Reinforcement learning in real-time geometry assurance

Available online at www.sciencedirect.com

ScienceDirect

Procedia CIRP 72 (2018) 1073–1078

www.elsevier.com/locate/procedia

51st CIRP Conference on Manufacturing Systems

Reinforcement learning in real-time geometry assurance

Emilio Jorge[a,*], Lucas Brynte[a], Constantin Cronrath[b], Oskar Wigström[b], Kristofer Bengtsson[b], Emil Gustavsson[a], Bengt Lennartson[b], Mats Jirstrand[a]

[a] *Fraunhofer-Chalmers Centre for Industrial Mathematics, SE-412 88 Gothenburg, Sweden*

[b] *Chalmers University of Technology, Department of Electrical Engineering, SE 412-96 Gothenburg, Sweden*

[*] Corresponding author. *E-mail address:* emilio.jorge@fcc.chalmers.se

Abstract

To improve the assembly quality during production, expert systems are often used. These experts typically use a system model as a basis for identifying improvements. However, since a model uses approximate dynamics or imperfect parameters, the expert advice is bound to be biased. This paper presents a reinforcement learning agent that can identify and limit systematic errors of an expert systems used for geometry assurance. By observing the resulting assembly quality over time, and understanding how different decisions affect the quality, the agent learns when and how to override the biased advice from the expert software.

Peer-review under responsibility of the scientific committee of the 51st CIRP Conference on Manufacturing Systems.

Keywords: geometry assurance; reinforcement learning; expert advice;

1. Introduction

Geometrical variations of parts or products is a common root cause for product quality problems. For that reason, software tools and techniques aimed at minimising the effects of geometrical variations has been developed within the field of geometry assurance and are used throughout the product realisation cycle [1].

In addition to increased computational capacity, advances in simulation and optimisation algorithms have lowered the computation time of geometry assurance tasks from weeks to hours and minutes in recent years. In [2], an online geometry assurance algorithm used during production is therefore proposed. This algorithm will improve the production processes and part matching for each individual assembly task. This algorithm is referred to as the expert system, or the expert in this paper.

The online geometry assurance expert must be able to adjust various processes, for example fixture locator positions or task sequences, as well as decide what parts should be assembled together. However, since the expert is based on an approximated and simplified model of the real system or inaccurate input parameters, the control signal from the expert may not always be correct. To reduce unnecessary residual geometrical variation due to modelling errors or approximate dynamics in the expert, some form of feedback loop from the real system is required.

Within Adaptive Control, one approach suiting these requirements is Reinforcement Learning (RL). RL is a quickly growing research area, which deals with optimal decision making in uncertain environments, e.g. for autonomous driving, or for playing games such as Backgammon [3], Go [4], and even computer games [5]. It has been shown that RL can be applied also in manufacturing for scheduling [6–8], maintenance [9], and ramp-up optimisation [10]. Typically, an agent (decision maker) learns to act independently through exploration of an environment and through exploitation of previously acquired knowledge [11]. In this way, RL allows for high-dimensional data – like the data present in geometry assurance tasks – and its algorithms can be model-free. Therefore, RL is considered to be well-suited for the problem at hand.

This paper presents the first steps towards an RL agent that can utilise the knowledge from the geometry assurance expert and the feedback from the real system, to improve the assembly quality of the products. More specifically, an RL agent is designed that acts in a continuous action space, to emulate the tuning of an adjustable fixture. Important architectural considerations are discussed, and using a simple low dimensional example, the characteristics of our agent architecture is explored.

The remainder of this paper is structured as follows. Section 2 provides an introductory background on RL and Section 3 describes our the architecture of our RL agent. A computational example is detailed in Section 4, and finally the paper is concluded in Section 5.

2212-8271
Peer-review under responsibility of the scientific committee of the 51st CIRP Conference on Manufacturing Systems.
10.1016/j.procir.2018.03.168

2. Reinforcement learning

Reinforcement learning has been around for a long time but recent high-profile papers [4,5,12] have brought it back into the public eye. In RL, an agent acts in an environment, from which it observes both its current state and the outcome (so called reward) of its actions. Based on these observations, the goal is to learn better actions in the future such that the reward (which in the context of this paper refers to production quality) is maximized.

