Generating of Task-Based Controls for Joint-Arm Robots with Simulation-based Reinforcement Learning

Georg Kunert*, Thorsten Pawletta

Wismar University of Applied Sciences, Fac. of Engineering, Research Group CEA, Postfach 1210, 23952 Wismar, Germany; *georg.kunert@cea-wismar.de*

SNE 28(4), 2018, 149 - 156, DOI: 10.11128/sne.28.tn.10442
Received: October 25, 2018; Revised November 15, 2018;
Accepted November 24, 2018
SNE - Simulation Notes Europe, ARGESIM Publisher Vienna
ISSN Print 2305-9974, Online 2306-0271, www.sne-journal.org

Abstract. The paper investigates how a robot control for a pick-and-place application can be learned by simulation using the Q-Learning method, a special Reinforcement Learning approach. Furthermore, a post-optimization approach to improve a learned strategy is presented. Finally, it is shown how the post-optimized strategy can be automatically transformed into an executable control using the simulation-based control approach.

Introduction

The conventional robot oriented programming focuses on a special type of robot and on one explicit control problem. That means, one special program is developed to solve one problem. Typical programming approaches according to [1] are: (i) offline programming with a specific robot programming (RP) language, (ii) teach-in procedures, (iii) master-slave procedures and (iv) play-back procedures. In conjunction with the RP there are different manufacturer-specific and non-proprietary Computer-Aided-Robotics (CAR) systems, such as KUKA-Sim or EASY-ROB.

Another approach is the specification of the Robot Operating System (ROS) - Industrial for robot controller, which was driven by the administration of the ROS consortium. ROS-Industrial defines a layer model and an interface standard. The layer model supports the integration of CAR systems and the application of different abstract programming approaches.

One abstraction approach established in robotics is the task-oriented programming. Using this approach, tasks that are specified are robot independent. A transformation method is required to execute the task-based control.

Already before ROS-Industrial was launched, the Research Group Computational Engineering and Automation (RG CEA) at Hochschule Wismar started the development of the open Robotic Control and Visualization (RCV) Toolbox for MATLAB (RCV Tbx., www.cea-wismar.de, accessed 05/2018; [3], which supports manufacturer-independent control development.

The RCV toolbox provides a set of abstract robot-oriented commands in MATLAB in analogy to an RP language and a robot control can be developed virtually. In addition, the RG CEA proposed a specific framework and procedure model, called *Simulation Based Control* (SBC) approach [4]. Robot controls can be developed type independent, task-oriented, and model-based using the SBC approach in conjunction with the use of ROS-Industrial or the RCV toolbox.

The usage of a Reinforcement Learning (RL) method in conjunction with a simulated process model to develop a real robot control is investigated in this paper. The approach is illustrated by a typical pick-and-place problem (PP). The PP problem is mapped to a simulation model according to the requirements of the learning algorithm. The control strategy is learned offline using the RL method and then transformed into an executable robot control using the SBC approach. This research is seen as a first step to learn and generate executable robot controls using the SBC approach and integrated learning methods for whole problem classes, such as the class of assembly problems.

The basics of RL and a special approach, the Q-Learning method [5, 6], are explained with the help of an example at the beginning. Thereafter, it is pointed out how a learned behavioral strategy can be improved by a post-optimization. Then it is shown, how a learned and post-optimized strategy can be automatically transformed into an executable robot control using the SCB approach. Finally, an outlook on future work is given.

1 Reinforcement Learning

Besides the supervised and unsupervised learning, the RL forms an independent group of learning methods [6]. The aim of RL is to learn a behavior strategy to propose actions to reach a given target state.

The advantage of this learning approach is that no prior knowledge in the form of training data and no teacher is needed. An RL method is training itself by the trial-and-error principle.

1.1 Basics of Reinforcement Learning

Figure 1 shows the basic structure of the RL approach. An RL method comprises the environment and at least one agent.

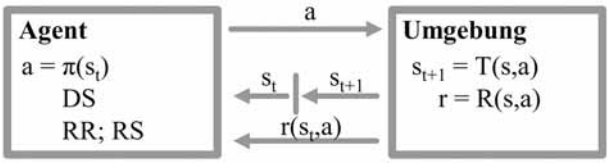


Figure 1: Basic structure of the RL approach.

The environment can enter a finite number of states $s \in S$. The environment communicates its start state s_t to the agent. The agent then reacts with an action a. Based on the action a of the agent, the environment sends a reward r and shifts into a following state s_{t+1}. The reward r is a feedback on the action a of the agent. The interaction between agent and environment is repeated until the environment reaches a successful or unsuccessful target state $s_{target} \in S$. In RL, this iteration until the environment reaches a target state is called an episode.

