Reduction of Complexity in Q-Learning a Robot Control for an Assembly Cell by using Multiple Agents

Georg Kunert[1*], Thorsten Pawletta[1], Sven Hartmann[2]

[1]Research Group Computational Engineering and Automation, Wismar University of Applied Sciences: Technology Business and Design, Philipp-Müller-Straße 14, D-23966 Wismar, Germany; *georg.kunert@cea-wismar.de

[2] Department of Informatics, Clausthal University of Technology, Julius-Albert-Straße 4, D-38678 Clausthal-Zellerfeld

SNE 30(3), 2020, 117-124, DOI: 10.11128/sne.30.tn.10524
Received: August 8, 2020 (Selected ASIM vSST 2020 Postconf. Publ.); Revised: August 15, 2020; Accepted: August 20, 2020
SNE - Simulation Notes Europe, ARGESIM Publisher Vienna, ISSN Print 2305-9974, Online 2306-0271, www.sne-journal.org

Abstract. Production systems in Industry 4.0 are characterized by a high degree of system networking and adaptability. They are often characterized by jointed-arm robots, which have a high degree of adaptation. Networking and adaptivity increase the flexibility of a system, but also the complexity of the control, which requires the use of new development methods. In this context, the Simulation-Based Control approach, a model-based design method, and the concept of Reinforcement Learning (RL) are introduced and it is shown how a task-based robot control can be learned and executed. Afterwards, the time complexity of the Q-learning method will be examined using the application example of a robot-based assembly cell with two differently flexible system configurations. It is shown that, depending on the system configuration, the time complexity of learning can be significantly reduced when using several agents. In the studied case, the complexity decreases from exponential to linear. The modified RL structure is discussed in detail.

Introduction

Production systems of Industry 4.0 have a high degree of networking and adaptivity. The latter characterizes the flexibility of a system to adapt to changing influences [1]. Systems are often characterized by jointed-arm robots which, according to [2], have a high degree of flexibility in terms of design and control. Adaptivity and networking increase the complexity of the control software, which requires the application of new development methods. In recent years, similar methods have been established under various terms, such as *Model-Based Design (MBD)* [3], *Rapid Control Prototyping (RCP)* [4] or *Virtual Commissioning (VC)* [5]. What they have in common is that they are based on a continuous model- and simulation-based development from the design to the operation phase. The *Simulation-Based-Control (SBC)* approach [6] was developed adequately for this purpose by the Computational Engineering and Automation research group at the University of Applied Sciences in Wismar. This approach was adapted in [7] and [8] specifically for task-oriented control development for jointed-arm robots.

In [9], a connection of the SBC approach with machine learning methods based on *Reinforcement Learning (RL)* according to [10] is shown. The RL is defined by a structure of at least one agent, which has a learning method and an environment. The specific learning method is Q-learning. Based on predefined tasks and transformation modules, a control strategy is learned and automatically transformed into an SBC-compliant program. Since Q-learning is a model-free algorithm, it can be applied to various problems. However, impracticable computing times result relatively quickly, even though learning can often be accelerated for problems with limited state space by means of parallel processing and binary trees [11]. A significant reduction of the state space and, thus, the computational effort is achieved with model-based RL approaches [12]. Here the agent already has process-relevant knowledge at the beginning of the learning process, but thereby loses its universali-

ty. For problems with large state space, the computing time can be reduced by combining *artificial neural networks (ANN)* as function approximators [13], [14]. However, this increases the complexity of the software architecture. Furthermore, the design and training of the ANN requires experience and time. In contrast, the simple architecture of the original Q-learning is an advantage if the computational effort can be mastered.

In this paper, we investigate how the computing time for Q-learning of a task-based robot control can be reduced by using several agents. Two differently flexible system structures of an assembly cell are considered and the time complexity of learning with one agent compared to several agents is analyzed. The learning is performed on simulated system environments. The generation of an executable robot control based on the SBC approach and basics of the RL approach using Q-learning are discussed in the following background section.

1 Background

Starting from the adapted SBC framework for articulated arm robot systems according to [7] and [8], this section deals with the principle of integration with a machine learning process and the basics of RL based on Q-learning according to [10] and [13].

