26th International Conference
ENGINEERING MECHANICS 2020
Brno, Czech Republic, November 24 – 25, 2020

AIR SPRING CONTROLLED BY REINFORCEMENT LEARNING ALGORITHM

Rágulík J.[*], Sivčák M.[**]

Abstract: *The paper deals with the replacement of the analogy PID stroke controller of a bellows pneumatic spring, by machine learning algorithms, specifically deep reinforcement learning. The Deep Deterministic Policy Gradient (DDPG) algorithm used consists of an environment, in this case a pneumatic spring, and an agent which, based on observations of environment, performs actions that lead to the cumulative reward it seeks to maximize. DDPG falls into the category of actor-critic algorithms. It combines the benefits of Q-learning and optimization of a deterministic strategy. Q-learning is represented here in the form of critic, while optimization of strategy is represented in the form of an actor that directly maps the state of the environment to actions. Both the critic and the actor are represented in deep reinforcement learning by deep neural networks. Both of these networks have a target variant of themselves. These target networks are designed to increase the stability and speed of the learning process. The DDPG algorithm also uses a replay buffer, from which the data from which the agent learns is taken in batches.*

Keywords: Air spring, Deep deterministic policy gradient, DDPG, Reward function design.

1. Introduction

Deep Deterministic Policy Gradient (DDPG) is an actor-critic algorithm based on the deterministic policy gradient, that can operate over continuous action spaces (Sutton et al., 2000) and (Silver et al., 2014). It is an algorithm which concurrently learns a Q-function - critic and a policy - actor. It uses off-policy data and the Bellman equation to learn the Q-function, and uses it to learn the policy.

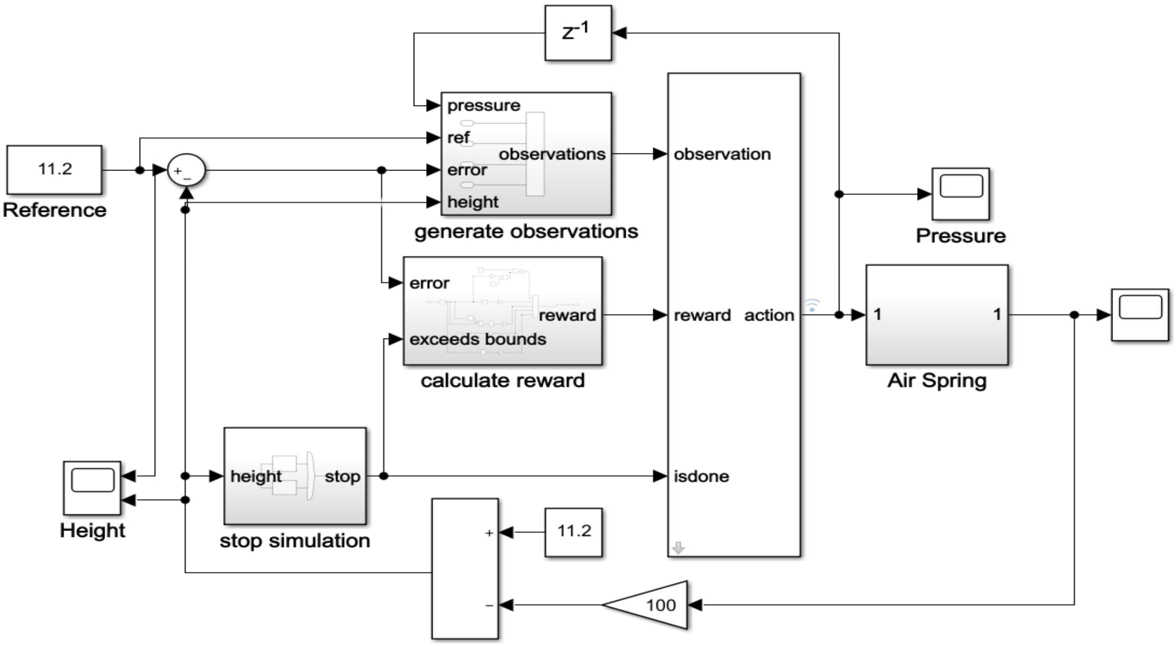

Fig. 1: Schematic representation of the environment and the agent.

