Available online at www.sciencedirect.com

ScienceDirect

Procedia CIRP 63 (2017) 107 – 112

www.elsevier.com/locate/procedia

The 50th CIRP Conference on Manufacturing Systems

# Motion Planning for Industrial Robots using Reinforcement Learning

Richard Meyes[a,*], Hasan Tercan[a], Simon Roggendorf[b], Thomas Thiele[a], Christian Büscher[c], Markus Obdenbusch[b], Christian Brecher[b], Sabina Jeschke[a], Tobias Meisen[a]

[a] *Institute for Information Management in Mechanical Engineering (IMA) of RWTH Aachen University, Dennewartstr. 27, 52064 Aachen, Germany*
[b] *Laboratory for Machine Tools and Production Engineering (WZL) of RWTH Aachen University, Steinbachstr. 19, 52074 Aachen, Germany*
[c] *Saint-Gobain Sekurit Deutschland GmbH & Co. KG, Glasstr. 1, 52134 Herzogenrath, Germany*

* Corresponding authors. Tel.: +49-241-80-91146; fax: +49-241-80-91122. *E-mail address:* richard.meyes@ima-zlw-ifu.rwth-aachen.de

**Abstract**

A major challenge of today's production systems in the context of Industry 4.0 and Cyber-Physical Production Systems is to be flexible and adaptive whilst being robust and economically efficient. Specifically, the implementation of motion planning processes for industrial robots need to be refined concerning their variability of the motion task and the ability to adaptively deal with variations in the environment. In this paper, we propose a reinforcement learning (RL) based, cognition-enhanced six-axis industrial robot for complex motion planning along continuous trajectories as e.g. needed for welding, gluing or cutting processes in production. Our prototype demonstrator is inspired by the classic wire loop game which involves guiding a metal loop along the path of a curved wire from start to finish while avoiding any contact between the wire and the loop. Our work shows that the RL-agent is capable of learning how to control the robot to successfully play the wire loop game without the need of modeling the wire or programming the robot motion beforehand. Furthermore, the extension of the system by a visual sensor (a camera) allows the agent to sufficiently generalize the learning problem so that it can solve new or reshaped wires without the need of additional learning. We conclude that the applicability of RL for industrial robots and production systems in general provides vast and unexplored potential for processes that feature variability to some extent and thus require a general and robust approach for process automation.


Peer-review under responsibility of the scientific committee of The 50th CIRP Conference on Manufacturing Systems

*Keywords:* Reinforcement Learning; Cyber-Physical Production Systems (CPPS); Self-Optimization

## 1. Introduction

In the era of the fourth industrial revolution, frequently noted as Industry 4.0, one of the four key requirements of Cyber-Physical Production Systems is the ability to react adaptively to dynamic circumstances of production processes [1,2,3].

Motions of industrial robots that are part of a bigger production process are commonly programmed in a non-flexible way and require exact control over the circumstances of the motion task. For instance, the motion of a simple pick-and-place process requires exact knowledge about the position of the object to be picked up and about the container in which the object is to be placed in. Small deviations of either the object or the container would result in process failure, as the non-flexible programming of the motion is not able to deal with small variations of the environment. This general problem gives rise to the question how robots can be enabled to deal with such variations of the environment and autonomously adapt to them to plan their motion in a flexible way.

In this paper, we address this question and present a proof of concept for an augmentation of an industrial robot with cognitive capabilities. This concept is based on the idea to provide a robot with sensor technology that allows it to observe its environment and to complement it further by an operating agent that is able to control the robot and gather experiences about its interaction with the environment. Based on those experiences, the agent seeks to adapt its behavior and control the robot in such a way that the motion task is performed as intended.

We implemented this concept in an exemplary use case scenario in which a six-axis industrial robot (UR5 Robot from Universal Robots) is controlled by an agent that learned to play the wire loop game [4]. The robot autonomously guides a metal


Peer-review under responsibility of the scientific committee of The 50th CIRP Conference on Manufacturing Systems
doi:10.1016/j.procir.2017.03.095

loop along the path of a curved wire from start to finish while avoiding any contact between the wire and the loop. It is enhanced by a visual sensor (a camera) that provides vision of the agent's environment. The agent's algorithm learns to play the game which is modelled as a Markov Decision Process (MDP) by means of reinforcement learning (RL) [5,6] and Q-learning [7] without any domain knowledge, i.e. it does not know the concept of a loop or a wire and it does not know what its actions do exactly.