More formally this can be defined as following. Let S be a set of states that define an Markov Decision Process where each $s \in S$ contains a set of permitted actions $A(s)$. Given a state s the agent is tasked with selecting an action a, which transitions s to s' and yields the reward $R(s, a)$. In many settings the agent not only needs to take into account the immediate reward but also the possible rewards in the new state s', although in this paper the example focuses on the case where s' is always the final state in which no future rewards can be observed. The RL agent iteratively learns a policy for selecting actions by observing the rewards for the state and actions pairs. In other words, the agent learns by iteratively improving estimates of the expected value (i.e. rewards) of its actions and which actions yield the best value. If the actions are selected according to the agent's policy during learning, it is known as on-policy learning. If the actions partly (or totally) deviate from the policy, it is known as an off-policy algorithm. The learning process in many RL algorithms require that the learning is performed on-policy, restricting how negative rewards – i.e. losses, occurred during training – can be limited by restricting the agents action until its performance is satisfactory.

In the case considered in this paper, an expert is already in place and ought to be utilised. Within RL, work on behavioural cloning [13,14] and imitation learning [15,16] attempt to replicate an expert, teacher, or behaviour. Also the concept of inverse reinforcement learning could be used to learn and draw conclusions about the expert's behaviour [17–19].

However, the objective is to find an agent which does not only mimic the expert, but actually surpasses the expert's ability. An example with the same aim can be found in [20], where the authors improve the solution to structured prediction problems by occasionally following a given reference policy. In that particular case, bounds on the regret (i.e., the deviation from an optimal policy) can be guaranteed even if the reference policy of the expert is sub-optimal. Our interest though, lies more in the practical aspect of quickly improving upon the expert. The expert's actions are thus regarded as additional information that should be used by the RL agent.

Another motivation for many RL approaches are often to replace the expert, since it is expensive or impractical to use. An example of this is [21] where they use previous examples from an expert to as a method for demonstration and pre-training to speed up the agents learning process but do not allow the agent to further interact with the expert during training. The goal of this paper, however, is to combine the two methods to obtain improved performance, instead of replacing the expert. Rather than teaching the agent to mimic the expert, before attempting to improve upon it, we propose that the agent instead increments the expert's control action. This should result in a simpler representation, since the information comprising the expert does not need to be modelled by the agent.

Further information on RL can be found in [11]. A description of the algorithm for achieving learning in this paper can be found in Subsection 3.3.

3. The design of the RL agent

In this paper, a simplified use case is used for evaluating the suggested approach. The expert has a possibility to adjust a fixture that influenced the final product geometry.

The control actions from a geometry assurance expert could be both discrete (task order and part matching) and continuous (fixture adjustment and positioning). But since this paper focuses on the specific case of an adjustable fixture, the RL agent will only have a continuous action space.

3.1. Continuous action space

RL is often used for tasks with action spaces that are small and discrete or for problems that have been approximated through discretisation [22,23]. Although most algorithms are created for discrete actions, there are also a few that handle continuous action spaces. One of those is Deep Deterministic Policy Gradient (DDPG) [12], which is used in this paper. DDPG has successfully been applied to a variety of continuous control problems and falls within the family of actor-critic algorithms [11]. Actor-critic algorithms are characterized by the agent consisting of two parts; the actor learns to take actions and the critic learns to evaluate how good an action is, such that it can give feedback to the actor. Additionally DDPG is an off-policy algorithm [12], which allows for sampling actions according to any distribution (such as one that depends heavily on the expert), without affecting the learning of the agent.