1.1 The SBC Approach

As shown in Figure 1, an SBC-based robot control is layered in analogy to the concept of the *Robot Operating System (ROS)* [15].

The *Control Model (CM)* specifies the control strategy by composing predefined basic tasks. The *Process Model (PM)* implements the task transformer based on predefined task-specific modules. Additionally, the PM maps the state information. The *Interface Model (IM)* implements the interface to the hardware using a robot middleware. With the *RCV Toolbox for MATLAB* according to [16] as middleware, SBC-based robot controls can be developed and operated independently of robot manufacturers and model-based with *MATLAB/Simulink*. Virtual and real robots can additionally interact with a virtual process environment in the form of a simulation model.

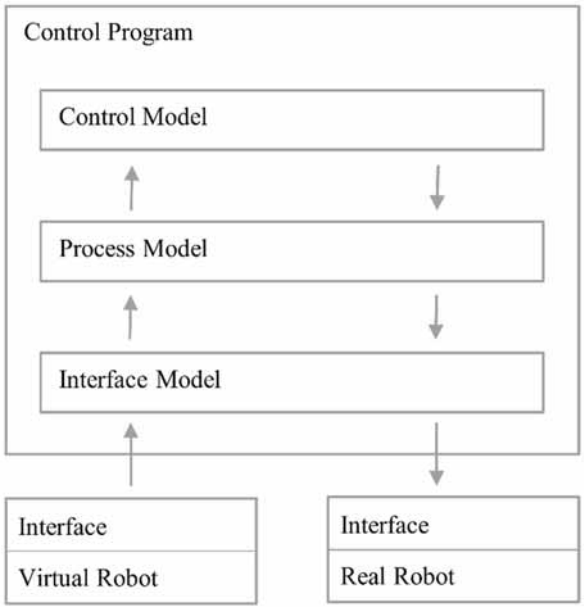

Figure 1: Structure of an SBC-based robot control.

With the integration of the SBC framework with a RL procedure, as introduced in [9], the task-based control specification of the CM according to formula 2 is automatically generated from a sequence of learned state/action tuples according to formula 1:

$$\begin{gathered}\left[(s_0, a_0), \ldots, (s_t, a_k), \ldots, (s_{target}, cancel)\right] \\ with\ s_t \epsilon S,\ a_k \epsilon A,\ S\ state\ set,\ A\ action\ set\end{gathered} \tag{1}$$

$$[\, move(\ldots), pick(\ldots), \ldots, stop(\ldots)\,] \tag{2}$$

Learning takes place offline using a simulated process environment. In principle, the learning algorithm could also run during the operating phase and adapt the control strategy in certain time windows if the real-time requirements are met.

1.2 Reinforcement Learning with Q-learning method

Besides supervised and unsupervised learning, the RL method forms a third class of machine learning procedures. The goal of RL is to learn a behavioral strategy $\pi: S \rightarrow A$ that assigns an action $a \in A$ to each state $s \in S$. The RL does not require explicit training data. It trains itself using a real or simulated environment according to the *trial and error principle*.

The RL is based on a framework as shown in Figure 2. In model-free RL, the agent only knows the allowed action set A at the start of training. The environment is defined by different states, $s \in S$. When an action $a \in A$ takes effect, the environment determines a subsequent state s' with the state transition model $T: S\ x\ A \rightarrow S$ and computes a reward value $r \in R$ for the current action. The subsequent state s' and the reward r are sent back to the agent. During training, the agent receives information about the possible states of the environment and the benefits of actions through iterative interaction, and gradually learns a behavioral strategy π. After completion of all training episodes, the behavioral strategy π is derived from the Q-matrix.