[*] Bc. Jiří Rágulík: Technical University of Liberec, Studentská 2; 461 17, Liberec; CZ, jiri.ragulik@tul.cz

[**] Ing. Michal Sivčák, PhD.: Technical University of Liberec, Studentská 2; 461 17, Liberec; CZ, michal.sivcak@tul.cz

It uses four neural networks. A Q network, a deterministic policy network, a target Q network, and a target policy network. The target networks are time-delayed copies of their original networks that slowly track the learned networks (Konda et al., 2000). Using these target value networks greatly improve stability of learning. In DDPG, the actor directly maps states to actions (Lillicrap et al., 2015).

The algorithm was written in Matlab and the air spring model was created in Simulink. The bellows air spring model was taken from the paper (Rágulík and Sivčák, 2019). A schematic representation of the entire environment and the agent is pictured in Fig. 1.

Total of 4 observations are presented to the agent, namely the required height, the deviation from the required height, the actual height and the agent's action (pressure) derived in the previous episode.

2. Mini-batch size

Learning takes place by adjusting the scales in deep neural networks. DDPG is an actor and critic. The data from the replay buffer, on the basis of which the learning takes place, are delivered in batches of the specified size (Mnih et al., 2015). In practice, the most commonly used dimensions are 32, 64 and 128. During each training episode, the agent randomly samples data from the experience buffer when computing gradients for updating the neural networks. Large mini-batch usually reduce the variance but increase the computational effort (Islam et al., 2017).

Since the mini-batch size is usually chosen from the sequence 2^n, in combination with the chosen learning rate, the size 256 proved to be optimal, because at the size 512 there was practically no improvement, but the computational complexity increased significantly and at the size 128 Although computational complexity has decreased, the time required to find the optimal strategy has increased significantly.

3. Learning rate

The amount that the weights are updated during training is referred to as the learning rate. It is a hyperparameter used in the training of neural networks that has a small positive value, often in the range between 10^{-6} and 1. The learning rate controls how quickly the model is adapted to the problem. Smaller learning rate may result in a long training process that could get stuck, whereas larger learning rates result in rapid changes and require fewer training epochs, but may result in learning a sub-optimal set of weights too fast or an unstable training process (Islam et al., 2017).

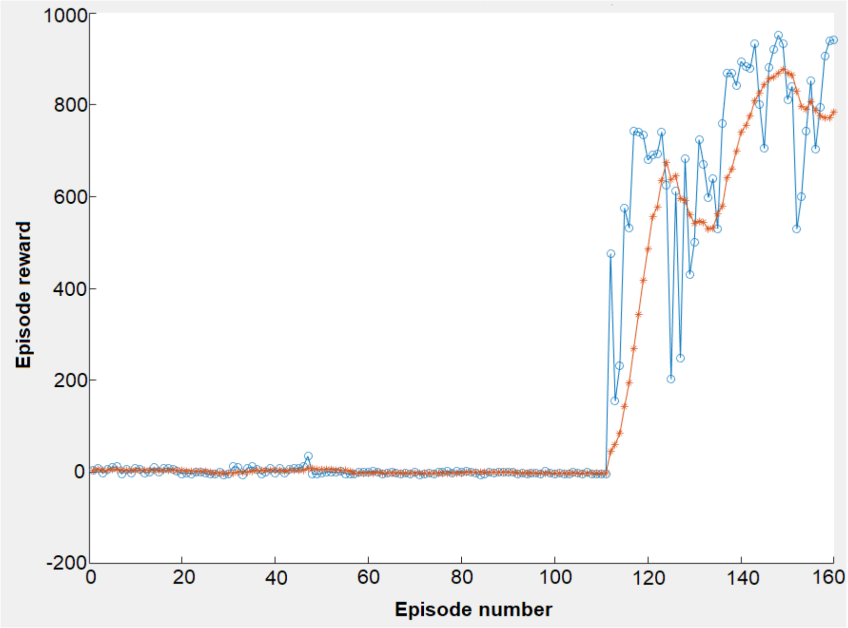

Fig. 2: Course of reward and average value (blue) for the last 20 episodes (red).