For Markov-processes of first order the environment generates a reward according to Equation 1 and calculates the following state $s_{t+1} \in S$ according to Equation 2.

$$r = R(s_t, a) \tag{1}$$

$$s_{t+1} = T(s_t, a) \tag{2}$$

The reward model R and the state transition model T are functions of the current state of the environment und the current action of the agent. The following state can also be identical to the state before. The reward is a scalar value. The range of the rewards of the environment is often given by $r \in \{-\infty, -1, 0, +1\}$. A negative infinity means that the action of the agent was not allowed.

A minus one means that the environment is in an unsuccessful target state. If the environment is in a successful target state, the reward is plus one. Actions that are rewarded with a zero, are allowed but did not lead to a target state. These states are states between start state and target state. The agent saves all states $s \in S$ he receives from the environment in a data structure *DS* (see Figure 5). In this way he collects knowledge about the environment. Through the ability of saving states, the agent can identify states that he has previously explored. Other parts of the agent as shown in Figure 1 are described later. *Q* is explained in Section 1.2, while *RR* and *RS* are explained in Section 3.

The training phase of the RL consists of a multitude of episodes. At the beginning, the agent explores the environment solely using the trial-and-error method. In later episodes, the actions of the agent are evaluated based on the collected knowledge using a learning method.

An often used learning method is the Q-Learning. The objective is to reach the target state with as little as possible numbers of state changes of the environment. The result of the learning phase is a learned behavior strategy π of the agent. According to Equation 3, π is a function of the current state of the environment and returns a target-oriented action a.

$$a = \pi(s_t) \tag{3}$$

While learning, the agent is in the so called *exploration-exploitation-dilemma*. Exploitation means learning based on the acquired knowledge and exploration means the study of unknown strategies by trying random actions.

Often, the RL needs many episodes for a successful iterative learning. The computing time depends significantly on the complexity of the state space and the reaction time of the environment. Due to the reaction time, the possibility of learning in the real environment is often not practicable for robotic applications. Therefore, the behavior of the environment is simulated using a model and learning takes place offline.

1.2 Q-Learning

As described before, the goal of the RL is to learn the best possible behavioral strategy π according to Equation 3 to achieve - in the simplest form - a target state.

Q-Learning is a special feature of RL for solving optimization problems. It is based on the idea of learning with the help of a *memory matrix W(a,s$_t$)*, called Q-Matrix for Q-Learning, as introduced by Watkins [7]. The method of Q-Learning exists in different forms [5, 6]. In the following, the approach of *value based learning* (VBL) will be briefly introduced. According to Akhtar [6], VBL represents a special approach to Q-Learning.

With VBL, the agent does not have an internal model of the environment at the beginning. He initially investigates by purely random actions $a \in A$ the states $s \in S$ as well as rewards *r* of the environment and learns, based on this information, iteratively his internal model of the environment, the so-called Q-function *Q(s$_t$,a)*. The Q-function describes the expected benefit, the Q-value, of an action *a* in the state *s$_t$*. The Q-function is mapped inside the agent using a dynamically growing matrix called Q-Matrix. Each row of the Q-Matrix represents an explored state $s \in S$ of the environment and each column represents a possible action $a \in A$ of the agent. That means that the Q-Matrix stores the iteratively learned knowledge of the agent.

At the beginning of the training phase, that means at the start of the first episode, the Q-Matrix is a *0 x n* matrix. The column dimension corresponds to the number of possible actions. For each newly investigated state $s \in S$ the Q-Matrix is extended by one row and all elements of the new row are initialized with 0. The calculation of the Q-value of a current state-action-pair *(s$_t$,a)* can be fulfilled in several ways. Equation 4 shows a simple Q-Learning approach.

$$Q(s_t, a) = r(s_t, a) + \gamma \cdot \max(\mathrm{Q}(s_{t+1}, \forall a \epsilon A)) \qquad (4)$$

As described in the previous subsection, the environment responds to an action *a* of the agent with a following state *s$_{t+1}$* and a reward *r* according to Equation 1 and Equation 2. The benefit *Q(s$_t$,a)* is given by Equation 4 from the sum of the current reward of the state-action-pair *(s$_t$,a)* and the weighted maximum benefit of all follow-up actions. The latter is calculated from the previously established Q-Matrix values. For this purpose, all columns $\forall a \epsilon A$ in the row of the following state *s$_{t+1}$* of the Q-Matrix are analyzed.

The weighting factor γ is called the discounting factor. It ranges between 0 and 1 and controls the impact of expected future rewards on the current decision.

Extended Q-Learning methods additionally introduce a learning rate parameter α. They have the possibility of weighting between the already learned value *Q(s$_t$,a)*, the current reward and the expected future rewards [5,8].