### *1.1. State of the Art*

The successful application of RL and a variation of Q-learning, was previously demonstrated by enabling an agent to play board games, e.g. backgammon [8] or even Atari games directly from visual sensory input [9,10]. MDPs and partially observable MDPs (POMDPS) have been largely used in for motion planning in mobile robotics [11,12], autonomous planning for unmanned ground and aerial vehicles [13,14] and human assisted teleportation [15].

Recent research has attempted to utilize deep RL to tackle a wide variety of continuous motor control problems, e.g. motion planning for industrial robots directly from sensory input [16,17]. Although these attempts demonstrated the conceptual usability of the methods they rely on heavy computational effort both in terms of the number of used cores (either CPUs or GPUs) and the required computation time which is of the order of several days.

With our approach, we avoid directly processing sensory input to decrease the computational effort of the learning problem significantly. We utilize a well-approved approach from the mobile robotics domain that relies on the formalization of the planning problem as an MDP and transfer this approach to the stationary robotics domain. We show that our agent is able to learn how to play the wire loop game within only a few minutes of computation time on a single CPU. Furthermore, it is capable of generalizing the learning problem within a few hours so that it is able to play the game with wire configurations that were not used during the training phase. Our results imply that RL-based augmentations for robots provide a feasible way to deal with processes that feature variability to some extent and thus, in order to be automated, require a general and robust approach.

## 2. Background

The presented concept for an augmentation of robots is based on RL and the condition to model the learning task at hand as a Markov Decision Process allowing to utilize a variation of Q-learning, a learning paradigm that has been successfully applied to various virtual learning scenarios.

### *2.1. Reinforcement Learning and Markov Decision Processes*

Reinforcement learning (RL) is a machine-learning paradigm inspired by behaviorist psychology and addresses the procedure of how an agent (an animal, a human or even a machine) interacts with its environment [5]. It describes the problem of an agent that tries to develop a behavioral strategy in order to maximize some notion of cumulative reward as a result of taking the right actions in any state of its environment [6]. Figure 1, left hand side illustrates the underlying state-action-reward principle of RL problems. In general, these kinds of problems can be formalized as MDPs.

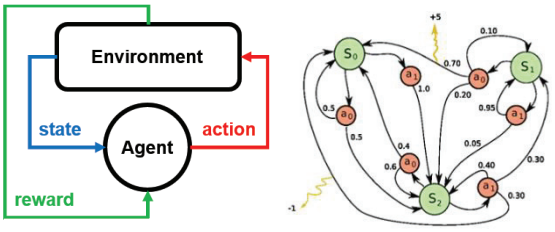


**Figure 1: (left)** Schematic illustration of the RL paradigm. An agent interacts with its environment and receives a reward for each action that is taken in a specific state of the environment. **(right)** Schematic illustration of a simple, discrete MDP with three states $s_0$, $s_1$, $s_2$ and two actions $a_0$, $a_1$. The probability of reaching a state $s_i$ by taking an action $a_i$ is given by the black number next to the black transition arrows. The reward that is given after taking certain actions in certain states is represented by yellow arrows. Figure adopted from [18].

An MDP is a mathematical framework for modeling decision making as a discrete time stochastic control process [19]. It assumes that the modelled stochastic process possesses the Markov property, i.e. the conditional probability distribution of each future state depends only on the present state and not on the sequence of events that preceded it [20]. Figure 1, right hand side illustrates a simple example of an MDP with three states and two actions. In order for an agent to maximize its reward in the exemplary MDP in Figure 1, right hand side, the agent needs to learn that the cumulative reward over time can only be maximized when temporary punishments, i.e. negative rewards, are accepted. Thus, in general, an agent needs to take into account not only immediate rewards but also possible future rewards. A single episode $e_{MDP}$ of any given MDP forms a finite sequence of states, actions and rewards and can be expressed as:

$$e_{MDP} = \{s_0, a_0, r_0, s_1, a_1, r_1, s_2, \dots, s_{n-1}, a_{n-1}, r_{n-1}, s_n\} \quad (1)$$

where $s_i$, $a_i$, $r_i$ represent the i-th state, action and reward that is received after performing the action, respectively. The total future reward from any time point $t$ is given by:

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{n-t} r_n \quad (2)$$

where $\gamma \epsilon (0, 1)$ is the discount factor and models how strongly the agent takes future rewards into account. Values close to 0 will represent a short-sighted strategy as higher-order terms for rewards in the distant future become negligible. If the environment is deterministic, $\gamma$ can be set to 1 as the same actions always result in the same rewards. A good strategy for an agent trying to maximize its discounted future reward can be learned by means of the Q-learning paradigm.

### *2.2. Q-Learning*

Q-learning is a paradigm that can be used to allow an agent to find an optimal policy for choosing an action for any given finite MDP. In general, a policy is a deliberate system of principles to guide decisions or more specifically, a decision function that specifies what the agent will do for each possible value that it can sense [21]. In contrast to other learning paradigms such as SARSA (state-action-reward-state-action), Q-learning is an off-policy learner and allows learning from

different policies that can be changed during training [22]. Q-learning is based on the existence of the Q-function

$$Q(s_t, a_t) = max\{R_{t+1}\} \quad (3)$$

that models the quality of an action $a$ which is performed in state $s$ at time step $t$. It represents the maximum discounted future reward assuming that the optimal policy

$$\pi(s) = argmax_a\{Q(s,a)\} \quad (4)$$

is followed henceforward. As the optimal policy is not known, it has to be estimated by iteratively approximating the Q-function. Inserting equation (2) into (3), the Q-function can be written as:

$$Q(s,a) = r + \gamma \cdot max_{a'}\{Q(s',a')\} \quad (5)$$

which is called the Bellman equation [23]. It shows that the Q-value for a specific state-action pair $(s, a)$ can be expressed with respect only to the future state $s'$ allowing for an iterative estimation of the Q-value based on future observations. The update of the Q-value for each state-action pair is done by considering the Q-value for the future state $s'$ that is reached after taking action $a$:

$$Q(s,a) = Q(s,a) + \alpha\big(r + \gamma \cdot max_{a'}\{Q(s',a')\} - Q(s,a)\big) \quad (6)$$

where $\alpha$ is the algorithm's learning rate which determines how strongly the difference between the previous Q-value and the newly proposed Q-value is taken into account. Note that for $\alpha = 1$ equation (6) simplifies to the Bellman equation (5). It has been shown, that this iterative approach of estimating the Q-function converges and, given enough iterations, represents the true Q-value [24].

## 3. Use Case: The Wire Loop Game

In our use-case scenario, we enabled a six-axis industrial robot to play the wire loop game [4]. The wire loop game presents a simple to understand, yet appropriately complex use-case scenario which provides a sufficient amount of variability of the setup posing a challenging task for an agent to master.

### *3.1. Experimental Setup*

The experimental setup consists of a Universal Robots' UR5 [25] that is fixed on a wooden board and placed on a table. Within the reach of the robot, a metal wire of about 150 cm length and about 1 cm thickness is clamped between two fixed-points so that the starting-point and the end-point are predefined. The wire can be curved arbitrarily between those two points by hand. An industrial camera is placed above the setup providing vision of the whole wire. Figure 2 shows the setup and its key components. For reasons of simplicity, the game is reduced to two dimensions, i.e. the wire is placed in a plane and the robot's tool center point (TCP) only needs to move in two dimensions. This reduction greatly simplifies the automatic generation of the environment model by means of image processing algorithms (c.f. section 3.2), however, it does not impair the implications of our results considering that the algorithm can be easily extended to three dimensional problems. It has to be noted though, that the extension to the third dimension comes at the cost of higher computational effort. However, for many possible applications the planning of a two dimensional trajectory is sufficient, e.g. for a laser cutting process.