The optimal learning rate cannot be calculated in advance, it must be determined on the basis of experience by trial and error. In our case, a learning rate of 10^{-3} was used. When using a learning rate of 10^{-2}, a suboptimal strategy was always achieved, moreover, in a very long time, which does not exactly correspond to the theory that when using too high a learning rate, the final strategy should be achieved relatively quickly. On the contrary, using the learning rate 10^{-5} algorithm, he was not able to find a suitable strategy even after 1000 episodes. Using a learning rating of 10^{-3}, the discovery of a suitable strategy was achieved after approximately 50 episodes, and this strategy was further improved. After 150 episodes, the required average of rewards for the last 20 episodes was achieved (Fig. 2), which is a parameter that very well describes learning outcomes.

4. Reward function design

Probably the most important task in applying the DDPG algorithm is the design of the reward signal. By obtaining it, the agent finds out how good his last action was and how he should change it to achieve a higher reward (Islam et al., 2017).

When designing the reward signal, it was clear that the agent should receive the maximum reward at zero deviation from the desired height. It was also necessary to respect the physical possibilities of a real pneumatic spring, because its mathematical model allows almost any stretching or compression. Exceeding the physical possibilities leads to the termination of the episode and a high penalty. The system should therefore try to keep height in this interval at all costs. The reward signal is calculated only from the deviation. The course of the reward should have a concave shape. If, for example, a linear dependence was used, the system would not be sufficiently motivated to improve. A possible improvement in the reward signal would be a penalty for rapid pressure fluctuations, resulting in higher compressed air consumption. However, this is not necessary here, because thanks to the simplicity of the whole system, the agent will find a solution in which the pressure will not fluctuate unnecessarily, automatically. This is guaranteed by two cycles of learning. The first cycle consists in teaching the agent to maintain the required height without variable external loading. The agent thus finds the pressure at which the spring reliably holds the height, and during the second cycle, when the system is excited by the change in weight, the pressure only compensates for the error introduced by the load.

The reward function was designed as follows:

$$r_t = 3 \cdot$$

The first part of the reward function rewards (r_t) the agent for the smallest possible deviation (e_t), the second part adds another reward for staying in the interval of 2 mm. The third part rewards the agent when the deviation from the previous step (e_{t-1}) is reduced. This part of the reward is very treacherous because it can lead to the algorithm oscillating around zero deviation with a high frequency to obtain the maximum reward, in order to obtain a higher reward than for remaining at zero deviation. This is prevented by the fourth part of the reward signal, which rewards the agent for staying at zero deviation. The last part of the remuneration function penalizes the agent for exceeding the set limits (EB), determined by the physical possibilities of air spring.

5. Conclusions

The settings of the mentioned mini-batch size, learning rate and many other parameters were set, the values of which were inspired by the values in (Lillicrap et al., 2015) and adjusted for the needs of the control of the given mechanical system. The correct setting of the topology of deep neural networks and their size was quite problematic. The size was again inspired by the (Lillicrap et al., 2015) and then reduced until the minimum size required for successful spring control was revealed. Reducing the size of neural networks by a quarter has resulted in a huge reduction in computational complexity. In the future, we would like to replace the DDPG algorithm, which was the first to be used for the problem of continuous action space, to replace it with the Trust Region Policy optimization (TRPO) algorithm (Shulman et al., 2015), which was created in response to the shortcomings of the DDPG algorithm.