Regardless of the specific Q-Learning method, a large number of episodes are executed in the training phase. An episode starts with a random or known initial state $s \in S$ of the environment. Subsequently, in each episode randomly (*exploration*) or on the basis of already known Q-values (*exploitation*) actions $a \in A$ are tried on the environment in order to improve the internal model of the agent in the form of Q-values. An episode ends when a target state of the environment has been reached. The agent recognizes this by evaluating the returned reward value. An additional parameter may be used for an exploration-exploitation-weighting. This feature can be used for example to decrease the exploration rate as the number of episodes increases.

After the training phase the Q-Matrix represents the learned behavior strategy π. Therefore, in the case of Q-Learning, Equation 3 can be concretized as follows.

$$\pi(s_t) = \mathrm{argmax}(\mathrm{Q}(s_t, \forall a \epsilon A)) \qquad (5)$$

The agent gets the initial state from the environment as the current state *s$_t$*. According to Equation 5, the agent looks in the row associated with the current state *s$_t$* in the Q-Matrix for the maximum Q-value and determines the argument of the column. This represents the best action *a* in the state *s$_t$*. The determined action *a* is executed and the following state *s$_{t+1}$* is received. Thereafter, *s$_{t+1}$* is regarded as the current state *s$_t$* and the next action *a* according to Equation 5 is determined. The iteration is continued until the following state *s$_{t+1}$* transmitted by the environment corresponds to a target state. The agent detects the end of the episode by evaluating the reward value *r*. In this case, rows of the Q-Matrix that represent a target state remain zero, because the agent does not execute any actions after reaching the target state.

The Q-Learning approach is based on a backpropagation algorithm. It is not certain that the learned strategy is optimal, because it is not sure, that the shortest state sequence has been found from a start state to the target state. This problem is discussed in more detail in Section 3. Before, Q-Learning will be illustrated with an example in the next section.

2 Application Example

As an application example, the game *Towers of Hanoi* (ToH), a single player application, has been chosen. This corresponds to a typical *pick-and-place* application for joint-arm robots. The rules of the game ToH are simple and will be explained in the following.

There are three bars. On one bar an arbitrary number of different sized discs are stacked on each other. They are sorted by size and start at the bottom with the largest disc. The aim of the game is to relocate the discs to another bar, whereby only one disc has to be moved per turn. Furthermore, it is not allowed to put a larger on a smaller disc. The storage options are defined by the bars. For the purposes of the RL, the bars and discs form the environment and the player, later the robot, is the agent. The states of the environment result from the positions of the disks on the bars. The movement of discs are actions of the agent that lead to state transitions of the environment. Figure 3 shows all possible states of a two-disc game.

The rules of the game are to be modeled in state transition model T of the environment. According to Equation 2, T generates the following state s_{t+1} as a reaction on an action a. If an action a was not allowed, the following state s_{t+1} returned by T remains the current state s_t.

Figure 3 shows the state transition diagram for an environment with two discs. The numbering of the states corresponds to the numbering in Figure 2. Furthermore, a reward model R according to Equation 1 is specified in the environment.

The rules of the game provide rewards $r \in \{0, 1, -\infty\}$. The reward $r = 1$ represents, that the target state s_9 as depicted in Figure 2 has been reached.

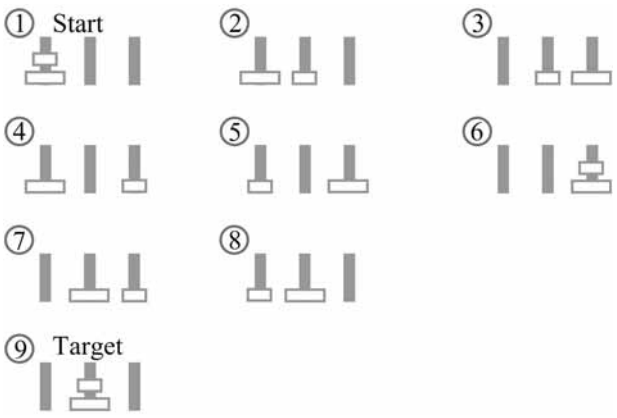


Figure 2: All states of ToH with two disks.

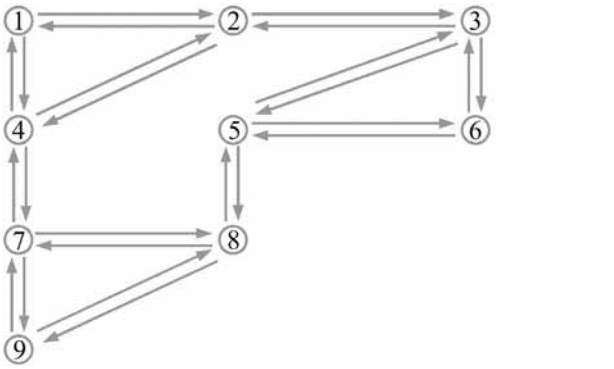


Figure 3: State transition diagram for ToH with two discs.