3.2. Sample efficiency

Another important consideration is sample efficiency. In certain settings, many RL algorithms use several millions of samples for training [24]. From the geometry assurance perspective, it is not enough just to perform well after learning for many iterations, the agent also must not incur unreasonable quality defects, since the agent is changing the actual results of the production process. This problem is difficult to address, but one mechanism that improves this aspect, is experience replay that has been used to improve stability of learning [5,12]. Experience replay allows the agent to keep a replay memory of previous experiences to learn from. Each time the agent takes an action, the tuple (state, action, resulting reward) is stored in a database. The training of the agent is then purely done by sampling tuples from this memory and training with them. This replay memory can have a maximum size, removing the oldest examples as new data is observed. In [25,26] they improve the sample efficiency for tasks with discrete actions by allowing the agent to remember previous performance of the different actions in state similar to its current state and using this to guide exploration.

Additionally, a mechanism which restricts the agents decision space is required, since the RL agent does not learn to mimic the expert. The incorporation of such pretraining is discussed in the next subsection.

3.3. The agent implementation

Based on the above discussion, the RL agent is based on the DDPG algorithm, which is an actor-critic algorithm. One of the benefits of DDPG is that it allows for off-policy learning, which is important for ensuring good performance during the training phase.

The RL-agent is divided into an actor part and one critic part. The actor part will learn to take actions to adjust the fixture based on the proposed action from the expert as well as the measured result from the real process.

The actor

Given a measurement $\hat{p}$ and a proposed action (expert advice) a_E (which together become the agent's observation of the state), the agent may either use the advice (case 1), or improve upon it based on the information in $\hat{p}$ (case 2). A time-varying parameter p_A, will determine the probability for the RL agent to override the expert advice, using case 2 instead of case 1. p_A increases from 0 to 1 over an interval $[k_1, k_2]$ based on $\frac{1}{2}\left(1 - \cos\left(\frac{k-k_1}{k_2-k_1}\pi\right)\right)$, where k is the current iteration, and is constant 0, or 1 respectively, outside the interval.

The agent's action a is sampled from

$$W \sim \text{Lognormal}(\mu, \sigma^2), \text{[1]} \tag{1}$$

where μ depends on whether the expert's advice is used (Eq. 2) or overridden (Eq. 3) The extent to which the agent explores the action space is determined by the parameter σ. Note that some exploration is needed for training stability, even when using the expert advice.

Case 1: - Use Expert Advice

The action a is sampled from W according to (1), with

$$\mu = \log(a_E) - \frac{\sigma^2}{2}, \tag{2}$$

in order to fulfil $E[W] = a_E$.

Case 2: - Override Expert Advice

In this case, a deviation is added to μ:

$$\mu = \log(a_E) - \frac{\sigma^2}{2} + \Delta(\hat{p}, a_E; \theta_\Delta), \tag{3}$$

where Δ is a neural network parameterized by θ_Δ, and will be referred to as the actor.

[1]The lognormal distribution is selected because only positive actions are considered in this paper. An alternative distribution could be used instead in other applications.

The critic

The actor is complemented by a critic, which given the observation $\hat{p}$, expert advice a_E and an action a, estimates the action value function using a neural network $Q(\hat{p}, a_E, a; \theta_Q)$, parameterized by θ_Q.

For defining the loss functions, it will be helpful to first derive the expected value of W, conditioned on overriding the expert advice (case 2):

$$m_W = e^{\mu + \frac{\sigma^2}{2}} = a_E e^{\Delta(\hat{p}, a_E; \theta_\Delta)}. \tag{4}$$

Consequently, Δ can be interpreted[2] as a relative correction of a_E.

The architecture is the same for Δ and Q (but with separate weights), with three fully connected layers. The two hidden layers (32/64 units respectively) are followed by the activation $\max(0, x)$, which is commonly known as the rectified linear unit (ReLU), while the output layers have a linear activation function.