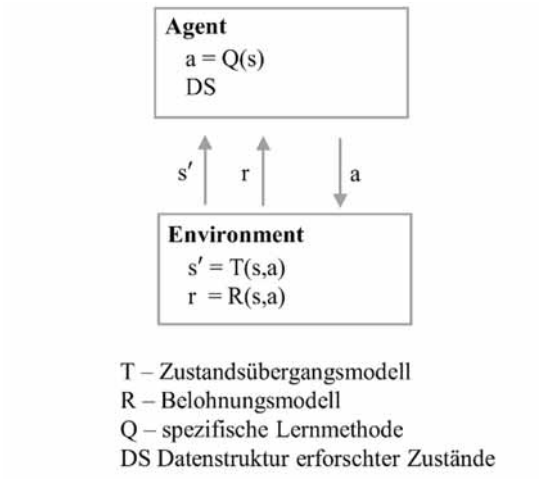

Figure 2: RL framework with Q-learning method.

Learning takes place in phases, also known as episodes. These are independent of each other, always start in an initial state s_0 of the environment and end when a target state s_{target} or abort state s_{abort} is reached. At the beginning of the training, the agent selects an action $a \in A$ purely randomly (*exploration*). As the learning process progresses, the agent increasingly uses the knowledge it has acquired to select an action (*exploitation*). The ratio of exploration to exploitation is adjusted in the course of training.

Q-learning is based on a table function called Q-matrix. A matrix element $Q(s, a)$ represents the benefit Q of an action a when it is performed in the state s of the environment. From the cardinality $|A|$ follows the column dimension of Q. The row dimension of Q grows dynamically during the training with the number of states explored.

The data of explored states are stored in an indexed data structure DS based on the row index of Q. This allows the agent to clearly recognize already explored states. The first episode of the training starts with an empty Q-matrix and an empty data structure. Subsequent episodes build on the already acquired knowledge in Q and DS. After each interaction with the environment, the agent checks whether the received state s' is known. If not, the Q-matrix is extended by a new row and the state data are added to the data structure DS. The successive adjustment of the Q-values results from formula 3 according to [13]:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \cdot \max_{a'} Q(s',a') - Q(s,a) \right] \quad (3)$$

The updated Q-value of the current state/action tuple (s, a) is calculated from the previous Q-value, the currently received reward r, and the maximum Q-value of all possible actions in the currently received subsequent state s'. The influence of the individual variables is determined by the hyperparameters: (i) discount factor γ and (ii) learning rate α. The discount factor controls the influence of rewards expected in the future and the learning rate controls the influence of the current observation. In environments with deterministic behavior, a high learning rate can be applied with up to $\alpha = 1$.

2 Application Example

An automated assembly cell (AC) with an articulated arm robot for the production of different assemblies is considered as an example. By means of RL, product-compliant assembly sequences are to be learned, on the basis of which robot controls can be generated according to Section 1.1. Figure 3 shows the system layout of the AC. The AC consists of the articulated arm robot (R), an assembly station (AS) and a belt conveyor (BC). The BC feeds input parts from upstream production sections and serves as a transfer zone (TZ) to the robot. Depending on the specific system configuration, different numbers of transfer locations (TL) are possible, which influences the system flexibility. Unused input parts and assembled modules are automatically removed. The sequence in which the individual parts enter the AC is unknown and is assumed to be random.

Figure 4 shows an example of an assembled module as an exploded view, whose assembly is examined below for two system variants of the AC:

1. Minimum variant with only one TL as TZ and
2. Variant with three TLs as TZ.

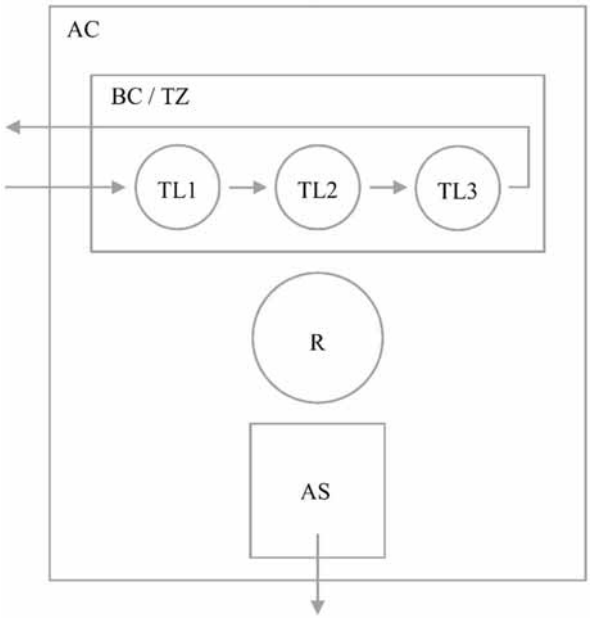

Figure 3: System layout of the assembly cell with maximum three TLs as TZ.