Illegal actions a receive the reward $r = -\infty$. The reward $r = 0$ stands for reaching a following state s_{t+1}, which is not a target state.

A reward $r = -1$ is not used here because there is no target state that is considered to be a failure. Moving a disc to another bar is an action a. As shown in Figure 4, the agent at ToH always has a choice of six actions. The restriction of the action set A to these six actions implies that: (i) a picked up disk can only be placed on one of the other two bars and (ii) only one disk can be taken at a time.

From the action set A shown in Figure 4, one can derive that the Q-Matrix of the agent has the column dimension of six. The number of rows grows dynamically during learning. At the start of the training phase no state is known for the agent.

Figure 2 shows the fixed initial state s_1, for ToH. This state is communicated to the agent as the first state s_1 of the environment.

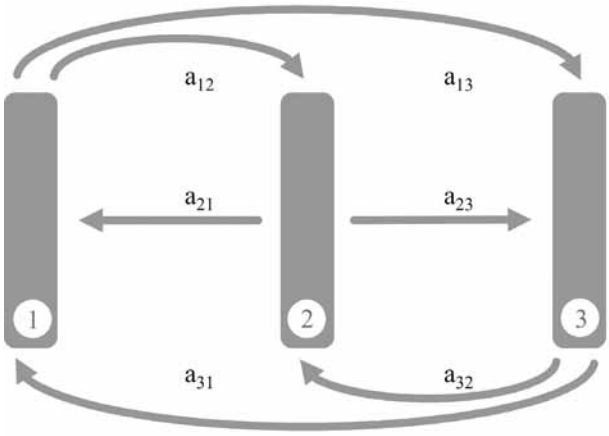
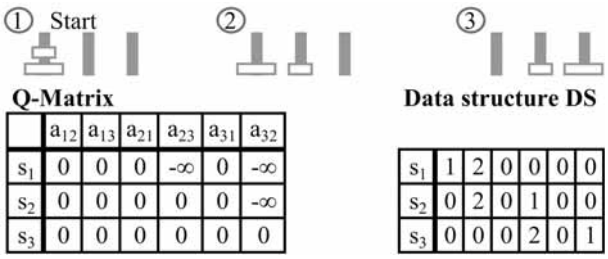


Figure 4: Actions set A of the agent at ToH.

Each state s_t is characterized by the positions of the two discs on the three bars, which can be mapped to six attributes. The current values of the state attributes are stored in the agent in the data structure *DS*, as shown in Figure 5. The attribute values have the following interpretation: 0 no disc; 1 small disc; 2 big disc.

The first element (attribute) of each line describes the existence or size of the upper disk of the first bar and the second the existence or size of the lower disk of the first bar. The third to sixth elements of each row similarly describe the states of the second and third bars.



	a_{12}	a_{13}	a_{21}	a_{23}	a_{31}	a_{32}
s_1	0	0	0	-∞	0	-∞
s_2	0	0	0	0	0	-∞
s_3	0	0	0	0	0	0

s_1	1	2	0	0	0	0
s_2	0	2	0	1	0	0
s_3	0	0	0	2	0	1

Figure 5: Possible development of the Q-Matrix and the data structure *DS* of the agent at beginning of the training phase.

During the training phase, each following state $s_{t+1} \in S$ communicated by the environment is analyzed in the agent. If the state is not yet contained in *DS*, then: (i) the state is stored in *DS* and (ii) the Q-Matrix is extended by one row in which all elements are zero. If the state was already known or was newly recorded, then according to Equation 4 the utility *Q* of the last state action pair (s_t,a) is calculated.

For this, the reward *r* communicated by the environment is evaluated. In this example, a discount factor γ of 0.8 was chosen. As shown in Figure 5, at the beginning of the training phase, often only new states are recorded. Thus, new states are building up the Q-Matrix with zero elements. The iterative learning of the utility values $Q(s_t,a)$ takes place in subsequent episodes.

Figure 6 shows examples of Q-Matrices calculated in different episodes. Ideally, in the first episode, all states $s \in S$ of the environment are explored by the agent. As shown in Figure 6, the Q-Matrix then has the row dimension nine.

After completion of episode 1, the Q-Matrix further shows that the state-action-pair (s_8,a_1) with utility value 1 leads to the target state (see Figure 2 and Figure 4). The Q-Matrix of episode 2 shows a possible learning progress.

