Transferring Human Manipulation Knowledge to Industrial Robots Using Reinforcement Learning

Arexolaleiba, Nestor Arana; Anguiozar, Nerea Urrestilla; Chrysostomou, Dimitrios; Bøgh, Simon

Published in:
29th International Conference on Flexible Automation and Intelligent Manufacturing

DOI (link to publication from Publisher):
10.1016/j.promfg.2020.01.136

Publication date:
2019

Document Version
Early version, also known as pre-print

Link to publication from Aalborg University

Citation for published version (APA):
Arexolaleiba, N. A., Anguiozar, N. U., Chrysostomou, D., & Bøgh, S. (2019). Transferring Human Manipulation Knowledge to Industrial Robots Using Reinforcement Learning. In *29th International Conference on Flexible Automation and Intelligent Manufacturing: FAIM 2019* (Vol. 38, pp. 1508 - 1515). Elsevier. https://doi.org/10.1016/j.promfg.2020.01.136

29th International Conference on Flexible Automation and Intelligent Manufacturing
(FAIM2019), June 24-28, 2019, Limerick, Ireland.

Transferring Human Manipulation Knowledge to Industrial Robots Using Reinforcement Learning

N. Arana-Arexolaleiba[a,*], N. Urrestilla-Anguiozar[b], D. Chrysostomou[b], S. Bøgh[b]

[a]Robotics and Automation Research Group, Mondragon University, Spain
[b] Robotics & Automation Group, Dept. of Materials and Production, Aalborg University, Fibigerstræde 16, Aalborg Øst, DK-9220, Denmark

Abstract

Nowadays in the context of Industry 4.0, manufacturing companies are faced by increasing global competition and challenges, which requires them to become more flexible and able to adapt fast to rapid market changes. Advanced robot system is an enabler for achieving greater flexibility and adaptability, however, programming such systems also become increasingly more complex. Thus, new methods for programming robot systems and enabling self-learning capabilities to accommodate the natural variation exhibited in real-world tasks are needed. In this paper, we propose a Reinforcement Learning (RL) enabled robot system, which learns task trajectories from human workers. The presented work demonstrates that with minimal human effort, we can transfer manual manipulation tasks in certain domains to a robot system without the requirement for a complicated hardware system model or tedious and complex programming. Furthermore, the robot is able to build upon the learned concepts from the human expert and improve its performance over time. Initially, Q-learning is applied, which has shown very promising results. Preliminary experiments, from a use case in slaughterhouses, demonstrate the viability of the proposed approach. We conclude that the feasibility and applicability of RL for industrial robots and industrial processes, holds and unseen potential, especially for tasks where natural variation is exhibited in either the product or process.

Keywords: Reinforcement Learning; Q-learning; Robot Control; Self-Learning Capabilities

[*] Corresponding author. Tel.: +34 647504340
E-mail address: narana@mondragon.edu

1. Introduction

The current trend and challenge in the emerging era of Industry 4.0 is the absence of the human worker during the digital transformation of the smart factories. This is especially true when it comes to introducing new robot systems. Additionally, when building advanced robotic systems, the process knowledge possessed by the human worker is often paramount for the efficiency of the process and the product's quality. However, the transfer of this, often tacit, knowledge to the robot remains a difficult problem.

Industrial robots are commonly programmed in an inflexible way. Setting up the motions for simple pick-and-place operations often require many adjustments and does not generalize well to new situations. Small variations in the process setup will often result in task failure. The current state-of-the-art in flexible robot programming either entails using *robot skills* concepts [1,2] or kinesthetic teaching combined with robot skills [2]. However, these approaches still suffer from the ability to take new and small variations in the process into account. The general problem of current robot programming points to the question: can a robot more easily and naturally learn from the (near-) optimal trajectories carried out by a human expert worker and be able to adapt to future small variations?

