Extended Abstract: Reinforcement Learning for Cyber-Physical Systems by Exploiting Prior Information

PHD FORUM SUBMISSION

John Wikman
Department of Computer Science
KTH Royal Institute of Technology
Stockholm, Sweden
jwikman@kth.se

***Abstract*—In this extended abstract, we present ongoing work on two papers and visions for future work within the scope of the thesis.**

***Index Terms*—reinforcement learning, cyber-physical systems, domain-specific languages**

I. Introduction

Deep reinforcement learning (RL) has demonstrated learning control policies, also called agents, that match or exceed the performance of humans in games such as Atari [16] and Go [20]. However, similar success stories are absent when it comes to cyber-physical systems (CPS) [10], such as humanoid robots. Viral demonstrations in robotics often use manually constructed control policies [12]. The possibility of combining deep learning techniques such as image, audio, and text recognition with RL for CPS opens up for robots to perform human-level tasks, with significant benefits such as replacing humans in dangerous work environments where the task is too difficult to encode manually or when the robot needs to learn and adapt to a changing environment.

A likely key reason why RL is more difficult for CPS is that real data is less accessible than simulated data. State-of-the-art RL algorithms such as PPO [19] and SAC [8] usually require millions of samples before learning a decent policy, as well as requiring the system to explore good and bad states. Not only is this time-consuming, but depending on the CPS it can also be expensive or dangerous when exploring bad states. An active area of research trying to address this problem is called *sim2real* (or *sim-to-real*) [21, 24], where the policy is learned on a simulated version of the system, and then transferred and applied to the real system. While this solves the sample efficiency and cost problems, even small inaccuracies in the simulation against the real system can cause a policy that is good in simulation to fail when applied to the real system.

In the scope of our thesis, we investigate topics related to the following problem:

> Design and implement reinforcement learning methods that exploit prior information about a system, and demonstrate that those methods work on physical systems to solve a desired task.

We present ongoing work in two papers as well as topics of interest for future research. The first paper investigates an RL methodology of exploiting the knowledge of delayed interaction of a physical system to learn a better policy, while the latter paper investigates a method of exploiting access to the CAD design of a system to streamline the development of learned control policies for users not necessarily familiar with simulation or RL.

A key goal with the thesis is to combine the developed methods and evaluate them on a physical humanoid robot, performing tasks such as standing up from lying down, and walking.

II. Ongoing Work

A. Interaction Delayed Reinforcement Learning

A problem with the standard formulation of RL is that it assumes that either interaction is instantaneous or that the relevant parts of our surroundings remain unaffected unless we (the agent) act upon it, neither of which may be true for physical systems such as a self-driving car or an autonomous drone. In the standard formulation, the agent observes the state of the system (reading the sensors), generates an action based on that state, and applies the action on the same system state that was observed. In reality, there will be a delay in observing the state, in deciding on an action, and in applying the action to the environment [3, 23], the combination of which we refer to as the *interaction delay* which is visualized in Figure 1.

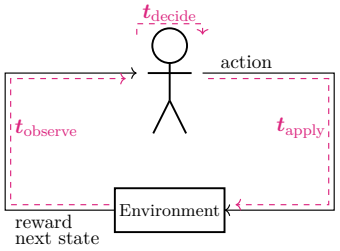

Fig. 1. Illustration of contibuting factors to interaction delay.

An example of interaction delay would be an autonomous drone where necessary computational resources cannot fit on the drone. The drone would then wirelessly transmit video and sensor readings to a computational server, the server would spend time inferring an action based on a deep neural network, and then send the action back to the drone to be applied to its motors. The negative effect that this interaction delay has on the agent unless accounted for has been demonstrated both in simulation and for physical systems [21, 11].

A common approach used by previous work is the concept of an action buffer [14, 7, 18, 5, 1, 11], which is assumed to be able to immediately apply an action as the state is being observed, for example by being implemented on a small microcontroller on the system itself. In this setting, the agent instead sends actions ahead of time to this action buffer, which is responsible for applying the action at the intended time.

There have been several algorithms proposed where the actions are generated ahead of time, either by assuming that the delay is constant [7, 18, 5, 11] or observable by the agent [1]. We investigate a more realistic scenario, where an unobservable random interaction delay varies as we interact with the environment.

Our solution to this setting learns a predictive model of the environment, represented as a deep neural network, which can predict states for many different possible steps ahead and generate actions based on those predictions. The advantage of this predictive approach is that it can generate actions for any length of delay, whereas previous approaches focus on learning policies which only generate actions for a specific delay [18, 1, 11]. While this kind of predictive solution has been disregarded by previous work for not performing well [11], our preliminary results show that this approach does work with the right kind of neural network architecture.