Episode 1

	a_{12}	a_{13}	a_{21}	a_{23}	a_{31}	a_{32}
s_1	0	0	-∞	-∞	-∞	-∞
s_2	0	0	0	0	0	-∞
s_3	0	0	0	0	0	-∞
s_4	0	-∞	-∞	0	0	0
s_5	0	0	0	0	0	0
s_6	-∞	0	0	0	0	0
s_7	-∞	0	0	-∞	0	0
s_8	1	0	-∞	0	0	0
s_9	0	0	0	0	0	0

Episode 2

	a_{12}	a_{13}	a_{21}	a_{23}	a_{31}	a_{32}
s_1	0	0	-∞	-∞	-∞	-∞
s_2	-∞	0	0	0	0	-∞
s_3	0	0	0	0	0	-∞
s_4	0	-∞	-∞	0	0	0
s_5	0	0	-∞	0	0	0,8
s_6	-∞	0	-∞	-∞	0	0
s_7	-∞	0	0	-∞	0	0
s_8	1	0	-∞	0	0	-∞
s_9	0	0	0	0	0	0

$Q(s_8,a_1)=r(s_8,a_1)+\gamma\cdot\max(Q(s_9, \forall a\epsilon A))$

$1 = 1 + \gamma \cdot 0$

$Q(s_5,a_5)=r(s_5,a_5)+\gamma\cdot\max(Q(s_8, \forall a\epsilon A))$

$0{,}8 = 0 + \gamma \cdot 1$

Episode n

	a_{12}	a_{13}	a_{21}	a_{23}	a_{31}	a_{32}
s_1	0,41	0	-∞	-∞	-∞	-∞
s_2	-∞	0,51	0	0	-∞	-∞
s_3	-∞	-∞	0,64	0	0	-∞
s_4	0	-∞	-∞	-∞	0	0
s_5	0	0	-∞	-∞	-∞	0,8
s_6	-∞	-∞	-∞	-∞	0	0
s_7	-∞	-∞	0	-∞	0	1
s_8	1	0	-∞	0	-∞	-∞
s_9	0	0	0	0	0	0

Figure 6: Possible development of the Q-Matrix during the training phase.

Due to the backpropagation algorithm, the learning phase in ToH is completed, when an element of the first row of the Q-Matrix has a value greater than zero. This case is shown in the Q-Matrix of episode n.

After the training phase the learned behavioral strategy π is derived from the Q-Matrix according to Equation 5. The example in Figure 7 shows the generation of actions based on π. *E* stands for the environment and *Ag* for the agent. The example refers to the states $s \in S$ in Figure 2, the actions $a \in A$ in Figure 4, and the Q-Matrix of episode n in Figure 6.

From the example results the sequence of state-action-pairs:

$[(s_1,a_{12}), (s_2,a_{13}), (s_3,a_{21}), (s_5,a_{32}), (s_8,a_{12}), (s_9, \text{cancel})]$.

A comparison with the state transition diagram in Figure 3 shows that no optimal π was learned, because the shortest 'path' from the start to the target state was not found.

```
E:   s_t = s_1
Ag:        π(s_1)=a_12
E:   s_t+1 = s_2
Ag:        π(s_2)=a_13
E:   s_t+1 = s_3
Ag:        π(s_3)=a_21
E:   s_t+1 = s_5
Ag:        π(s_5)=a_32
E:   s_t+1 = s_8
Ag:        π(s_8)=a_12
E:   s_t+1 = s_9
Ag:        s_t+1 == target state → cancel
```

Figure 7: Example for the generation of actions based on π.

3 Post-Optimization of the Behavior Strategy

The goal of post-optimization is to search for a shorter sequence of state-action-pairs that map the shortest *path* from the start to the target state (Figure 3). For this it is reasonable to save all rewards *r* and following states s_{t+1} communicated by the environment in the agent during the training phase. For this purpose, two more matrices *RR* (Received-Reward) and *RS* (Received-followingState) are set up dynamically in the agent (Figure 1). Figure 8 shows the *RS* and *RR* matrices of the considered ToH example. Their structure corresponds to the Q-Matrix. The row index represents the states explored by the agent and the columns correspond to the action set *A*. In *RS*, as in the example of ToH, a deterministic behavior of the environment is assumed.

The post-optimization is based on an iterative recalculation of the Q-Matrix using the knowledge stored in the matrices *RR* and *RS*. Listing 1 shows the algorithm as pseudocode and Figure 9 shows the resulting Q-Matrix after the first, second and last (ninth) iteration.