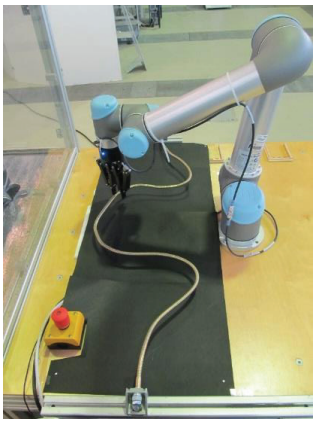

**Figure 2:** Experimental Setup with its key components. A golden metal wire is placed in front of a black background so that the camera (out of picture) can easily distinguish the wire from its environment.

The robot is controlled via representational state transfer (REST) web services, which provide basic robot control functions as well as some specific functions for the wire loop game. The basic functions allow movements of the robot's TCP and read-out of the current position of the TCP as well as the orientation of the loop. The REST API is integrated in the learning algorithm so that the agent can control the robot via the REST interface directly. In order to ensure conformity between the real setup and the virtual environment, an initialization function moves the TCP to the starting-point of the wire. Thenceforth, the robot's TCP can be moved in discrete steps of 5 mm in four distinct directions or it can be rotated 45 degrees clockwise or counterclockwise. In addition to the real robot a detailed robot simulation environment (Siemens Robot Expert) was used to verify the agent's learning progress. The simulation can be controlled with the same REST interface as the real robot.

### *3.2. The Wire Loop Game as an MDP*

Once an arbitrarily curved wire is installed into the setup, the camera takes a static picture of the wire from which a simple environment model for the learning agent is created in four steps with an image-processing algorithm from the open source computer vision library (OpenCV) [26]. The calibrated camera takes a grayscale image of the installed wire from the bird's-eye view (c.f. Figure 3, left). Distortions of the image are removed by the algorithm with the help of four distinct markers at predefined positions in the image, which span a rectangle and ensure that the wire is placed in a plane. From the rectified grayscale image, a thin model of the wire is created after dilation and subsequent thinning of the imaged wire and serves as the foundation of the environment for the learning phase of the agent.

The virtual model assumes that the TCP is positioned at the starting-point of the wire and initializes the environment in a

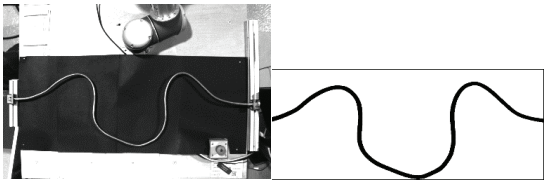

**Figure 3:** Generation of the environment model **(right)** from a raw greyscale image **(left).** The industrial camera takes a raw greyscale image from the bird's eye view that is 1) rectified and a corrected 2) dilated and 3) thinned to create a high-resolution path with a thickness of one pixel as the environment model.

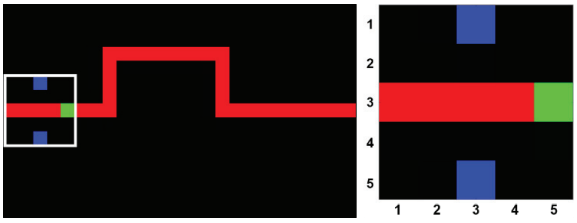


**Figure 4: (left)** Simplified visualization of the environment model. The model of the wire is autonomously created from the camera image. **(right)** Environment state represented by a 5x5 cut-out of the environment.

fixed initial state. Figure 4, left hand side shows a visualization of the environment model and an exemplary state derived from that model on the right hand side. The two blue pixels represent the top and bottom parts of the loop with the TCP being in the middle of it and placed on the wire. The green pixel represents a goal that has to be reached with the TCP. A state is represented by a three-dimensional matrix of size 5x5x4 and contains information of a small cut-out of the environment, namely the positioning of the TCP and the loop in relation to the wire and the goal. The first two dimensions are coordinates and the last dimension describes what is placed at the specific location that can be either a part of the wire, a part of the loop, a goal that has to be reached, or nothing at all.