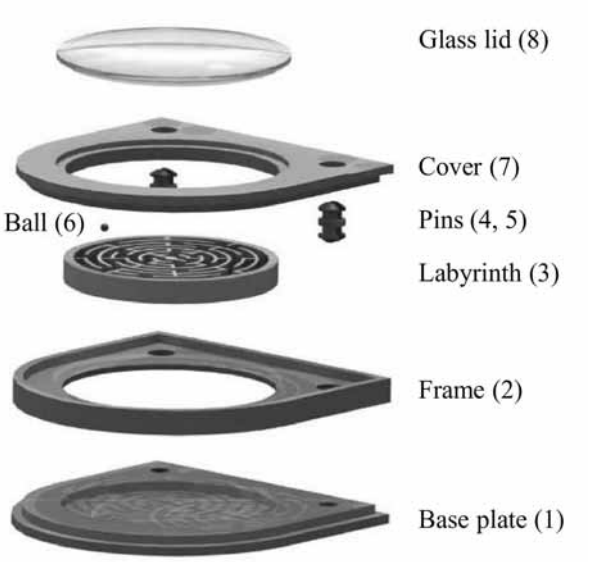

Figure 4: Exploded view of an assembled module according to [7].

For the desired complexity consideration, the assembly is reduced to a *pick & place application*. The assembly of the module in Figure 4 is subject to six rules. Four of the rules are shown in [7]. Rules 5 and 6 were added:

1. Base plate (1) must be mounted first.
2. Pins (4, 5) may only be mounted after cover (7).
3. Comprehensive parts must be mounted after enclosed parts.
4. The glass cover (8) must be mounted last.
5. Each part is only assembled once and the pins (4, 5) are not identical.
6. Gripping attempts on empty TLs of the TZ are prohibited.

In the following two sections, the learning of an assembly strategy with first one agent for both system variants and then using several agents for the second variant is examined.

3 Learning with one Agent

According to Section 1.2, the RL framework consists of an agent and an environment. In the application case under study, the robot forms the agent and the BC and AS form the environment. In this section, the learning of an assembly strategy based on this RL framework is examined for the two system variants of the AC.

3.1 System variant with one TL as TZ

In the case of only one TL, the robot has two possible actions $A = \{0,1\}$. The action $a = 0$ encodes the task *None* and the action $a = 1$ encodes the task *Pick&Place*. The state of the environment results from the TZ allocation and the assembly state of the module on the AS. In the case of only one TL, there are nine possible states of the TZ with $S_{TZ} = \{0,1, \dots 8\}$. The state value 0 stands for no component and the values greater than zero for a component according to the part numbers in Figure 4. The final assembled module consists of eight individual parts. Accordingly, the assembly state on the AS can be represented by an 8-tuple. Each tuple element describes the non-assembly or assembly of a component by the values $0,1, \dots ,8$.

A state $s \in S$ of the entire environment is, therefore, described by a 9-tuple. The first eight elements describe the assembly state on the AS and the ninth element the state of the TZ. Figure 5 shows the representation of states $s \in S$ of the environment with a state vector (SV).

State s_0 represents the initial state, s' a possible later subsequent state and s_{target} the target state. The indicated state s' encodes an assembly of the components' base plate and labyrinth as well as the allocation of the TZ with a component glass lid. The target state s_{target} is reached when the first eight elements of SV are not equal to zero. The character X in the initial and target state stands for any state $s_{TZ} \in S_{TZ}$ of the TZ.