RR-Matrix

	a_{12}	a_{13}	a_{21}	a_{23}	a_{31}	a_{32}
s_1	0	0	-∞	-∞	-∞	-∞
s_2	-∞	0	0	0	-∞	-∞
s_3	-∞	-∞	0	0	0	-∞
s_4	0	-∞	-∞	-∞	0	0
s_5	0	0	-∞	-∞	-∞	0
s_6	-∞	-∞	-∞	-∞	0	0
s_7	-∞	-∞	0	-∞	0	1
s_8	1	0	-∞	0	-∞	-∞
s_9	0	0	0	0	0	0

RS-Matrix

	a_{12}	a_{13}	a_{21}	a_{23}	a_{31}	a_{32}
s_1	2	4	1	1	1	1
s_2	2	3	1	4	2	2
s_3	3	3	5	6	2	3
s_4	7	4	4	4	1	2
s_5	3	6	5	5	5	8
s_6	6	6	6	6	5	3
s_7	7	7	4	7	8	9
s_8	9	7	8	5	8	8
s_9	0	0	0	0	0	0

Figure 8: *RR*- and *RS*-Matrix after the training phase.

set all elements of the Q-Matrix to zero
for i =1 to |S|
  for t =1 to |S|
    for a =1 to |A|
      $Q(s_t, a) = RR(s_t, a) + \gamma \cdot \max(Q(RS(s_t, a), \forall a \epsilon A))$
    end for
  end for
end for
Scaling of the Q-Matrix values between [-∞, 1]

Listing 1: Pseudocode of the post-optimization algorithm.

Q-Matrix - 1. Iteration

	a_{12}	a_{13}	a_{21}	a_{23}	a_{31}	a_{32}
s_1	0	0	-∞	-∞	-∞	-∞
s_2	-∞	0	0	0	-∞	-∞
s_3	-∞	-∞	0	0	0	-∞
s_4	0	-∞	-∞	-∞	0	0
s_5	0	0	-∞	-∞	-∞	0
s_6	-∞	-∞	-∞	-∞	0	0
s_7	-∞	-∞	0	-∞	0	1
s_8	1	0,8	-∞	0	-∞	-∞
s_9	0	0	0	0	0	0

Q-Matrix - 2. Iteration

	a_{12}	a_{13}	a_{21}	a_{23}	a_{31}	a_{32}
s_1	0	0	-∞	-∞	-∞	-∞
s_2	-∞	0	0	0	-∞	-∞
s_3	-∞	-∞	0	0	0	-∞
s_4	0,8	-∞	-∞	-∞	0	0
s_5	0	0	-∞	-∞	-∞	0,8
s_6	-∞	-∞	-∞	-∞	0,6	0
s_7	-∞	-∞	0,6	-∞	0,8	1
s_8	1	0,8	-∞	0,6	-∞	-∞
s_9	0	0	0	0	0	0

Q-Matrix - 9. Iteration

	a_{12}	a_{13}	a_{21}	a_{23}	a_{31}	a_{32}
s_1	0,51	0,64	-∞	-∞	-∞	-∞
s_2	-∞	0,51	0,51	0,64	-∞	-∞
s_3	-∞	-∞	0,64	0,51	0,51	-∞
s_4	0,8	-∞	-∞	-∞	0,51	0,51
s_5	0,51	0,51	-∞	-∞	-∞	0,8
s_6	-∞	-∞	-∞	-∞	0,64	0,51
s_7	-∞	-∞	0,64	-∞	0,8	1
s_8	1	0,8	-∞	0,64	-∞	-∞
s_9	0	0	0	0	0	0

Figure 9: Q-Matrix after first, second and ninth iteration.

At the beginning of post-optimization, all elements of the Q-Matrix are set to zero. The formula for the recalculation of utility values in Listing 1 is of the same structure as Equation 4. The reward of a state-action-pair (s_t,a) results from the *RR* matrix. The following state s_{t+1} is determined from the *RS* matrix.

The innermost loop iterates over all actions of a state and the middle loop over all known states. This is a single recalculation of all elements of the Q-Matrix. The outer loop implements the backpropagation approach. The utility values are thereby developed from the target state to the start state, as shown partly in Figure 9.

The number of repetitions of the computation of all Q-Matrix elements results from the number of known states (row dimension of Q-Matrix). Finally, a scaling of the Q-Matrix values can optionally be done.

All nine states were explored during the ToH example in the training phase. The result of post-optimization is the Q-Matrix after the ninth iteration, as shown in Figure 9. Analogous to the generation of actions on the basis of the learned behavior strategy π in Figure 7, the following shorter sequence of state-action-pairs is identified using the post-optimized Q-Matrix:

$[(s_1,a_{13}), (s_4,a_{12}), (s_7,a_{32}), (s_9, \text{cancel})]$.

4 Generating a Robot Control

With a previously defined behavior strategy π, an executable control specification can be generated automatically. It is based on the structure of the Simulation Based Control (SBC) approach in [4], as shown in Figure 10.