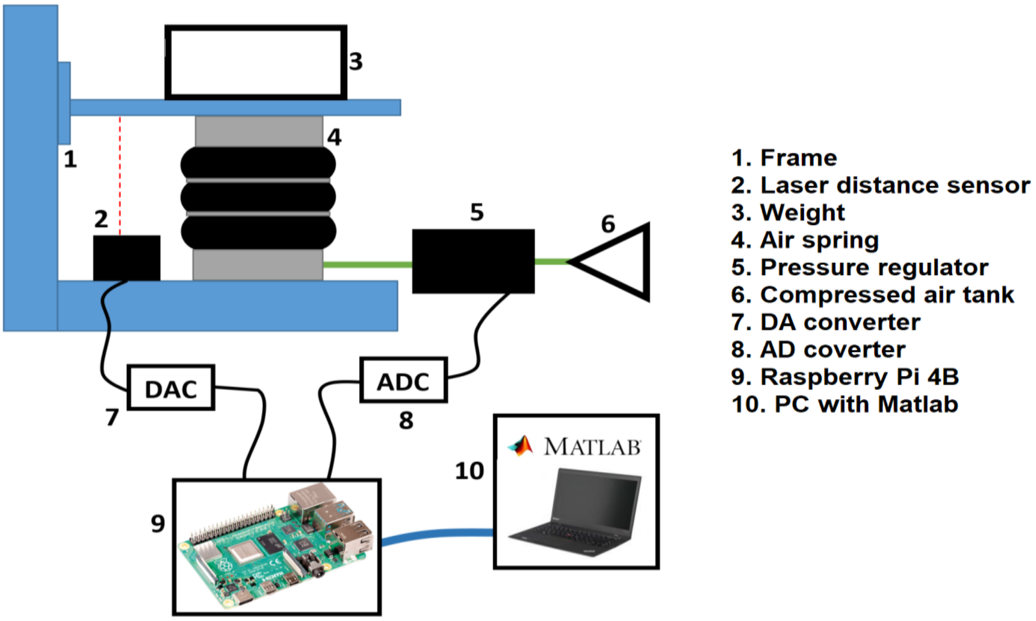

Fig. 3: Scheme of the experimental device.

The next planned step is online learning. In this case, the agent would be trained on the model in Matlab and then trained on a real spring, clamped in a test stand. After learning on a real spring, a properly tuned agent would affect the neglected or simplified characteristics of the pneumatic system. An experimental device has already been designed for this purpose (Fig. 3) and its assembly is planned in the future.

Acknowledgement

This work was supported by the Student Grant Competition of the Technical University of Liberec under the project No. SGS-2019-5072.

References

Islam, R., Henderson, P., Gomrokchi, M. and Precup, D. (2017) Reproducibility of Benchmarked Deep Reinforcement Learning Tasks for Continuous Control. Reproducibility in Machine Learning Workshop. ICML 2017. arXiv:1708.04133.

Konda, V. and Gao, V. (2000) Actor-critic algorithms.

Lillicrap, T., Hunt, J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D. and Wierstra, D. (2015) Continuous control with deep reinforcement learning. CoRR.

Matheron, G., Perrin, N. and Sigaud, O. (2019) The problem with DDPG: understanding failures in deterministic environments with sparse rewards. ICLR 2020. arXiv:1911.11679.

Mnih, V. et al. (2015) Human-level control through deep reinforcement learning. Nature. 518. 529-33. 10.1038/nature14236.

Rágulík, J. and Sivčák, M. (2019) Modeling of the Controlled Air Spring. Strojnícky časopis - Journal of Mechanical Engineering, 69(3), pp. 107-112. Retrieved 18 Nov. 2019, from doi:10.2478/scjme-2019-0037

Sutton, R., Mcallester, D., Singh, S. and Mansour, Y. (2000) Policy Gradient Methods for Reinforcement Learning with Function Approximation. Adv. Neural Inf. Process. Syst. 12.

Schulman, J., Levine, S., Moritz, P & Jordan, M. and Abbeel, P. (2015) Trust Region Policy Optimization. ICML 2015. arXiv:1502.05477.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D. and Riedmiller, M. (2014) Deterministic Policy Gradient Algorithms. 31st International Conference on Machine Learning, ICML 2014.