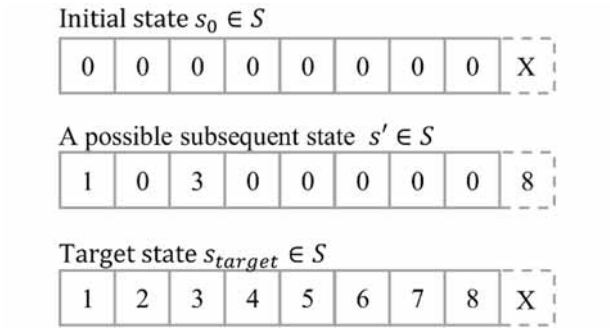

Figure 5: Mapping of the state as a state vector (SV).

The environment reacts to an action a of the agent, as shown in Figure 2, with a reward value r and a subsequent state s'. The reward model R of the environment defines three possible rewards for an action $a = 1$ based on the six assembly rules according to Section 2:

- $r = -\infty$, if the action is not allowed,
- $r = 0$, if the action is allowed and $s' \neq s_{target}$ and
- $r = 1$, if the action is allowed and $s' = s_{target}$.

The action $a = 0$, i.e. a refusal to mount the component on the TL, always leads to the reward value $r = 0$.

The state transition model T works according to the Markov Decision Process (MDP) paradigm. Listing 1 shows the specification of T in MATLAB pseudocode.

```
%A={0,1} action set of the agent
1  num_parts = 8;
2  idx = num_parts + 1;
3  a = action(agent); %current action
%Pick&Place only
4  if a > 0
5    SV(SV(idx)) = SV(idx);
6    SV(idx)     = 0;
7  end
%Pick&Place and None
8  SV(idx) = randi(num_parts +1) - 1;
```

Listing 1: State transition model for an TZ with one TL.

With a permitted action $a = 1$ (*Pick&Place*), the last element of the SV that represents the part in the TC is restored according to its value in the SV (lines 4-7), thereby updating the assembly state. In addition, for each allowed action $a = 1$ or $a = 0$ the last element of SV is assigned a random integer value from the interval $[0, num\_parts]$, which represents a new input part in the TC (line 8).

Each episode starts with the transfer of a start state s_0 of the environment to the agent. A reward is not transferred at the beginning of an episode. Learning is done as described in Section 1.2. The column dimension of the Q-matrix corresponds to the cardinality of the action set $|A| = 2$. An episode ends when the agent receives a reward, $r = 1$, from the environment. The training must include enough episodes to learn a strategy, π. It should be noted that, with the random generation of new input parts in the TZ, the state transition model T is subject to stochastic influences.

Without assembly rules, this system structure results in $2^n(n+1) - n$ states of the environment depending on the number of parts n to be assembled. Due to assembly rules, the number of states is considerably reduced. For the assembly under consideration with eight components, 144 states result according to [7].

The complexity of an algorithm as a function of the input data is described with the big O notation [17]. An empirical analysis showed that a linear time complexity $O(n)$ of the learning process resulted depending on the number of components, n, to be assembled. The computing time for learning an assembly strategy was less than one hour on a standard PC with an implementation in MATLAB.

3.2 System variant with three TLs as TZ

In the variant with three TLs the allocation of the TZ is analogous to a shift register. In fixed time units, the BC moves one position to the right. The robot always has access to all three TLs. This results in four possible actions for the robot, $A = \{0,1,2,3\}$. The action $a = 0$ again encodes the task *None* and the other three actions encode the task *Pick&Place* with different parameterization depending on the TL to be approached.

The state of the environment expands by two elements to an 11-tuple. The first eight elements describe the assembly state on the AS. The other three elements each describe the non-occupancy or occupancy of a TL.

Due to the three TLs, the number of possible states of the environment increases. For the TZ with three TLs and eight possible input parts, as well as the case of non-occupancy of a TL, results in 729 states $s_{TZ} \in S_{TZ}$ with $S_{TZ} = \{(0,0,0), (1,0,0), \dots, (5,1,1)\}$. The values 1 to 8 encode a component according to the part numbers in Figure 4 and the value 0 encodes the non-occupancy of a TL.