The control problem is mapped to the control and process model in a task-oriented way. For simple problems such as the ToH example, a separate process model can be dropped.

The interface model (IM) implements a robot type independent task transformation and is implemented with the Robotic Control and Visualization (RCV) Toolbox for MATLAB [3]. An interpreter on the controller of the respective robot acts as the robot type dependent layer according to [3].

The ToH application can be specified using three tasks: (i) *move(EulerC)*, (ii) *pick()*, and (iii) *place()*. The task *move* takes a vector with the six Euler coordinates of the position to proceed to as the parameter *EulerC*. The other two tasks are for picking and placing a disc. The three tasks are implemented as reusable software components. Due to the simplicity of the tasks, the IM was realized as an integral part of each task.

For the automatic generation of the control specification, a state-action-sequence

$[(s_t,\exists a \in A), (s_{t+1}, \exists a \in A), \ldots , (s_{target}, \text{cancel})]$

must be calculated using the behavior strategy π. This is done as explained in Section 2 in Figure 7 using the simulated environment. The task-oriented control specification is then derived from the state-action-sequence.

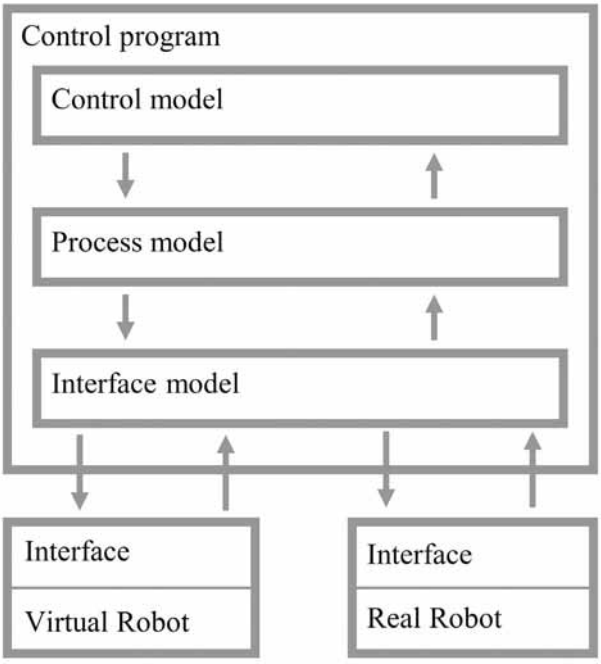


Figure 10: Principle control structure based on the SBC approach.

The ToH application is a typical *pick-and-place* application. Each state-action-pair $(s_t,\exists a \in A)$ can be executed by the same task sequence:

$$move(EulerC) \rightarrow pick() \rightarrow move(EulerC) \rightarrow place().$$

The parameter *EulerC* encodes the x- and y-coordinates of a bar and the z-coordinate for picking or placing the disc. The x- and y-coordinates of a task result from the action *a* (Figure 4) and the z-coordinate from the state s_t (Figure 2). The angle positions are constant.

In Section 3, the post-optimized state-action-sequence: $[(s_1, a_2), (s_4, a_1), (s_7, a_5), (s_9, \text{cancel})]$ was calculated for two discs. The four state-action-pairs are transformed into a control, consisting of three task sequences, each of the following form:

$$move (...) \rightarrow pick () \rightarrow move (...) \rightarrow place ().$$

In order to execute the generated control, the coordinates of the bars, the constant angles and the parameters of the disk geometry must be passed to the control program.

The practical implementation took place in the laboratory of the RG CEA with a joint-arm robot of the type KUKA Agilus, as shown in Figure 11. For technical convenience, the discs were replaced by numbered cubes. The learning program has been implemented for any number of discs. On a standard PC, the training phase for four discs took about two seconds.

5 Conclusion

It has been shown, that a task-oriented control for a joint-arm robot can be learned offline and simulation-based using an RL method. The control is put into operation by using the SBC approach. Furthermore, it has been shown how the solution of a classic learning process can be improved by post-optimization. For the chosen application example, a typical *pick-and-place* application, a task-based control has been successfully learned, generated and executed.

In future work, it will be investigated whether the implemented RL method and the control generation can be applied directly to similar problems or to an entire problem class. Furthermore, applications with human-robot collaboration will be investigated. For those applications, often a complete simulation model of the environment cannot be created and learning has to take place partly online. Hence, it will be examined whether it is possible to learn proactively after the training phase.

In the context of applications with human-robot- or robot-robot-collaboration, RL methods with multiple agents should be investigated.

References

[1] Weber W. *Industrieroboter – Methoden der Steuerung und Regelung (Industrial Robots – Methods of Control and Feedback Control).* Carl Hanser Verlag, München, 2008.