In this paper, an industrial robot setup with self-learning capabilities using Reinforcement Learning (RL) is proposed in order to demonstrate how human workers can transfer their task behavior to the robot without any programming requirements. The use case takes place in meat processing in slaughterhouses, which are often characterized by heavy pick and place manipulations. The research is a three-step development consisting of:

1. Acquire human motions and behaviors through recordings in virtual reality (VR)
2. Train the RL model from the recorded trajectories
3. Evaluate the final policy on the real robot system

2. Related work

Reinforcement Learning has previously proved to be successful in different environments such as board games [3] and computer games such as Atari [4,5]. The sensory input varies in different applications of RL, and in recent years many demonstrations in game environments have applied visual sensory input such as raw pixel data applying a Deep Convolutional Neural Network (CNN) as part of the function approximator [4].

Exploration in environments with a sparse reward has been a persistent problem in Reinforcement Learning, which is often true in many robot applications as well [6]. Usually, the reward is triggered at the very end of an episode when the agent reaches its goal. This means that most of the experiences do not yield any reward, which makes it challenging and time-consuming to train the robot agent.

There are studies where training data is obtained from a Virtual Reality (VR) system simulating the real environment [7]. This can in many cases help accelerate the training due to the availability of more data in the simulation environment vs. the real-world environment.

In other work such as [8], an RL agent learns how to control a robot to complete a *wire loop game*[†]. Our presented work builds on top of some of the same ideas and takes the human-factor and training data into account, by recording trajectories from human workers in VR, training the RL agent, and finally evaluating the optimal policy on a real robot system.

2.1. Background

Reinforcement Learning

Reinforcement Learning (RL) [9] is a category of machine learning where the actions an agent must take are based on a reward function, and the agent must explore an environment in order to learn how to behave and maximize the performance. The main difference with other machine learning techniques, such as unsupervised and supervised

[†] https://en.wikipedia.org/wiki/Wire_loop_game

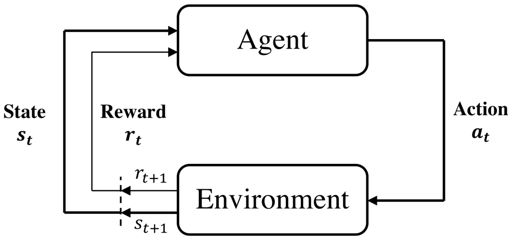

Fig. 1. The Reinforcement Learning environment.

learning, is that no external labelled data is introduced, and the agent is responsible for exploring the unknown environment.

As shown in Fig. 1, the agent learns from a direct interaction with its environment. It selects an action a_t, e.g. move forward, and this action affects the environment and a state change takes place. From the environment, the agent observes the new state s_{t+1}, and the reward r_{t+1}, after the action is taken. The goal of the learning process is to find the optimal policy, which maximizes the cumulative reward R_t as seen in Eq. 1:

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

where γ in (0,1) is the discount factor.

While learning, the agent needs to explore the environment e.g. by taking random actions, but also exploit the partially learned policy to move to good states that yield a high reward. One of the challenges is to keep a balance between exploration and exploitation. A common policy is an ϵ-greedy policy. The ϵ-greedy policy defines the exploration-exploitation rate and how it changes over time, going from a high exploration ($\epsilon = 0.9$) behavior to an exploitation one ($\epsilon = 0.1$).

Q-learning

Q-learning is an off-policy RL method that enables the agent to select the optimal action, by following the optimal policy for a given *Markov Decision Process* (MDP) [10]. A stochastic process has the Markov property if the conditional probability distribution of the future states of the process only depends on the present state. A set of actions, states and rewards can be described as an MDP episode with e_{MDP} as seen in Eq. 2:

$$e_{MDP} = \{s_0, a_0\ , r_0, s_1, a_1, r_1, \ldots, s_{n-1}, a_{n-1}, r_{n-1}\ \} \quad (2)$$

