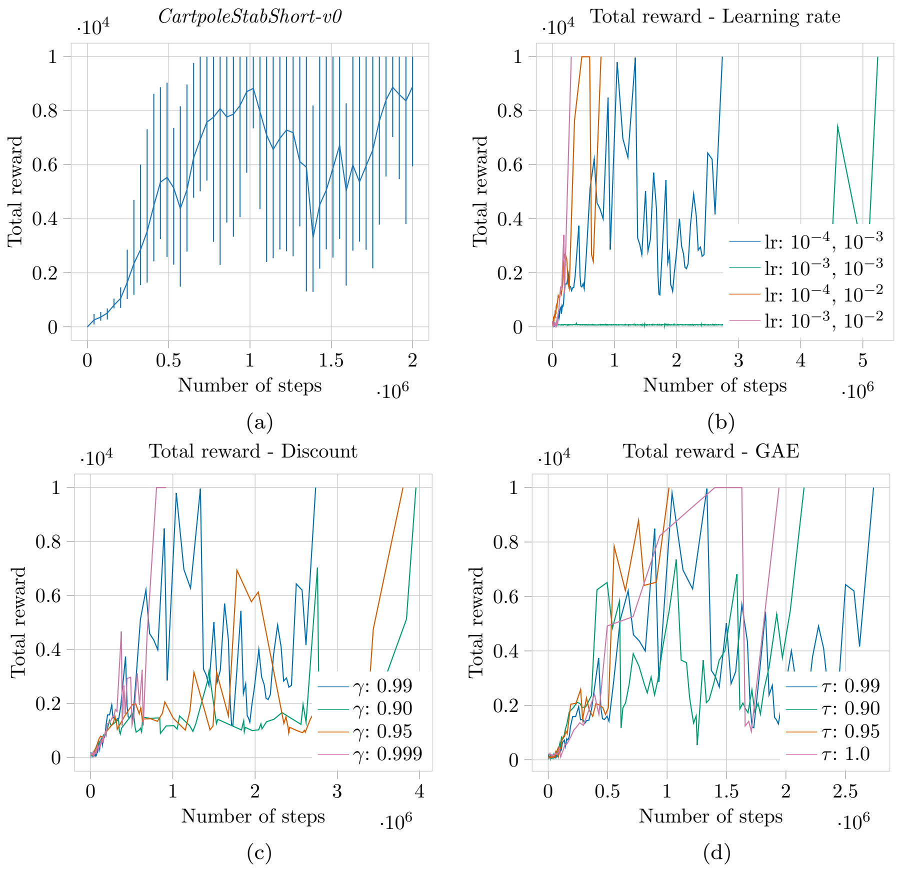
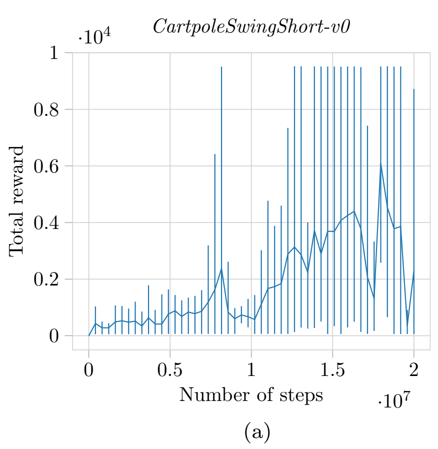


Fig. 1: A2C: Total reward on *CartpoleStabShort-v0* over five seeds (1a). Convergence to 9999 total reward for different learning rates (1b) for actor (left) and critic (right), discount values (1c) and GAE's bias variance trade-off (1d).