From the initial state, the agent starts to explore its environment by performing actions, triggering state transitions in the environment and gathering corresponding rewards. The agent is able to choose one out of six possible actions in every state. The available actions correspond to fixed steps of 5 mm of the TCP which can be either moved up, down, left or right with respect to its current position or rotated clockwise or counterclockwise. The agent is rewarded for reaching the goal which is replaced once it has been reached to ensure the agent's progress along the wire. Any contact between the loop and the wire as well as any action that would move the loop outside of the environment is severely punished. Figure 5 illustrates a few consecutive steps of how the agent moves in the environment and how its perceived state changes accordingly. From the initial state the agent moves to the right four times before it has to rotate counterclockwise and move right once again in order to pass the corner. Once a goal is reached, it is automatically replaced at the furthest point of the wire that can be perceived by the agent. Note that the agent replaces the goal automatically based purely on the information it perceives in the environment states and that no fixed path along the wire is implicitly given by predefined placements of the goals.

### *3.3. Learning Phase of the Agent*

As the agent explores its environment it stores experiences as state-action-reward triples $(s, a, r)$ and iteratively estimates the Q-values for each state-action pair it encounters. The agent starts with no experience about the environment at all and will perform random actions every time it finds itself in an unknown state initializing the corresponding Q-Values at the same time. As experiences are gathered over time, the agent starts to update the Q-values according to equation (6) and learns how to progress along the wire while avoiding any contact between the loop and the wire.

A general problem of such scenarios is the exploration-exploitation dilemma [27]. It describes the issue of how an agent's learned strategy tends towards a local optimum. As a result of the arbitrary (thus, in most cases random) initialization of the Q-values, the agent starts to take random actions in every

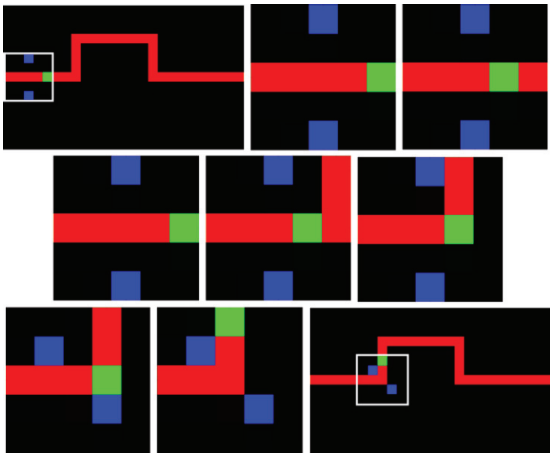

**Figure 5:** Illustration of how the agent moves the loop along the wire path. Reading the images from left to right and from top to bottom, the first and the last image illustrate the environment while the images in between illustrate how the agent perceives the environment states as it takes specific actions to progress along the wire.

state, following the rule of choosing the action with the highest Q-value, performing crude exploration of the environment. As the Q-function converges, the amount of exploration decreases and the agent settles for a greedy exploitation strategy without attempting further exploration.

This problem can be addressed by extending the Q-learning algorithm by the so-called $\epsilon$–greedy exploration. It enforces that the agent chooses a random action with probability $\epsilon$ or the greedy action with probability $1 - \epsilon$. Our agent starts with an initial value for $\epsilon$ of $\epsilon_{init} = 0.9$ which is decreased until it reaches the lower limit of $\epsilon_{lim} = 0.1$. This way, a minimum amount of exploration is guaranteed regardless of how good the agent's learned strategy has become. Algorithm 1 shows the pseudo-code illustrating how the original Q-learning algorithm is extended by the $\epsilon$–greedy exploration.