Q-learning is based on the idea that for each state-action tuple there is an associated Q-value: $Q_{s_t,a_t} = max[r_{t+1}]$. Assuming that the optimal policy *π(s)* is known, the best action for each state. $\pi(s) = argmax_a Q(s,a)$ can be selected. As the policy is unknown, one need to compute the Q_{s_t,a_t}values iteratively using the Bellman equation [10]. In each of the states, the agent is located with the Bellman equation, defined in Eq. 3, the Q-value of the current state is retrieved, considering the reward r_{t+1} received from transitioning from the current state to the next state s_{t+1}, and the max discounted Q-value of the next state $\gamma max_a Q_{s_{t+1},a_{t+1}}$.

$$Q_{s_t,a_t} = r_{t+1} + \ \gamma max_a Q_{s_{t+1},a_{t+1}} \quad (3)$$

In noisy environments, the robustness of the learning process can be increased by considering a part of the new knowledge, avoiding the portion corresponding to the noise. This is done by introducing the learning rate α in the Bellman equation, Eq. 4, where α (0,1). Low α values are commonly used in noisy environments.

$$Q_{s_t,a_t} = Q_{s_t,a_t} + \alpha\left[r_{t+1} + \ \gamma max_a Q_{s_{t+1},a_{t+1}} - Q_{s_t,a_t}\right] \quad (4)$$

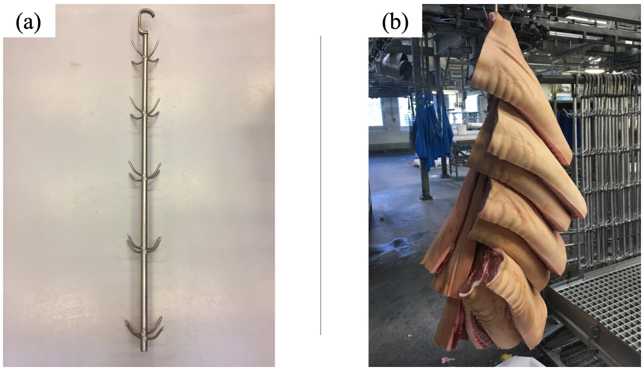

Fig. 2. (a) *Christmas tree* commonly seen in slaughter houses around Europe. The *Christmas tree* contains 20 hooks. (b) Meat is hanged onto the hooks for curing and transportation.

3. Use case

Meat processing industry and slaughterhouses, in particular, are often identified as one of the harshest work environments for humans. Handling of unique items, flexibility, and food safety are only some of the reasons why human hands and minds are still required for competitive production lines in the food industry. Even though novel automation solutions and robotic technologies have slowly been integrated into production lines, a native human-robot collaborative environment is still non-applicable. The natural variation of the products, the machines' inability to be altered and the current high cost of the robot solutions lead to solutions, which usually underperform, are inflexible to quick changes, and unintuitive to use by the operators.

Today, one of the many repetitive tasks found in slaughterhouses, concern hanging heavy pieces of meat onto hooks on what is commonly known in the industry as a *Christmas tree*, see Fig. 2 left. A *Christmas tree* consists of several hooks e.g. 20 hooks. Christmas trees are e.g. used for transportation and are standardized for slaughterhouses around Europe. Currently, 1-2 workers handle large pieces of meat, which generally weigh up to 20 kilos each. The workers have to lift and position the objects accurately onto the hooks on the *Christmas tree*. Such a sequence of movements creates many challenges on the production line. Initially, the high number of heavy, repetitive lifts cause severe stress to the back and arms of the workers. Consequently, they have to stop working every 45 min. in order to regain their strength. A Christmas tree has to be filled within a specific time frame, so the allowed time for detecting the correct pose of the object and choosing the appropriate placement point at the hook is significantly small. Such a small error margin creates an additional mental load to inexperienced workers who often make mistakes that lead to delays in the production and reduced quality.

The vision is to automate the processing of those large pieces of meat as it involves heavy and repetitive lifting. This way, the worker avoids such a stressful task (both physically and mentally). The first step is to have the robot system learn how to handle the pieces of meat and transfer them from a conveyor or table onto a designated hook on the Christmas tree. In the following section, the method and RL architecture applied in this research are explained.