Agent training

The actor and critic each compute the difference between network output and desired output based on their own loss function, according to DDPG [12]:

$$L_\Delta = L_P + \frac{1}{N}\sum_{i=1}^{N} -Q(\hat{p}^i, a_E^i, m_W(\hat{p}^i, a_E^i; \theta_\Delta); \theta_Q) \tag{5}$$

$$L_Q = \frac{1}{N}\sum_{i=1}^{N} \frac{1}{2}\left(Q(\hat{p}^i, a_E^i, a^i; \theta_Q) - R^i\right)^2, \tag{6}$$

where N is the batch size and L_P is an extra term used for pretraining, explained below. The loss functions are needed for training the neural networks. For the critic loss L_Q, Q is evaluated at the actual action a which is fed to the environment, and for which a corresponding reward R is received. Note, however, that the actor loss L_Δ evaluates Q at m_W instead, which enables back-propagation of gradients[3].

Adam algorithm [27] is then applied on L_Δ and L_Q, with the corresponding gradients $\nabla_{\theta_\Delta} L_\Delta$ and $\nabla_{\theta_Q} L_Q$. The gradients are then clipped, limiting the Euclidian norm, before being applied to tune the network parameters. The actor and critic have their own respective learning rates described in Table 1 together with the threshold for the gradient clipping. Note that although the critic network Q is present in L_Δ as well as L_Q, its parameters θ_Q are fixed when updating θ_Δ.

Pretraining

As mentioned previously, the actor loss L_Δ incorporates a pretraining loss L_P, which we will now explain further. This

[2]It can even be shown that sampling from W with μ given by (3), is equivalent to sampling from W with μ given by (2) followed by multiplication with e^Δ.

[3]Back-propagation could not have been carried out through a, since it is randomly sampled, and thus not differentiable w.r.t. θ_Δ.

pretraining loss is to be applied in the beginning of training, and thus decaying exponentially according to:

$$L_P = 0.5^{k/H} \frac{1}{N} \sum_{i=1}^{N} \frac{1}{2} \left(\frac{a_E - m_W(\hat{p}^i, a_E^i; \theta_\Delta)}{a_E} \right)^2, \qquad (7)$$

punishing the agent for deviating too much from the expert during the initial phase of training (k is the current iteration, and the half-life H is a parameter).

4. Experimental results

The proposed agent has been evaluated using a low dimensional computational example with the following setup. The process is modelled by a function $y = f(a, p)$, where $a \in \mathbb{R}$ is a scalar action, $p \in \mathbb{R}^n$ is a vector of material parameters and $y \in \mathbb{R}$ is a scalar representing the resulting part geometry. The objective is to apply an action to achieve a nominal part geometry y_r, i.e. minimize $(y - y_r)^2$. However, neither our expert nor agent observe the true parameters p, but rather biased parameters $\hat{p} = p + b$. Additionally, the expert has a small model error such that its model $\hat{f}(a, p) \approx f(a, p)$. Given biased parameters and its imperfect model, the expert computes an action $a_E = \text{argmin}_a |\hat{f}(a, \hat{p}) - y_r|$.

As for $f(a, p)$, in this paper, a simple cantilever beam bending equation has been used

$$f(a, p) = \frac{a p_1^3}{3 p_2 p_3},$$

and with a model error

$$\hat{f}(a, p) = \frac{a p_1^{3.05}}{3 p_2 p_3},$$

where a is the force applied to the beam, p_1 the length, p_2 the modulo of elasticity, p_3 the area moment of inertia and the resulting output is the beam deflection. This can be seen as a simplified example of an actuated fixture that flexes non-nominal parts into position by applying a force.

A beam instance is generated synthetically from one in four classes of beams. All classes are equally probable, and each is characterised by a multivariate normal distribution of length and elasticity. The bias b also has a class specific value.

From an RL perspective, the case example can be regarded as a continuum armed contextual bandit [28,29]. We select the reward as $R = -(y - y_r)^2$, since the agent is set to maximize R.

The experiments are performed on variations of the default agent described in Subsection 3.3. The different variations are:

- Default
- No experience replay[4]

[4]Here the batch size is not used, the sample observed by the agent is used immediately to update the loss function rather than being stored in the replay memory.