The calculation of rewards $r \in R$ is analogous to the variant with only one TL. The state transition model T, which has been extended to a TZ with three TLs, shows Listing 2.

```
%A={0,1,2,3} action set of the agent
1  num_parts = 8;
2  a = action(Agent); //current action
3  idx = a + num_parts;
//Pick&Place only
4  if a > 0
5    SV(SV(idx)) = SV(idx);
6    SV(idx) = 0;
7  end
%Pick&Place and None
%Move TL allocations (TL1→TL2→TL3)
%cardinality |A| is 4
8  for k = |A| -1 : -1 : 2
9    idx = k + num_parts;
10   SV(idx) = SV(idx -1)
11 end
%generate new input part on TL1
12 idx = num_parts + 1;
13 SV(idx) = rand_i(num_parts +1) - 1;
```

Listing 2: State transition model for the TZ with three TLs.

The 11-tuple representing the state of the environment is again implemented as a state vector (SV). If the agent selects a permitted action $a = \{1,2,3\}$ for a *Pick&Place* task (line 2), the input part is taken from the corresponding TL in the TZ and mounted at the AS by the robot (line 4-7). Subsequently, for each allowed action $a \in \{1,2,3\}$ of the robot, the shift register movement is realized by the BC (lines 8-11) and a new input part is randomly generated in the form of an integer value in the interval $[0, num\_parts]$ for the first TL in the TZ (lines 12-13).

Learning a behavioral strategy π is analogous to the representation in Section 3.1. However, despite the applicable assembly rules, the state space of the environment increases exponentially with 144·9·9=11644 states.

The complexity of the learning algorithm corresponds to $O(n^c)$ with n the number of components to be assembled and c the number of TLs in the TZ. In an empirical study, the computing time was about 10 hours and it was about ten times longer than the computing time for the variant with only one TL.

4 Learning with Multiple Agents

Based on the structure of the application problem, a multi-agent approach to reduce the computing time appears to be appropriate for the second system variant with three TLs in the TZ. Figure 6 shows an RL framework consisting of three agents, an environment, and a *management* component.

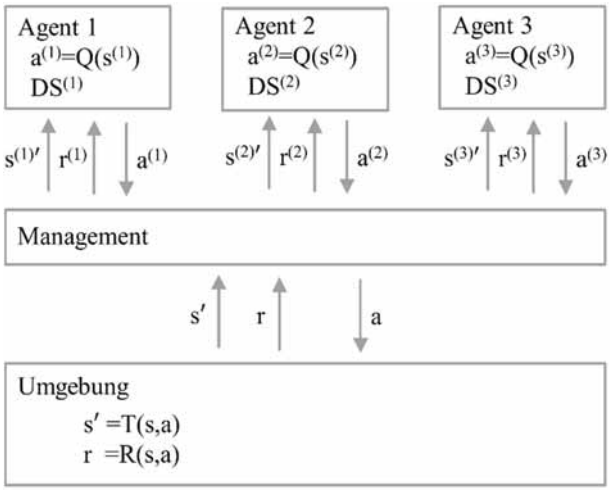

Figure 6: RL framework with multiple agents.

The three agents are identical. Their structure and behavior correspond to the individual agent in Section 3.1, and their action set $A = \{0,1\}$ represents the two tasks *None* and *Pick&Place*. The models T and R of the environment correspond to the representation in Section 3.2. Accordingly, the environment reacts to the action set $A = \{0,1,2,3\}$. In contrast to the original RL framework, the agents and the environment do not communicate directly with each other, but via the intermediate management. The management decomposes each state s' as shown in Figure 7. The first eight elements of s', which encode the assembly state on the AS, are passed on to all three agents. From the ninth to the tenth element, which code the allocation of the three TLs in the TC, each agent only receives the state of one TL at a time.

This reduces the state space from the viewpoint of each individual agent to 144 states, analogous to the agent in Section 3.1.

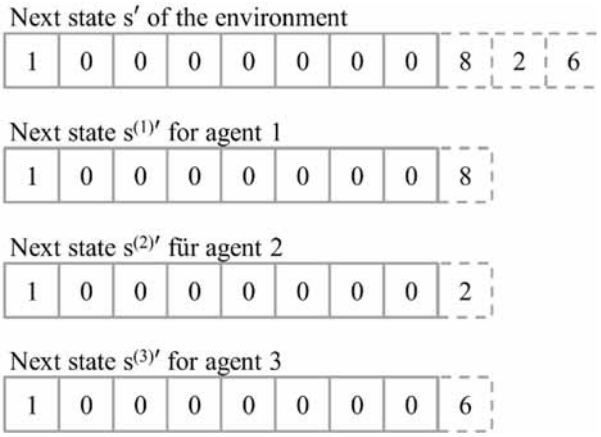

Figure 7: Decomposition of a next state s' of the environment by the management.