4. Method

The RL architecture used in this research is a two-step training-testing algorithm described in [8]. This algorithm is a modified version of the Q-learning ϵ-greedy exploration and considers a safe exploration space where the robot can learn without any risk of colliding with the physical environment.

There are six discrete actions corresponding to three Cartesian axes (x^-, x^+), (y^-, y^+) and (z^-, z^+). The step value used in each direction is 1mm. To obtain a more accurate path this value can be decreased, but the smaller the step value is, the larger the Q-table becomes, and the time needed for the learning process.

The state-space is defined by the current end-effector Cartesian position $E_P = x, y, z$ and the final goal Cartesian position $G_P = x_g, y_g, z_g$. The tuple composed by those two positions $S = (E_P, G_P)$ is transformed in a unique identifier $id = f(s)$ and recorded in the Q-table. More information can be included, e.g. rotations, but the Q-table easily become unmanageable. In this particular case, the Q-table can easily contain 154,000 states. It is well-known that the Q-learning approach suffers from large state-space dimensions. To overcome this problem, neural network-based approaches, such as DQN, are suggested [5].

The reward function is a linear combination of three weights and distances as seen in Eq. 5. It considers the Euclidean distance change between the actual position E_P and the goal G_P (d_1) and a reward r for achieving the goal (d_2). Moreover, an additional reward function has been tested, where the third parameter (d_3) considers the distance from E_P to the table. We hypothesize that those trajectories that carry the meat by sliding the objects across the table (hence minimizing overall payload), are ergonomically more adequate for the human. For this experiment, the weights of the reward function have iteratively been adjusted. In the example shown in this paper, the following weights are used: w_1=2, w_2=1.0 and w_3=0.1.

$$r = w_1 d_1 + w_2 d_2 + w_3 d_3 \tag{5}$$

In the presented experiment, the following hyperparameters are applied; The discount factor gamma was set to $\gamma = 0.9$ and the learning rate $\alpha = 0.5$. The ϵ-epsilon was initialized to $\epsilon = 0.9$ and the minimum $\epsilon = 0.1$. Those values are the outcome of several simulations. The distance between the initial and final position is around 0.9m. The ϵ decreases following the next equation $\epsilon = \epsilon - 1/N_e$ (Algorithm 1). Boundary conditions for safety purpose are added where the robot is not allowed to explore further than a predefined space.

Algorithm 1. Two-phase RL algorithm based on [8].

```
while condition do
    Result: LEARNING PHASE
    if Final Goal Reached then
        break
    end
    if First Cycle then
        Reset environment to $s_{cycle}^{init}$ and observe state $s'$
        Construct and initialize the $Q_{table}$
    else
        if last state is stable then
            $s_{cycle}^{stable} = s'$
        end
    end
    for Episode ranging from 1 to $n_e$ do
        Reset environment with $s = s_{cycle}^{stable}$
        while true do
            Select action $a$ from $Q$ using Greedy policy
            Take action $a$ and observe $r$ and $s'$
            if state $s'$ is final or unsafe then
                break
            end
        end
        if $\epsilon > \epsilon_{lim} = 0.1$ then
            decrease $\epsilon$, e.g. $\epsilon = \epsilon - 1/N_e$
        end
    end
    ⋮
    Result: TESTING PHASE
    Initialize environment state $s = s_{cycle}^{init}$
    while true do
        Select $a = argmaxQ(s, a)$
        Take action $a$ and observe $r$ and $s'$
        if state $s'$ is final or unsafe then
            break
        end
        Update $Q_{table}$ according to eq. (4)
        if state $s'$ is stable and goal is not reached then
            $s_{cycle}^{stable} = s = s'$
        else
            break
        end
    end
end
```

5. System implementation

The system architecture used in this work is shown in Fig. 3. It contains three nodes: (i) VR data acquisition, (ii) RL learning and (iii) robot control.

The first node (i) tracks and records the user movements by means of the VR system. It is composed of two satellites and a controller. The controller trigger is used to define the starting and ending points of the path, allowing the user to create as many paths as needed. Each path is afterwards transformed into the robot coordinate system.