**Algorithm 1:** Q-learning with $\epsilon$–greedy exploration:

initialize Q-values arbitrarily
observe initial state $s$
set initial value for $\epsilon$, e.g. $\epsilon_{init} = 0.9$
**repeat**
  select action $a$
    with probability $\epsilon$ select a random action $a$
    otherwise select $a = argmax_{a'}\{Q(s, a')\}$
  perform action $a$
  observe reward $r$ and new state $s'$
  update Q-value according to equation (6)
  update state: $s = s'$
**until** terminated
**if** $\epsilon > \epsilon_{lim} = 0.1$, **then**
  decrease $\epsilon$, e.g. $\epsilon = \epsilon - (1/number\ of\ episodes)$
**end if**

The agent continues to gather experiences until certain conditions for termination are met. An episode is terminated if the loop is moved outside of the boundaries of the environment model as this would correspond to a movement of the robot in the real world into some direction that lies out of scope of the agent and can be potentially harmful. Furthermore, an episode is terminated if the agent takes an action which results in an overlap of any part of the wire with any part of the loop. At last, an episode is terminated if the agent successfully reached the

end of the wire and completed the game. After an episode is terminated, a new episode is started in the same initial state as the previous episode.

However, this static initialization is not always feasible from a computational point of view. Considering some stage during the learning phase in which the agent has learned to successfully navigate along the first half of the wire, after a specific number of episodes the termination of the current episode upon making a mistake, e.g. touching the wire, would initialize the next episode at the starting-point of the wire, even though the agent has already learned a great deal of the wire and will not learn anything new by completing the first half over and over again.

In order to overcome this issue, the learning algorithm can be separated into two phases which are repeated iteratively. In the first phase, the training phase, of the first cycle, the agent's initial environment state $s_1^{init}$ is initialized at the starting-point of the wire and the agent starts to explore its environment, gathering experiences.

In the second phase, the testing phase, which starts after a fixed number of training episodes, the agent's learning progress is tested and the last stable environment state $s_1^{stable}$ (the last state in which the agent reached a goal) is saved. In the next cycle, the training phase is initialized at the last stable state, i.e. $s_2^{init} = s_1^{stable}$. This iterative cycle of training and testing results in two parameters that influence the learning of the agent in different ways. The number of episodes $N_e$ within the training phase determines how thoroughly the agent explores its environment locally, as every episode starts in the same initial state. Large values of $N_e$ guarantee that the agent keeps exploring the environment (the extent of which is depending on the parameter $\varepsilon$) even though it may already have gathered all the experiences that can be gathered. The number of cycles $N_c$ determines how many times the agent's learning progress is tested. Larger values for $N_c$ can compensate for smaller values of $N_e$ as it forces the agent to continue its training at later stages along the wire path abandoning the possibility to improve on what it has learned so far. This way, a tradeoff between local thoroughness and overall duration of the learning phase can be made depending on the complexity of the wire. Algorithm 2 shows the pseudo-code illustrating the implemented learning algorithm of the agent using a variation of Q-learning with $\epsilon$–greedy exploration. A dashed line separates the training and testing phases.

**Algorithm 2:** Implemented learning algorithm including a modified version of Q-learning with $\epsilon$–greedy exploration:

**for** cycle ranging from 1 to $N_c$ **do**
  initialize environment state $s_{cycle}^{init}$
  **if** first cycle, **then**
    $s_{cycle}^{init}$ is initialized at the starting-point of the wire
    **else** $s_{cycle}^{init} = s_{cycle-1}^{stable}$
  **end if**
  set initial value for $\epsilon$, e.g. $\epsilon_{init} = 0.9$
  **for** episode ranging from 1 to $N_e$ **do**
    **repeat**
      observe current state $s$
      **if** $s$ is an unknown state, **then**
        initialize $Q(s, a)$
      **end if**
      select action $a$
        with probability $\epsilon$ select a random action $a$
        otherwise select $a = argmax_{a'}\{Q(s, a')\}$
      perform action $a$ and observe reward $r$ and new state $s'$
      **if** $s'$ is an unknown state, **then**
        initialize $Q(s', a')$
      **end if**
      update Q-value according to equation (6)
      update state: $s = s'$
    **until** terminated
    **if** $\epsilon > \epsilon_{lim} = 0.1$, **then**
      decrease $\epsilon$, e.g. $\epsilon = \epsilon - 1/N_e$
    **end if**
  **end for**
  initialize environment state $s_{cycle}^{init}$
  **repeat**
    select and perform an action $a = argmax_{a'}\{Q(s, a')\}$
    observe reward $r$ and new state $s'$
    **if** $s'$ is a stable state, **then**
      $s_{cycle}^{stable} = s'$
    **end if**
    update Q-value according to equation (6)
    update state: $s = s'$
  **until** terminated
**end for**

Note that the Q-values are initialized on the fly during the training phase rather than prior to the training phase. This has the advantage that no prior knowledge about the size of the state space is needed in order to initialize all the Q-values for all possible environment states and that no computational effort is wasted on states that the agent never explores during his training.