Table 1. Default parameters for the experiments

Parameter	Value
Repeated runs for each setting	5
Iterations	30000
Actor learning rate	10^{-4}
Critic learning rate	10^{-3}
Gradient clipping threshold	10^{0}
Batch size for experience replay	8
Exploration parameter σ	0.1
Replay memory buffer	∞
Pretraining half-life H	2000
$[k_1, k_2]$ for increasing p_A	[100, 2000]

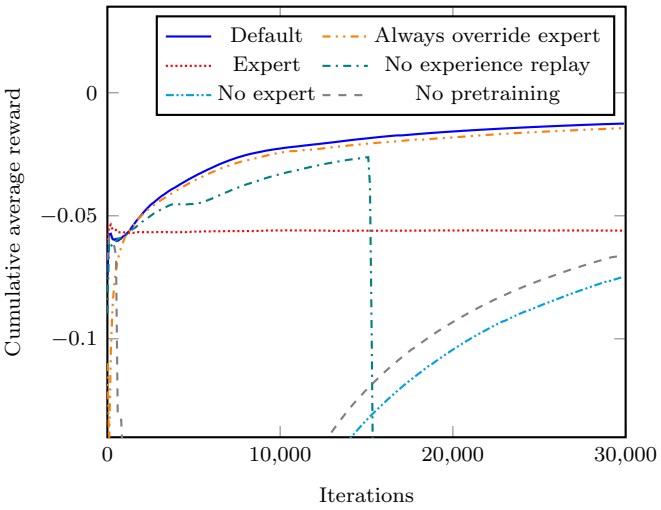

Fig. 1. The normalized cumulative reward for each agent variation. Each variation is run 5 times, after which the data is averaged over the runs.

- No pretraining (deviating from the expert is not punished)
- Always override expert (instead of slowly increasing the probability that the agent overrides the expert)
- No expert[5]

4.1. Maximizing cumulative reward

In addition to the final performance of the model, the cost of getting there is interesting in a manufacturing setting. A measure of this is the cumulative reward, normalized by the amount of samples. Figure 1 depicts the averaged results of the experiments for each variation of the agent.

It can be observed, that the proposed RL agent outperforms the expert for two variations. Table 2 clarifies where the agent variations cross the reference line (the expert by itself). Although the "No experience replay" variation performs well initially, it seems that removing this feature leads to temporal instability and worse long-term results (i.e. the "nose-dive" after approximately 15000 iterations). Furthermore, without the ini-

[5]Essentially this is a combination of "No pretraining" and "Always override expert". More specifically, $p_A = 1$ and "Case 2. - Override Expert Advice" is always used. Also, the actor/critic networks Δ and Q will no longer be fed with a_E. However, we set $a_E = 1$ in eq. (3). The actor network Δ can then no longer be interpreted as a correction of the expert advice, since eq. (4) will be changed from $m_W = a_E e^{\Delta}$ to $m_W = e^{\Delta}$. In addition, no pretraining is applied.

Table 2. Average number of iterations until the agent outperforms the expert.

Agent variation	Iterations
Default	1178
No experience replay	Did not outperform expert
No pretraining	Did not outperform expert
Always override expert	1188
No expert	Did not outperform expert

tial guidance of the expert ("No pretraining" and "No expert") large initial negative rewards incur that require many iterations to average out. For the remaining two variations ("Default" and "Always override expert"), however, a significant improvement can be observed.

Whether the actions are sampled according to Equation 2 or 3 appears (at least for our configuration in this reasonably simple case) to make little difference. For the "Always override expert" variation, the agent always samples the actions according to Equation 3 and, therefore, learns in an almost on-policy fashion. In contrast to that, the "Default" agent learns initially in an off-policy fashion, which sometimes has been found to be slower [11]. In both variations, the agent will quickly learn to behave similarly to the expert, due to pretraining. The faster learning from closer-to-policy sampling in the "Always override expert" variation, however, could perhaps offset a few bad negative rewards in the beginning[6].

Nevertheless, it can be seen that utilizing the expert advice in all possible ways (i.e. the "Default") yields in the best short- and long-term performance.