is also congruent with the performance in the literature, where typically values of $0.90 < \tau < 0.99$ show improved results [12]. Reasonable changes to the rollout length do not result in any performance differences. However, more environments, hence larger batches, decrease sample efficiency. Further, when using RMSProp [13] for optimization, the agent is not able to solve the stabilization task, although it is used in the original paper [10].
For *CartpoleSwingShort-v0* the same baseline hyper-parameter setting are used with maximum action of $\pm 10V$. Our evaluation is conducted with a reduced episode length of 5000 steps because the relative policy performance does not show significant differences when using longer episodes. Due to the poor sample efficiency, solving the swing-up took approximately five hours of training, depending on the available hardware even longer. Figure 2a shows the total reward over five different seeds with minimal and maximal reward. Compared to the stabilization, the swing-up baseline requires ten times more steps, i.e. 15 million. Moreover, the learning progress is significantly more unstable. We find that A2C heavily relies on finding good random actions, which lead to high rewards. The increased complexity of the environment is exacerbating this problem even further. For the



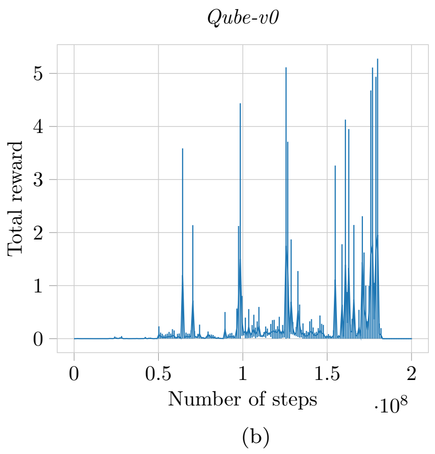


Fig. 2: A2C: Total reward of *CartpoleSwingShort-v0* (2a) and *Qube-v0* (2b)

swing-up, A3C is not able to solve the environment using the baseline, similarly for the RMSProp optimizer. A2C fails when reducing the action range to $\pm 5V$, even after half a day training and different hyper-parameter choices including discount, GAE parameter and rollout length. In general, A2C shows relatively quick improvements in the beginning, but is prone to local optima. A2C is trying to get in sync with the frequency of the environment, so that the pole is mainly upright when a reward is received. Continuing this overfitting trend, the working policies for the swing-up utilize the full length of the cartpole track and manage the swing-up within one movement. We achieve a total reward of 19521.84 ± 3.82 with an episode length of 10000.0 ± 0.0 for 100 consecutive test runs. Due to the strong overfitting, we reduce the action range to $\pm 5V$ when applying the policy to the real system. This results in a reward of 1202.94 ± 711.18 with an episode length of 1438.15 ± 705.69.

Hyper-parameter evaluations are similar to *CartpoleStabShort-v0*. The initial $\tau = 0.99$ performs poorly. However, the improvement of $\tau = 1$ is rather small and the sample efficiency can be improved by up to 50% for $0.90 < \tau < 0.99$. Although the task is more complex, we do not experience any improvements with different discount factors, the correct choice of τ was more important. Longer rollouts or more environments do not result in improvements either. We also found penalizing or rewarding based on the position $x = 0$ as well as squaring the reward does not support faster convergence. When using 100Hz control frequency we are able to solve the swing-up and accumulate a higher reward, but solely based on the reduced frequency.

For *Qube-v0* the aforementioned issues get aggravated. A2C cannot reliably learn a good policy after a constant amount of time and training takes three to four days. Due to this high computational cost we show the performance over only three seeds in Figure 2b. The inconsistency is more severe and solving the environment is only possible at small peeks after 70 to 140 million steps. Albeit, we experience that the loss is converging to 0 early in the beginning. This could be caused by the shape of the reward function. A2C was approaching this issue by increasing exploration, which resulted in a significantly higher entropy loss compared to both cartpole tasks. The final policy is again overfitted and finds a physically impossible solution for the real system. Using a control frequency of 500Hz we achieve a reward of 5.51 ± 0.01 with an episode length of 3000.0 ± 0.0 over 100 consecutive trials. On a shorter training run with 50Hz we achieve 3.87 ± 0.60 with an episode length of 293.88 ± 35.66. Applying the best policy to the real system, the reward decreases to 0.08 ± 0.02 with an episode length of 396.65 ± 27.09 over 20 runs. After continuing training for 1750 steps on the real system the mean reward increases to 0.15 ± 0.21 with an episode length of 467.0 ± 178.77. Directly approaching the reward problem, we removed the exponential in the reward computation and multiply the reward with a constant factor, however this does not yield in any improvements. Changing τ and rollout lengths as well as reducing learning rates does not help either. Lower learning rates were more likely to result in the inability to learn the task.