## 4. Summary and Discussion

In this paper, we presented a concept for a cognitive augmentation of industrial robots based on RL and Q-learning that enables a control agent to autonomously strive for an optimal strategy to solve its task. We applied the presented methods to a use case scenario in which an agent learns to control a six-axis industrial robot to play the wire loop game. The agent is able to learn the correct trajectory for each individual wire within a few minutes and saves the gathered experiences for future references. The experiences are stored as relations between states and actions with corresponding rewards in a non-relational database that represents the agent's knowledge base.

The agent's ability to generalize the problem and correctly handle new wires that were not seen before is based on the fact that its perception is reduced to a small cut-out of the wire which enables the agent to combine gathered experiences from previous wires in the right order to solve an unknown wire. However, the simplified representation of the environment states comes at the cost of a certain coarseness of the robot's movement as the minimum distance that is covered by a single step as well as the rotation angle is fixed. As a consequence, certain features of some curved wires cannot be solved, e.g. acute angles. Furthermore, it requires the wire and the loop to be thinner than or just as thick as the minimum step distance to ensure conformity between the environment model and the real environment. We evaluated the trained agent in the virtual environment as well as in the real world setup. The transfer of the learned sequenced of actions to the real world setup translates the action sequence into real world coordinates so

that the real world TCP is controlled without the need to consider robot speed and acceleration. Rather, the maximum allowed joint speed and acceleration are fixed values and the learned actions are performed with that fixed values.

Although modelling the interaction of the control agent with the environment as an MDP represents a drastic simplification of the real world, it avoids the challenging task to learn a control policy for the robot directly from sensory input, i.e. visual information of the environment recorded by a camera and motor-sensory information of the robot's joint positions and torques. Attempts to tackle this task are based on the use of artificial neural networks as an estimator for the Q-function [16] and bring a completely new set of problems that need to be solved for a successful implementation on the one hand, but also offer prospects to create a general understanding of the task on the other hand.

Although our approach limits the training of the agent to the virtual environment it can be extended to the real world, enabling the agent to learn wires that cannot be correctly represented in the simplified environment model. This requires to extend the visual sensory input from an initial static image to real-time input during the game. In addition, the experimental setup needs to be able to provide the agent with some sort of feedback that assesses the agent's interaction with the environment, e.g. upon contact between the wire and the loop an electrical circuit is closed which results in a negative reward.

In general, RL-based augmentations for robots offer a wide range of possibilities to deal with variabilities of production processes to some extent. As long as the task at hand is essentially the same, particular variations of that task can be solved by a single robot. The presented approach can be transferred to processes that involve motion planning of a single tool similar to the wire loop game, e.g. welding, gluing or laser cutting processes. The time and the cost of ramp-up processes of new production processes and product lines can be decreased significantly as the production task can be autonomously adapted from the previous lines and products and does not necessarily need to be programmed from scratch. Considering the average lifetime of robots which is of the order of several years or even decades, the mere accumulation of experiences gathered automatically by those robots offers great potential to consolidate process and domain specific knowledge in a non-volatile way. It is the first step to allow robots to increase their value based on the accumulation of expertise over time similar to human labor.

## Acknowledgements

The authors would like to thank the German Research Foundation DFG for the kind support within the Cluster of Excellence "Integrative Production Technology for High-Wage Countries".