B. Streamlining RL for Cyber-Physical Systems

As previously discussed, sim2real is a method in which agents are learned in simulation before being applied to real robots as a way to make training faster, safer, and cheaper. These are properties desirable for those designing domain-specific robots on a smaller scale, which cannot afford to manufacture large amounts of robots for testing. However, the sim2real methodology requires that a simulator and RL environment is created for the system, neither of which may be expertise that the domain experts possess.

We envision a methodology which abstracts the simulation and RL behind an annotation interface, where the domain experts annotate a CAD model of the system with properties such as materials, servos, and the high-level objective that it wants the system to achieve (e.g. get to the other side of the wall). A simulator and RL environment are automatically generated from these high-level annotations, which learn a policy that attempts to fulfill the annotated objective.

A key aspect we see with this approach is that it enables rapid prototyping, which contrasts much of related work which focus on commercial platforms [21, 24]. For example, if an issue is detected when evaluating the learned policy in simulation or on the real system, the CAD model or annotations are adjusted accordingly to fix the issue and the simulator and RL environment are automatically generated for the updated CAD model.

This high-level interface poses several technical challenges, such as how to handle hard-to-model aspects including friction, flexibility of materials, actuator power consumption, etc. There is also the issue of generating sufficiently informative reward signals, which is regarded as a non-trivial problem. We look at various ways of overcoming this, such as allowing for fine-grained annotations of soft and hard constraints. A soft constraint is something that is discouraged but not forbidden, whereas a hard constraint is strictly forbidden from happening. We have evaluated these kinds of constraints on a custom-designed quadruped robot and found that they make a significant difference in the quality of the learned agent both in simulation and when evaluated in the real world.

III. Topics for Future Work

An issue with RL, which is not as apparent in regular supervised learning, is that there is a lot of custom code created for each problem that guides the learning process. The code used for learning also suffers from being implemented in different backends that do not efficiently integrate, such as PyTorch [17] for training and MuJoCo [22] for simulation. This issue has also been highlighted by Nvidia, who developed Isaac Gym to allow training and simulation to be efficiently run in the same backend [15].

A possible joint interface that solves this issue is domain-specific languages (DSL) which translates a high-level description of the training to generate code that runs on a joint backend. While frameworks such as Isaac Gym run on a unified backend, it is restricted by the API to that backend. Having a DSL that translates code into a unified backend allows the user to specify things such as custom environment dynamics using differential equations. The compilation could target high-level machine learning backends such as JAX [2], PyTorch [17], or something lower level like MLIR [13]. Apart from enabling more efficient execution, a joint backend also opens up for adaptation of RL algorithms that require specific properties from the environment, such as differentiable transitions [9].

This holistic approach also opens up opportunities such as statically checking correctness in various aspects of the training procedure which is highly sensitive to errors. Challenges with this include finding the right level of abstraction that allows for flexibility in the language while still being able to reduce subtle, but critical errors such as incorrect post-transform probabilities, optimization targets, broadcasting operations, etc.

Additional opportunities include the ability to more easily compose different RL methodologies, such as the more established online RL algorithms with the more recent offline RL algorithms [4, 6], where the latter allows the agent to learn without having to interact with the environment.

REFERENCES

[1] Yann Bouteiller et al. "Reinforcement Learning with Random Delays". In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=QFYnKlBJYR.

[2] James Bradbury et al. *JAX: composable transformations of Python+NumPy programs*. Version 0.3.13. 2018. URL: http://github.com/google/jax.

[3] D.M. Brooks and C.T. Leondes. "Technical Note - Markov Decision Processes with State-Information Lag". In: *Operations Research* 20.4 (Aug. 1972), pp. 904–907. DOI: 10.1287/opre.20.4.904.

[4] Arunkumar Byravan et al. "Imagined Value Gradients: Model-Based Policy Optimization with Tranferable Latent Dynamics Models". In: *Proceedings of the Conference on Robot Learning*. Ed. by Leslie Pack Kaelbling, Danica Kragic, and Komei Sugiura. Vol. 100. Proceedings of Machine Learning Research. PMLR, 30 Oct–01 Nov 2020, pp. 566–589. URL: https://proceedings.mlr.press/v100/byravan20a.html.

[5] Baiming Chen et al. "Delay-aware model-based reinforcement learning for continuous control". In: *Neurocomputing* 450 (Apr. 2021), pp. 119–128. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2021.04.015. URL: https://www.sciencedirect.com/science/article/pii/S0925231221005427.