5 Experiments and Results for PILCO

In order to apply PILCO to the Quanser environments, the frequencies of the systems were reduced to 100Hz for *Qube-v0* and *CartpoleSwingShort-v0* as well as 50Hz for *CartpoleStabShort-v0*. Using PILCO in high frequency environments requires significantly longer rollouts, making the computation inefficient with our available hardware. Further, we mainly use sparse Gaussian processes approximations throughout our experiments as full Gaussian processes were mostly too computationally expensive. Moreover, PILCO uses maximal 300 samples per interaction with the environment. To avoid numerical instabilities, length-scales and noise variance are constrained when optimizing the dynamics model. Due to the finite horizon case, we consider discount $\gamma = 1$.
Equivalent to A2C, we define an initial baseline hyper-parameter setting for our evaluations of *CartpoleStabShort-v0*. We utilize a horizon of length 40, 300 inducing points, an *Radial Basis Function* (RBF) policy with 25 features and initialize the weight for the exponential loss with the identity matrix. For the stabilization, PILCO finds a policy with only two environment interactions (see Figure 3a). Including the 300 initial random samples, PILCO requires $\leq$ 900 samples in order to find a stable policy. In simulation over 100 test runs, PILCO achieves 19999.88 ± 0.10 with an episode length of 10000.0 ± 0.0 When evaluating the policy on the real system, we attain a total reward of 1145.95 ± 397.30 with an episode length of 574.0 ± 98.63 evaluated over 15 test runs. Our best run accumulates a total reward of 2192.11 with an episode length of 1097.0. Additionally, PILCO was trained exclusively on the real system over 11 episodes. Compared to the policy transferred from simulation, the reward increases to 1774.94 ± 539.13 with an episode length of 846.88 ± 287.77. The best run has a total reward of 2877.43 with an episode length 1440. When reducing the horizon length to 20 or the number of inducing points to 200, the performance is worse and PILCO does not stabilize the cartpole over 10000 time steps. Further, reducing the RBF policy complexity to only 10 features shows similar results. On the contrary, removing the position x and the derivatives $\dot{x}$, $\dot{\theta}$ from the weight computation results in the same performance as the baseline.
For *CartpoleSwingShort-v0* we change the baseline horizon length and RBF features to 50. We find that PILCO is not able to solve the swing-up, independent of the hyper-parameter setting. One of the reasons is the required length of the