4.1.1. Improving the actions

Figure 2 shows the result for the default agent in greater detail. The relative deviation of the expert's and the agent's action from the optimal action is displayed here for each beam class. It has to be noted that the agent has no way of distinguishing between beam classes other than identifying their different and disjoint distributions. Furthermore, beam class and instance are randomly sampled in each iteration.

In all four cases the agent learns to perform about as well

[6]This also seems reasonable since the critic will have more samples near the policy to help it learn how to guide the actor.

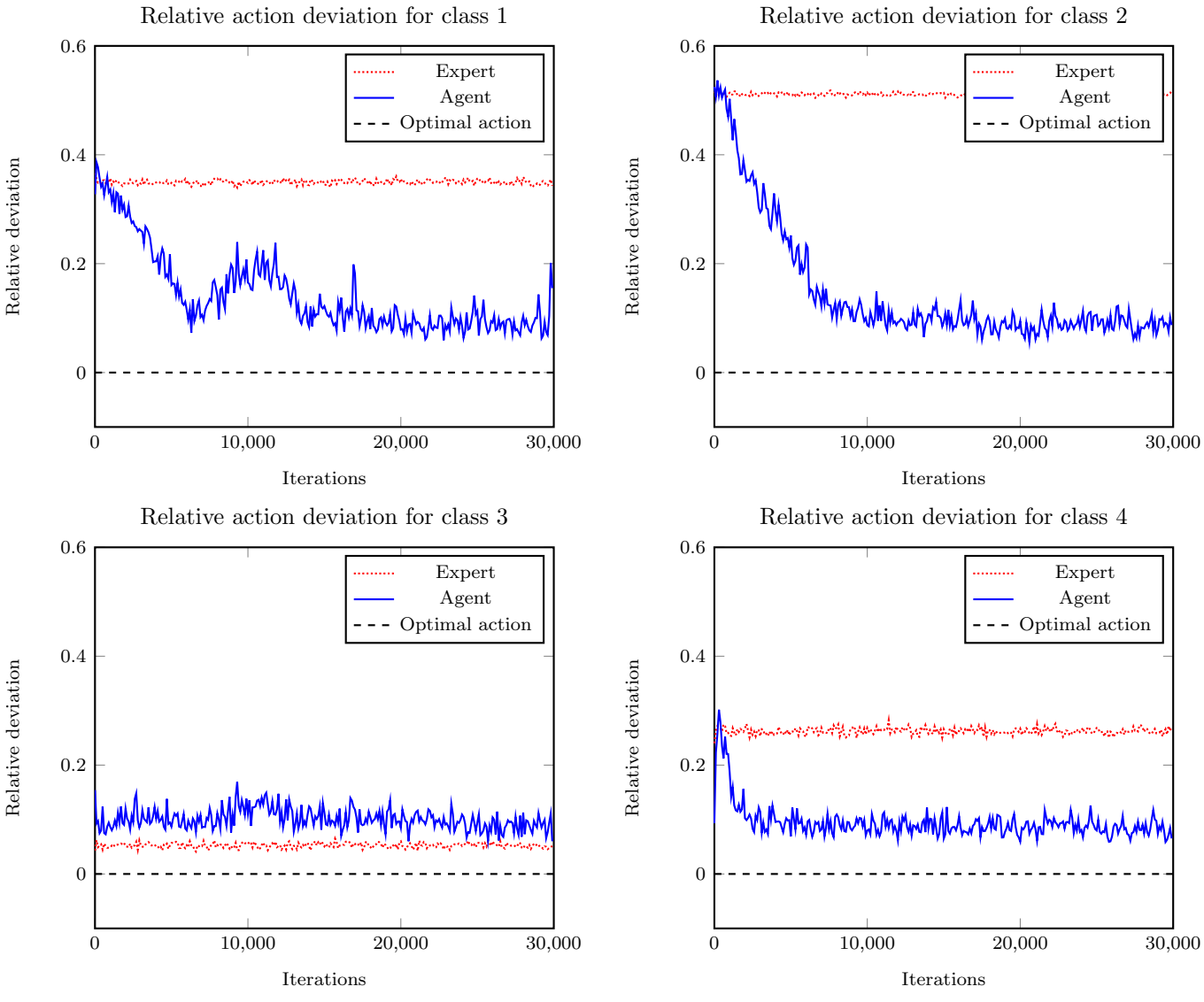

Fig. 2. The deviation of the action for the expert and the agent normalized by the optimal action for the four classes respectively. The plots are smoothed using a moving average.