[6] Lili Chen et al. "Decision Transformer: Reinforcement Learning via Sequence Modeling". In: *Advances in Neural Information Processing Systems*. Ed. by M. Ranzato et al. Vol. 34. Curran Associates, Inc., 2021, pp. 15084–15097. URL: https://proceedings.neurips.cc/paper_files/paper/2021/file/7f489f642a0ddb10272b5c31057f0663-Paper.pdf.

[7] Vlad Firoiu, Tina Ju, and Josh Tenenbaum. "At Human Speed: Deep Reinforcement Learning with Action Delay". In: *CoRR* abs/1810.07286 (2018). arXiv: 1810.07286. URL: http://arxiv.org/abs/1810.07286.

[8] Tuomas Haarnoja et al. "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, Oct. 2018, pp. 1861–1870. URL: https://proceedings.mlr.press/v80/haarnoja18b.html.

[9] Nicolas Heess et al. "Learning Continuous Control Policies by Stochastic Value Gradients". In: *Advances in Neural Information Processing Systems*. Ed. by C. Cortes et al. Vol. 28. Curran Associates, Inc., 2015. URL: https://proceedings.neurips.cc/paper/2015/file/148510031349642de5ca0c544f31b2ef-Paper.pdf.

[10] Julian Ibarz et al. "How to train your robot with deep reinforcement learning: lessons we have learned". In: *The International Journal of Robotics Research* 40.4-5 (2021), pp. 698–721. DOI: 10.1177/0278364920987859. eprint: https://doi.org/10.1177/0278364920987859. URL: https://doi.org/10.1177/0278364920987859.

[11] Jangwon Kim et al. "Belief Projection-Based Reinforcement Learning for Environments with Delayed Feedback". In: *Advances in Neural Information Processing Systems*. Ed. by A. Oh et al. Vol. 36. Curran Associates, Inc., 2023, pp. 678–696. URL: https://proceedings.neurips.cc/paper_files/paper/2023/file/0252a434b18962c94910c07cd9a7fecc-Paper-Conference.pdf.

[12] Scott Kuindersma et al. "Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot". In: *Autonomous robots* 40 (2016), pp. 429–455.

[13] Chris Lattner et al. "MLIR: Scaling Compiler Infrastructure for Domain Specific Computation". In: *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. 2021, pp. 2–14. DOI: 10.1109/CGO51591.2021.9370308.

[14] Rogelio Luck and Asok Ray. "An observer-based compensator for distributed delays". In: *Automatica* 26.5 (1990), pp. 903–908. ISSN: 0005-1098. DOI: https://doi.org/10.1016/0005-1098(90)90007-5. URL: https://www.sciencedirect.com/science/article/pii/0005109890900075.

[15] Viktor Makoviychuk et al. *Isaac Gym: High Performance GPU-Based Physics Simulation For Robot Learning*. 2021. arXiv: 2108.10470 [cs.RO]. URL: https://arxiv.org/abs/2108.10470.

[16] Volodymyr Mnih et al. "Playing Atari with Deep Reinforcement Learning". In: *CoRR* abs/1312.5602 (2013). arXiv: 1312.5602. URL: http://arxiv.org/abs/1312.5602.

[17] Adam Paszke et al. *Automatic differentiation in pytorch*. 2017.

[18] Simon Ramstedt and Chris Pal. "Real-Time Reinforcement Learning". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper/2019/hash/54e36c5ff5f6a1802925ca009f3ebb68-Abstract.html.

[19] John Schulman et al. "Proximal Policy Optimization Algorithms". In: *CoRR* abs/1707.06347 (2017). arXiv: 1707.06347. URL: http://arxiv.org/abs/1707.06347.

[20] David Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587 (Jan. 2016), pp. 484–489. ISSN: 1476-4687. DOI: 10.1038/nature16961. URL: https://doi.org/10.1038/nature16961.

[21] Jie Tan et al. *Sim-to-Real: Learning Agile Locomotion For Quadruped Robots*. 2018. arXiv: 1804.10332 [cs.RO].

[22] Emanuel Todorov, Tom Erez, and Yuval Tassa. "MuJoCo: A physics engine for model-based control". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012, pp. 5026–5033. DOI: 10.1109/IROS.2012.6386109.

[23] Thomas J. Walsh et al. "Learning and planning in environments with delayed feedback". In: *Autonomous Agents and Multi-Agent Systems* 18.1 (July 2008), p. 83. ISSN: 1573-7454. DOI: 10.1007/s10458-008-9056-7. URL: https://doi.org/10.1007/s10458-008-9056-7.

[24] Wenhao Yu et al. *Sim-to-Real Transfer for Biped Locomotion*. en. arXiv:1903.01390 [cs]. Aug. 2019. URL: http://arxiv.org/abs/1903.01390 (visited on 04/30/2024).