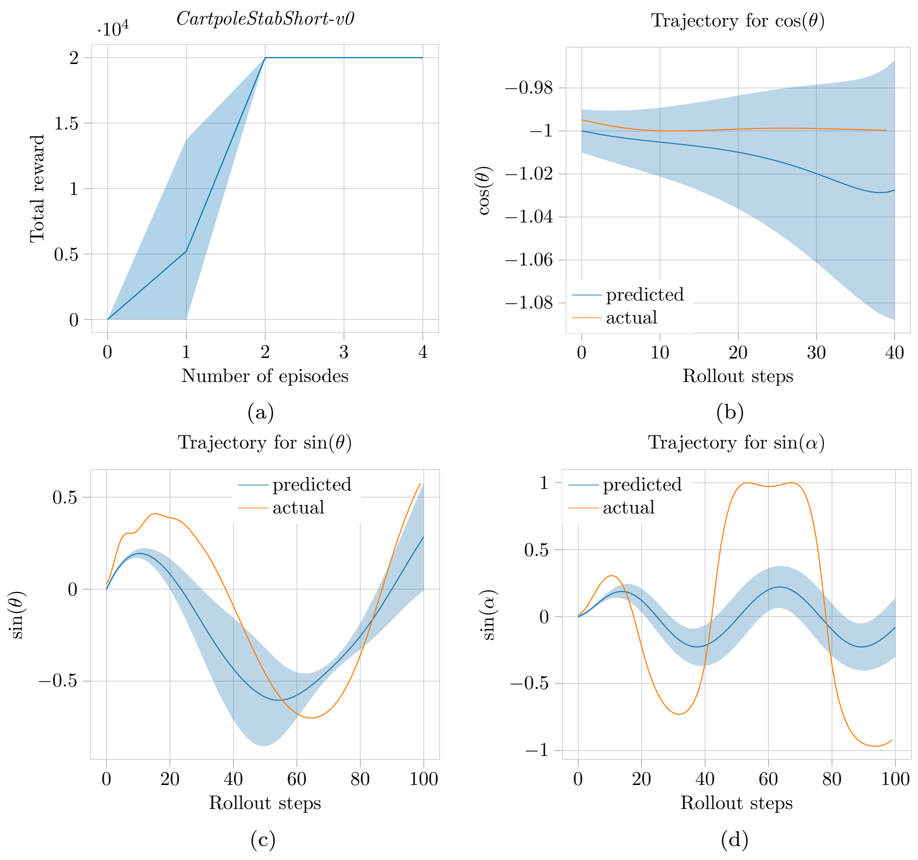


Fig. 3: PILCO: Total reward of *CartpoleStabShort-v0* (3a). Trajectory plots for $\cos(\theta)$ of *CartpoleStabShort-v0*, $\sin(\theta)$ of *CartpoleSwingShort-v0* and $\sin(\alpha)$ of *Qube-v0*

trajectory. Even with the reduced environment frequency and an horizon of 100, the task cannot be completed within this time frame. Consequently, the policy improvement step is not considering important information during optimization. Figure 3c shows that the dynamics model is underestimating the true dynamics and the Gaussian processes overly smooths the trajectory. For the simpler dynamics in *CartpoleStabShort-v0* (Figure 3b) this is not an issue. Furthermore, we experience premature termination of the environment even when including the position x in the loss. The dynamics sometimes causes gradients of almost zero, which leads to no policy improvement. We were not able find the root cause of this problem, but it appears that this more common for dynamics models with low certainty. Adjusting the hyper-parameter constraints for the dynamics model does not address this problem sufficiently. We also find no improvements when adjusting the weights for the loss, the number of features or the maximum actions. Evaluating the best policy over 100 test runs results in a total reward of 10548.76 ± 52.36 with an episode length of 10000.0 ± 0.0.
For *Qube-v0* as well as for *CartpoleSwingShort-v0* including the derivatives $\dot{x}$, $\dot{\alpha}$,

$\dot{\theta}$ in the loss computation is decreasing performance. The agent learns to perform no actions in order to optimize those values. Changes to the number of RBF features, inducing points, horizon length and loss weight does not yield any improvements for *Qube-v0*. Increasing the weight of the angle for the driven arm in the loss results in the agent performing less actions. Whereas removing or lowering the weight tends to encourage PILCO to perform a swing-up as fast as possible, leading to environment boundaries violations. Similar to *CartpoleSwingShort-v0* the dynamics are overly smoothed (Figure 3d). In general, PILCO is often only able to approximate either the sine or cosine of angles properly, but fails to give a good estimate for the corresponding other. Our best *Qube-v0* policy scored a total reward of 0.35 ± 0.17 with an episode length of 184.32 ± 49.35 over 100 test runs. The best run had a reward of 0.96 with an episode length of 300.0.