as the expert in a very short time. For class 3 and 4, we see that the agent learns quicker to take good actions compared to class 1 and 2. This is most likely due to a better expert being of better aid to the agent – both in terms of guiding the action, but also in terms of generating more relevant data, before the agent comes into action. However, these comparisons between the agent and the expert are not entirely fair. The agent's actions include noise to keep the agent exploring, which would eventually be turned off in an actual production setting, when a satisfactory performance has been reached.

In the case of class 3, the agent does not outperform the expert (at least as long the agent keeps exploring), since the expert is near-optimal for this class. This obviously implies, the worse the expert is, the more it can be improved by adding an reinforcement learning agent.

5. Conclusion and future work

This paper has presented a first step towards a reinforcement learning agent for improving the results of a real system for online geometry assurance. The performances of different variations of the agent architecture have been compared and the importance of using an expert to aid the agent on a benchmark example has been shown. In conclusion, it is possible to close the reality gap between an expert system and the reality through reinforcement learning. Additionally, the overall performance is improved, if expert and agent are utilized jointly.

In upcoming work, the proposed agent architecture will be applied to complex geometry assurance simulations, which will require the agent to handle high dimensional input data. The sample efficiency also needs to be further improved, as well as finding better ways of choosing actions, e.g. to maximise cumulative reward in an exploration/exploitation trade-off.

Acknowledgements

The work was carried out in collaboration within Wingquist Laboratory and the Area of Advance Production at Chalmers within the project Smart Assembly 4.0, financed by The Swedish Foundation for Strategic Research. The support is gratefully acknowledged.

References

[1] Söderberg, R., Lindkvist, L., Wärmefjord, K., Carlson, J.S.. Virtual geometry assurance process and toolbox. Procedia CIRP 2016;43(Supplement C):3 – 12. doi:10.1016/j.procir.2016.02.043; 14th CIRP CAT 2016 - CIRP Conference on Computer Aided Tolerancing.

[2] Söderberg, R., Wärmefjord, K., Carlson, J.S., Lindkvist, L.. Toward a digital twin for real-time geometry assurance in individualized production. CIRP Annals 2017;66(1):137 – 140. doi:10.1016/j.cirp.2017.04.038.

[3] Tesauro, G.. Temporal difference learning and td-gammon. Communications of the ACM 1995;38(3):58–68.

[4] Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., et al. Mastering the game of go with deep neural networks and tree search. Nature 2016;529(7587):484–489.

[5] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., et al. Playing atari with deep reinforcement learning. arXiv preprint arXiv:13125602 2013;.

[6] Aydin, M., Öztemel, E.. Dynamic job-shop scheduling using reinforcement learning agents. Robotics and Autonomous Systems 2000;33(2-3):169–178. doi:10.1016/S0921-8890(00)00087-7.

[7] Wang, Y.C., Usher, J.M.. Learning policies for single machine job dispatching. Robotics and Computer-Integrated Manufacturing 2004;20:553–562. doi:10.1016/J.RCIM.2004.07.003.

[8] Drakaki, M., Tzionas, P.. Manufacturing Scheduling Using Colored Petri Nets and Reinforcement Learning. Applied Sciences 2017;7. URL: http://www.mdpi.com/2076-3417/7/2/136. doi:10.3390/app7020136.

[9] Wang, X., Wang, H., Qi, C.. Multi-agent reinforcement learning based maintenance policy for a resource constrained flow line system. Journal of Intelligent Manufacturing 2016;27(2):325–333. URL: http://link.springer.com/10.1007/s10845-013-0864-5. doi:10.1007/s10845-013-0864-5.