## References

[1] N.N. Securing the future of German manufacturing industry: Recommendations for implementing the strategic initiative INDUSTRIE 4.0. Final report of the Industrie 4.0 Working Group, acatech, April 2013: 78.
[2] Monostori L. (2014) Cyper-physical production systems: Roots, expectations and R&D challenges, Procedia CIRP 17: 9-13.
[3] Simon P. (March 18, 2013) Too Big to Ignore: The Business Case for Big Data. Wiley. p. 89.
[4] Wikipedia (22/11/2016) Wire loop game, https://en.wikipedia.org/wiki/Wire_loop_game (last checked at 29.11.2016)
[5] Sutton R.S. and Barto A.G. (1998) Reinforcement learning: An introduction (Vol. 1, No. 1), Cambridge: MIT Press.
[6] Sutton R.S. (1992) Introduction: The challenge of reinforcement learning. In Reinforcement Learning (pp. 1-3). Springer US.
[7] Watkins C.J. and Dayan P. (1992) Q-learning, *Machine learning*, *8*(3-4): 279-292.
[8] Tesauro G. (1995) Temporal difference learning and td-gammon, Communications of the ACM 38(3): 58-68
[9] Mnih V. et al. (2013) Playing Atari with Deep Reinforcement Learning, arXiv preprint arXiv:1312.5602v1.
[10] Mnih V., Kavukcuoglu K., Silver D., et al. (2015) Human-level control through deep reinforcement learning, Nature 518: 529-533.
[11] Burlet, J., Aycard, O. and Fraichard, T. (2004), Robust motion planning using markov decision processes and quadtree decomposition, Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International *Conference on* (Vol. 3, pp. 2820-2825). IEEE.
[12] Lacerda, B., Parker, D. and Hawes, N. (2014), Optimal and dynamic planning for Markov decision processes with co-safe LTL specifications, In Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on (pp. 1511-1516). IEEE.
[13] Katrakazas, C., Quddus, M., Chen, W.H. and Deka, L. (2015), Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions, Transportation Research Part C: Emerging Technologies, 60, pp.416-442.
[14] Bertuccelli, L.F., Wu, A. and How, J.P. (2012), Robust adaptive Markov decision processes: Planning with model uncertainty. IEEE Control Systems, 32(5), pp.96-109.
[15] Côté, N., Bouzid, M. and Mouaddib, A.I. (2013), Integrating human recommendations in the decision process of autonomous agents. In Web Intelligence (WI) and Intelligent Agent Technologies (IAT), 2013 IEEE/WIC/ACM International Joint Conferences on (Vol. 2, pp. 195-200). IEEE.
[16] Levine S., Finn C. et al. (2016) End-to-End Training of Deep Visuomotor Policies, Journal of Machine Learning Research 17(39): 1-40
[17] Mnih V. et al. (2016) Asynchronous methods for deep reinforcement learning, arXiv preprint arXiv:1602.01783.
[18] Matiisen T. (2015) Demystifying Deep Reinforcement Learning, https://www.nervanasys.com/demystifying-deep-reinforcement-learning (last checked at 24.11.2016).
[19] Tijms H.C. (2003). A First Course in Stochastic Models. Wiley.
[20] Feller W. (1971) Introduction to Probability Theory and Its Applications, (Vol. 2, No. 2, pp. 9-20), Wiley.
[21] Poole D.L. and Mackworth A.K. (2010) Artificial Intelligence: foundations of computational agents, Cambridge University Press.
[22] DeWolf T. (2013), Reinforcement Learning Part 2: SARSA vs Q-Learning, https://studywolf.wordpress.com/2013/07/01/reinforcement-learning-sarsa-vs-q-learning/ (last checked at 22.12.2016)
[23] Bellman R. (1952) On the theory of dynamic programming. Proceedings of the national Academy of Sciences, 38(8): 716-719.
[24] Melo F.S., Convergence of Q-learning: A simple proof, http://users.isr.ist.utl.pt/~mtjspaan/readingGroup/ProofQlearning.pdf (last checked at 28.11.2016).
[25] Universal Robots (09/2016) UR5 Technical Specifications, https://www.universal-robots.com/media/50588/ur5_en.pdf (last checked at 29.11.2016)
[26] Itseez (2016) OpenCV, http://www.opencv.org (last checked at 05.12.2016).
[27] Rejeb L., Guessoum Z. and M'Hallah R. (March 2005), The exploration-exploitation dilemma for adaptive agents, In Proceedings of the Fifth European Workshop on Adaptive Agents and Multi-Agent System