An episode begins with the decomposition of the initial state s_0 of the environment by the management, analogous to Figure 7. Then, the management sends to each agent i the corresponding reduced state vector $s^{(i)\prime}$, as shown in Figure 6. Each agent reacts individually with an action $a^{(i)} \in \{0,1\}$. The management selects one of the three actions according to the following criteria:

1. An action $a = 1$ (*Pick&Place*) is preferred over an action $a = 0$ (*None*).
2. If several agents send an action $a = 1$, the agent with the highest index i is selected. The prioritization of the agents follows from the way the TZ works as a shift register. Parts on the second and first TL are still available in the next and next but one cycle. According to Figure 7, $Agent\ 1$ has the lowest priority and $Agent\ 3$ the highest.

The management remembers the selected agent. If an action $a = 0$ was selected, the action is forwarded to the environment unchanged. If an action $a = 1$ was selected, the action is converted into a value of the action set $A = \{1,2,3\}$ based on the index i of the agent and sent to the environment as action a. The environment reacts to the action with a reward r and a follow-up state s'. These are calculated with the reward model R as well as the state transition model T according to Section 3.2.

The management sends the reward r as $r^{(i)}$ to the agent i, whose action was selected before. The other agents receive an empty reward value. An empty reward means that there is no learning for the agent in this iteration step because its action has been discarded. The subsequent state s' is decomposed by management as described and the reduced states $s^{(i)\prime}$ are sent to the agents. Each agent learns a behavior strategy $\pi^{(i)}$ related to the TL assigned to it. Finally, an overall assembly strategy π is derived from the individual strategies.

The number of necessary episodes is hardly different from learning with one agent. Since only one agent is active in each iteration step, the state space to be analyzed has a maximum of 144 states. The empirical investigations revealed a linear complexity $O(n)$ of learning depending on the number n of components to be assembled. The multi-agent approach is scalable regarding the selection of components in the TZ and the complexity is, therefore, independent of the number of TLs. The computing time requirement is reduced to about one hour.

5 Conclusion and Outlook

Starting from the basics of RL using the method of Q-learning and the model-based SBC approach, it was shown how, in principle, a task-based control can be learned and executed.

Subsequently, the learning of a typical pick and place strategy for two differently flexible system structures was considered using the example of a robot-based assembly cell. The focus of the consideration was the time complexity of learning. For the simple system structure with binary selection option, it was found that the learning algorithm has a linear complexity $O(n)$ depending on the number n of assembled components. In contrast, the second system structure with c many simultaneous choices per assembly step had an exponential complexity $O(n^c)$ of learning.

As a result, an RL multi-agent framework with the Q-learning method was designed for the second system structure. It could be shown that, for learning with the multi-agent approach, a linear complexity $O(n)$ results as a function of the number n of components to be assembled and that the complexity is independent of the number of simultaneous choices due to the scalability of the approach.

The application example under study is characterized by learning an assembly sequence that conforms to the assembly. The different system structures were purposefully modelled as environments for the RL according to the MDP paradigm. In subsequent investigations, further influencing variables of a production process are to be taken into account during learning, such as the introduction of input parts as a function of buffer capacities. This requires the integration of the RL with a more complex dynamic simulation model of the production, which was not explicitly developed according to the MDP paradigm. Conceptual approaches for such simulation-based RL experiments are presented in Schmidt [18] and Adams [19].

References

[1] Bundesministerium für Wirtschaft und Energie. Was ist Industrie 4.0? (Federal Ministry for Economic Affairs and Energy. What is Industry 4.0?) https://www.plattform-i40.de/PI40/Navigation/DE/ Industrie40/WasIndustrie40/was-ist-industrie-40.html [Retrieved 27-July-2020].