[10] Doltsinis, S., Ferreira, P., Lohse, N.. An MDP Model-Based Reinforcement Learning Approach for Production Station Ramp-Up Optimization: Q-Learning Analysis. IEEE Transactions on Systems, Man, and Cybernetics: Systems 2014;44(9):1125–1138. URL: http://ieeexplore.ieee.org/document/6702489/. doi:10.1109/TSMC.2013.2294155.

[11] Sutton, R.S., Barto, A.G.. Introduction to Reinforcement Learning. 1st ed.; Cambridge, MA, USA: MIT Press; 1998. ISBN 0262193981.

[12] Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., et al. Continuous control with deep reinforcement learning. CoRR 2015;abs/1509.02971. URL: http://arxiv.org/abs/1509.02971.

[13] Bojarski, M., Testa, D.D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., et al. End to end learning for self-driving cars. CoRR 2016;abs/1604.07316.

[14] Pomerleau, D.A.. Alvinn: An autonomous land vehicle in a neural network. In: Advances in neural information processing systems. 1989, p. 305–313.

[15] Ross, S., Bagnell, J.A.. Reinforcement and imitation learning via interactive no-regret learning. CoRR 2014;abs/1406.5979.

[16] Ross, S., Gordon, G.J., Bagnell, D.. A reduction of imitation learning and structured prediction to no-regret online learning. In: International Conference on Artificial Intelligence and Statistics. 2011, p. 627–635.

[17] Ng, A.Y., Russell, S.. Algorithms for inverse reinforcement learning. In: in Proc. 17th International Conf. on Machine Learning. Citeseer; 2000, p. 663–670.

[18] Abbeel, P., Ng, A.Y.. Apprenticeship learning via inverse reinforcement learning. In: Proceedings of the Twenty-first International Conference on Machine Learning. ICML '04; New York, NY, USA: ACM. ISBN 1-58113-838-5; 2004, p. 1–. doi:10.1145/1015330.1015430.

[19] Abbeel, P., Coates, A., Ng, A.Y.. Autonomous helicopter aerobatics through apprenticeship learning. The International Journal of Robotics Research 2010;29(13):1608–1639.

[20] Chang, K.W., Krishnamurthy, A., Agarwal, A., Daume, H., Langford, J.. Learning to search better than your teacher. In: Proceedings of the 32nd International Conference on Machine Learning (ICML-15). 2015, p. 2058–2066.

[21] Hester, T., Vecerik, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., et al. Deep q-learning from demonstrations. arXiv preprint arXiv:170403732 2017;.

[22] Lazaric, A., Restelli, M., Bonarini, A.. Reinforcement learning in continuous action spaces through sequential monte carlo methods. In: Advances in neural information processing systems. 2008, p. 833–840.

[23] Van Hasselt, H.. Reinforcement learning in continuous state and action spaces. In: Reinforcement Learning. Springer; 2012, p. 207–251.

[24] Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., et al. Rainbow: Combining improvements in deep reinforcement learning. arXiv preprint arXiv:171002298 2017;.

[25] Blundell, C., Uria, B., Pritzel, A., Li, Y., Ruderman, A., Leibo, J.Z., et al. Model-free episodic control. arXiv preprint arXiv:160604460 2016;.

[26] Pritzel, A., Uria, B., Srinivasan, S., Puigdomenech, A., Vinyals, O., Hassabis, D., et al. Neural episodic control. arXiv preprint arXiv:170301988 2017;.

[27] Kingma, D.P., Ba, J.. Adam: A method for stochastic optimization. arXiv preprint arXiv:14126980 2014;.

[28] Agrawal, R.. The continuum-armed bandit problem. SIAM J Control Optim 1995;33(6):1926–1951.

[29] Lu, T., Pál, D., Pál, M.. Contextual multi-armed bandits. In: Proceedings of the Thirteenth international conference on Artificial Intelligence and Statistics. 2010, p. 485–492.