6 Comparison and Problems

A3C and PILCO are both policy search methods, however their approach is significantly different. The most important difference is their sample complexity. A3C was developed and tested on simulated environments using multiple environments in parallel, whereas PILCO was designed for data-efficiency on real systems. Additionally, A3C optimizes its policy based on trajectories from the real system, which increases the sample complexity even further. Compared to this model-free approach, PILCO learns a Gaussian process representation for the dynamics of the system. However, there are approaches which aim to replace the dynamics model with e. g. Bayesian neural networks [6]. Regarding the representation of the policy, A3C utilizes neural networks for both the actor and the critic model, PILCO is mainly designed around RBF controllers to maintain data-efficiency and express uncertainty. When evaluating the performance on the Quanser environments, we found that these differences also hold true in practice. For *CartpoleStabShort-v0*, PILCO was able to learn a stable policy after $\leq$ 900 steps in the simulation, whereas A3C required more than 600000 for the best hyper-parameter setting. The poor sample efficiency of A3C is also mentioned by others [7,14,11,15]. Despite the drastic disparity of sample complexity, A3C and PILCO require approximately the same training time. Further, the memory requirements for PILCO are high, mainly caused by Gaussian process regression as well as the RBF policy. Due to these issues, using full Gaussian processes is mostly infeasible for the given Quanser environments. These issues are partly resolved by using an sparse Gaussian processes approximation. Another reason is that PILCO's original environments are quite different from the Quanser environments. We showed that stabilization problems are reproducible, but the original swing-up problems are solvable quicker and therefore do not require long trajectory rollouts. Besides this, Quanser environments with 50HZ already return significantly more samples than the original PILCO environments. Finally, in our opinion, PILCO is complicated to implement and understand, but its performance does not justify this in our experiments. A3C has a lower complexity, but lacks sample efficient computation and good transferability, which makes it unsuitable for real systems. Nevertheless, its overfitting capabilities allow it to work incredibly well in simulation and achieve one of the best scores for the Quanser environments.

References

1. Candela, J.Q., Girard, A., Larsen, J., Rasmussen, C.E.: Propagation of uncertainty in bayesian kernel models - application to multiple-step ahead forecasting. In: IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), vol. 2, pp. II–701 (2003). DOI 10.1109/ICASSP.2003.1202463
2. Deisenroth, M., Rasmussen, C.E.: Pilco: A model-based and data-efficient approach to policy search. In: Proceedings of the 28th International Conference on machine learning (ICML), pp. 465–472 (2011)
3. Deisenroth, M.P.: Efficient reinforcement learning using gaussian processes. Ph.D. thesis (2010)
4. Deisenroth, M.P., Fox, D., Rasmussen, C.E.: Gaussian processes for data-efficient learning in robotics and control. IEEE transactions on pattern analysis and machine intelligence **37**(2), 408–423 (2015)
5. Deisenroth, M.P., Huber, M.F., Hanebeck, U.D.: Analytic moment-based gaussian process filtering. In: Proceedings of the 26th annual international conference on machine learning, pp. 225–232. ACM (2009)
6. Gal, Y., McAllister, R., Rasmussen, C.E.: Improving pilco with bayesian neural network dynamics models. In: Data-Efficient Machine Learning workshop, ICML, vol. 4 (2016)
7. Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., Silver, D.: Rainbow: Combining improvements in deep reinforcement learning. In: Thirty-Second AAAI Conference on Artificial Intelligence (2018)
8. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization (2014). URL: `http://arxiv.org/abs/1412.6980`
9. Levine, S., Pastor, P., Krizhevsky, A., Ibarz, J., Quillen, D.: Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. The International Journal of Robotics Research **37**(4-5), 421–436 (2016)
10. Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., Kavukcuoglu, K.: Asynchronous methods for deep reinforcement learning. In: International conference on machine learning, pp. 1928–1937 (2016)
11. OpenAI: OpenAI Baselines: ACKTR & A2C (2017). URL: `https://blog.openai.com/baselines-acktr-a2c/`
12. Schulman, J., Moritz, P., Levine, S., Jordan, M., Abbeel, P.: High-dimensional continuous control using generalized advantage estimation. In: Proceedings of the International Conference on Learning Representations (ICLR) (2016)
13. Tieleman, T., Hinton, G.: Lecture 6.5 - RMSProp, COURSERA: Neural Networks for Machine Learning. Tech. rep. (2012). URL: `https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`
14. Wang, Z., Bapst, V., Heess, N., Mnih, V., Munos, R., Kavukcuoglu, K., de Freitas, N.: Sample efficient actor-critic with experience replay. arXiv preprint arXiv:1611.01224 (2016)
15. Wu, Y., Mansimov, E., Grosse, R.B., Liao, S., Ba, J.: Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. In: Advances in neural information processing systems (NIPS), pp. 5279–5288 (2017)
16. Yahya, A., Li, A., Kalakrishnan, M., Chebotar, Y., Levine, S.: Collective robot reinforcement learning with distributed asynchronous guided policy search. In: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 79–86. IEEE (2017)