[2] Hägele M., Nilsson K., Pires J.N. Industrial Robotics. In: Siciliano B., Khatib O., editors. Handbook of Robotics. Berlin: Springer Pub; 2008. 963-986.

[3] Nicolescu G., Mosterman P.J. Model-Based Design for Embedded Systems. Boca Raton / FL: CRC Press; 2010.766 p.

[4] Abel D., Bollig A. Rapid Control Prototyping – Methoden und Anwendungen (Methods and Applications). Berlin: Springer Pub., 2006, 400 p.

[5] Turnbull C. What is Virtual Commissioning? https://virtualcommissioning.com/what-is-virtual-commissioning/ [Retrieved 27-July-2020].

[6] Pawletta T., Pawletta S., Maletzki G.: Integrated Modeling, Simulation and Operation of High Flexible Discrete Event Controls. In I. Troch, F. Breitenecker, editors, Proc. Mathematical Modelling - MATHMOD 2009 Feb, Vienna. Argesim Report No. 35, 13 p., ISBN 978-3-901608-35-3

[7] Maletzki, G. Rapid Control Prototyping komplexer und flexibler Robotersteuerungen auf Basis des SBC-Anatzes (Rapid control prototyping of complex and flexible robot controls based on the SBC approach) [Dissertation]. Universität Rostock / Hochschule Wismar; 2013. In: ASIM FBS 25 doi: 10.11128/fbs.25.

[8] Freymann, B. Aufgabenorientierte Multi-Robotersteuerungen auf Basis des SBC-Frameworks und DEVS (Task-oriented multi-robot controls based on the SBC framework and DEVS) [Draft Dissertation]. TU Clausthal / Hochschule Wismar; 2020 (unpublished).

[9] Kunert G. Pawletta T. Generating of Task-Based Controls for Joint-Arm Robots with Simulation-based Reinforcement Learning. SNE – Simulation Notes Europe. 2018; 28(4):149-156. doi:10.11128/sne.28.4.1044

[10] Sutton R., Barto A. Reinforcement Learning. 2nd Edition. Cambridge/ MA: MIT Press; 2012. 334 p.

[11] Jammer D., Pawletta S., Kunert G., Pawletta T. Beschleunigung eines Reinforcement-Learning-Algorithmus durch Parallelverarbeitung für Robotikanwendungen (Accelerate a reinforcement learning algorithm through parallel processing for robotics applications.). In Durak U. et al., editors. Proc. ASIM STS/GMMS Symposium; 2019 Feb; Braunschweig. Wien: ARGESIM Verlag. 49-52. doi: 10.11128/arep.57.

[12] The MathWorks. Reinforcement Learning With MATLAB – Part 1. Ebook. The MathWorks Inc.; 2019. 24 p.

[13] Russel S., Norvig P. Artificial Intelligence: A Modern Approach. 4nd Edition. Cambridge / MA: MIT Press; 2020. 1115 p.

[14] Zai A., Brown B. Deep Reinforcement Learning in Action. Shelter Island / NY: Manning Pub.; 2020. 359 p.

[15] ROS-Industrial. Homepage. https://rosindustrial.org [Retrieved 13-May-2020].

[16] Deatcu, C., Freymann, B., Schmidt, A., Pawletta, T. MATLAB/Simulink Based Rapid Control Prototyping for Multivendor Robot Applications. SNE – Simulation News Europe. 2015; 25(2): 69-78. doi:10.11128/sne.25.2.1029.

[17] Filho W.F. Computer Science Distilled. Las Vegas / NV: Code Energy LLC Pub.; 168 p.

[18] Pawletta T., Durak U., Schmidt A. Modeling and Simulation of Versatile and Adaptable Systems with an Application in Engineering. In Zhang L. et al., editors, Model Engineering for Simulation. Elsevier Inc. Pub., 2019, Chap. 18, 29 p.

[19] Adams S. et al. Reinforcement Learning from Simulated Environments: An Encoder Decoder Framework. In Proc. SCS SpringSim‘20, 2020 May 19-21, Fairfax / VA, 12 p.