[2] ROS.org. wiki.ros.org/Industrial, accessed 03/2018

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

[4] Freymann B, Pawletta S, Schmidt A, Pawletta T. Design, Simulation and Optimization of Task-Oriented Multi-Robot Applications with MATLAB/Stateflow. *SNE – Simulation Notes Europe.* 2016; 26(2): 83-90. doi: 10.11128/sne.26.2.1033.

[5] Sutton RS, Barton AG. *Reinforcement Learning: An Introduction*. 2nd Edition. Cambridge/MA: MIT Press; 2012. 334 p.

[6] Akhtar SMF. *Practical Reinforcement Learning*. 1st Edition. Birmingham/UK: Packt Publishing Ltd.; 2017. 320 p.

[7] Watkins CJCH. *Learning from Delayed Rewards* Ph.D. thesis, Cambridge Univ. / UK; 1989. 241 p.

[8] Kramer O. *Computational Intelligence.* 1st Edition. Berlin, Heidelberg/DE: Springer Pub.; 2009. 158 p.

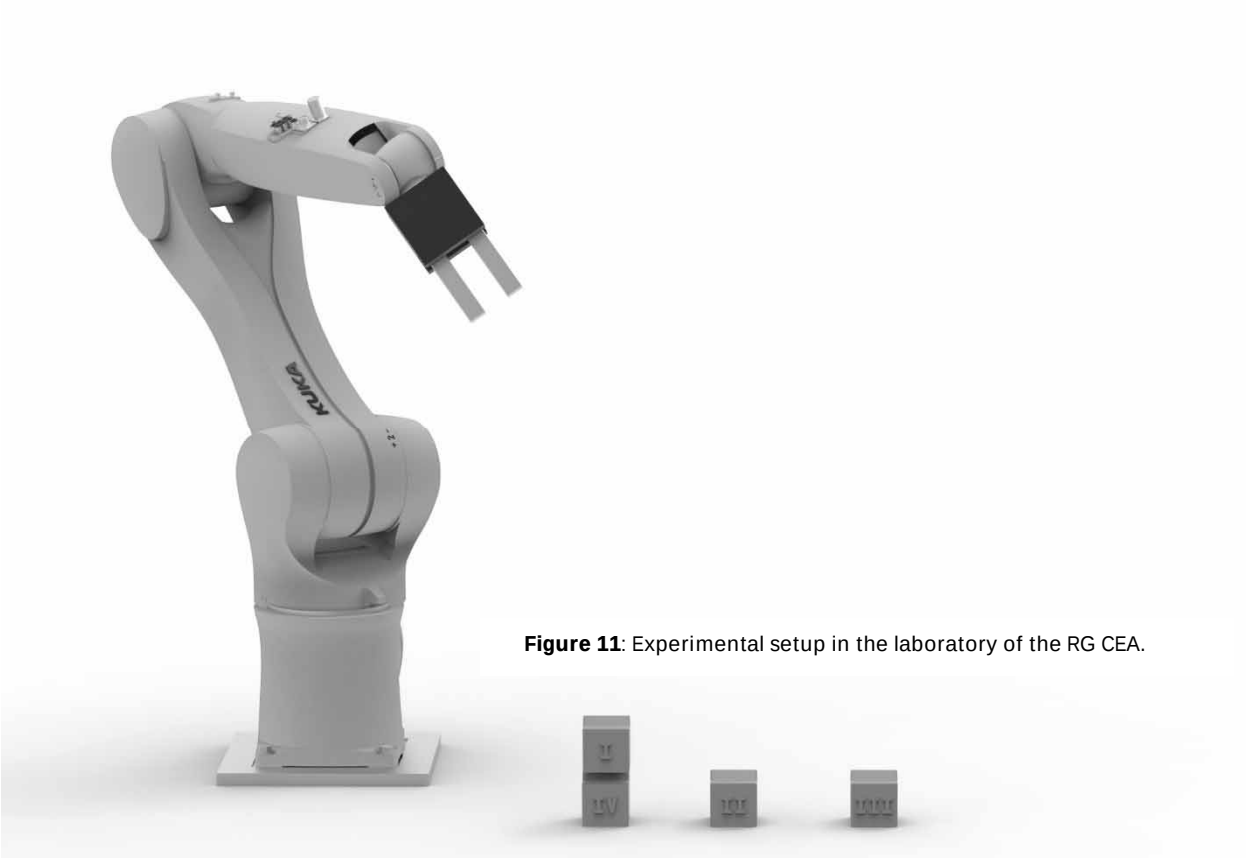


Figure 11: Experimental setup in the laboratory of the RG CEA.