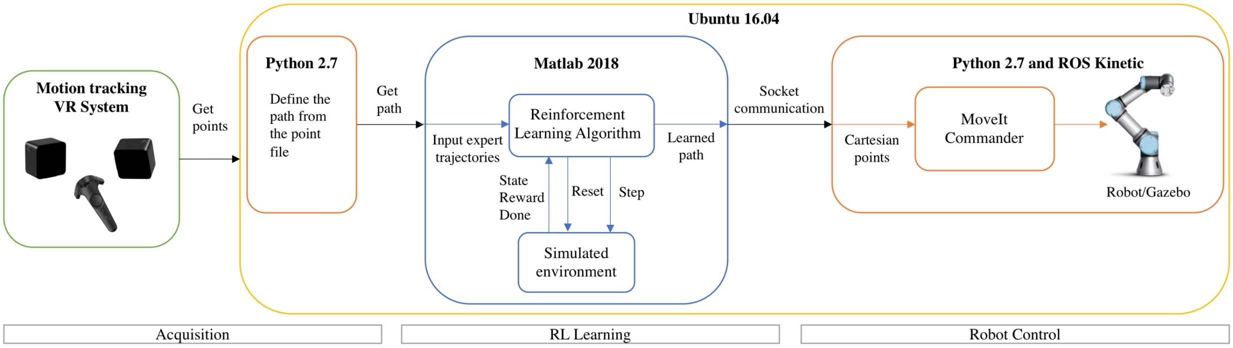

Fig. 3. Software and hardware architecture. Left: (i) Acquisition of VR data. Middle: (ii) Reinforcement Learning block. Right: (iii) Robot control either for Gazebo or the real robot.

The second node (ii) receives the path and launches the RL algorithm described in the previous section. This node learns the policy to track the human behavior. The position information is sent by means of a shared file or a socket communication to the robot. The coding structure of the simulation environment (blue box) uses the same set of functions (reset, step) and values observation, reward and done as does the OpenAI-Gym framework [11]. The RL node sends the reset and step functions (which are in charge of setting up the robot to the initial position and sending the actions to take) to the simulation environment and receive some variables that provide information about the agent such as; the current state, the reward obtained, and if the goal is reached or not. The third node (iii) controls the robot. The MoveIt planner included in ROS distribution is in charge of receiving the Cartesian points and applying the inverse kinematics to translate them into joint movements.

Table 1 lists the primary hardware elements utilized in the experiments. The standard deviation for static pose value obtained by the VR system at 1-2 m distance is 0.417 mm [12] and a sampling speed of 60 Hz. The software dependencies used are also indicated, where the computer system is in Ubuntu 16.04 LTS with ROS Kinetic installed.

Table 1: Hardware and software overview.

Robot manipulator	VR-system	Computer	Software
UR5	HTC VIVE	Quad Core i7	Ubuntu 16.04
Payload: 5 kg	2 trackers	16 GB RAM	ROS Kinetic
Working radius of 850 mm	$\sigma_{position}$: 0.417 mm	Nvidia RTX 2048 8 GB	Python 2.7
Repeatability +/-0.1 mm	Sampling speed: 60 Hz		Gazebo 7

6. Experimental results

6.1. Training

The robot is learning around a zone starting from the initial position T_P. The learning zone is a 3D space with a radius size defined in the hyperparameters. In Fig. 4(a), a trajectory (in green) is obtained by the VR system in the robot's end-effector coordinate system. At the initial point (in black), the robot tries to learn how to go through the trajectory. The robot observes the VR trajectory from what is referred to as a *safe TCP coordinate system*. A TCP is considered to be safe when the robot learns to move forward to a new position belonging to the VR trajectory. In Fig. 4(b) the trajectory from the initial TCP position and from a second TCP position is shown after moving forward. Finally, in Fig. 4(c), all the intermediate safe positions until the final goal G_P are shown.

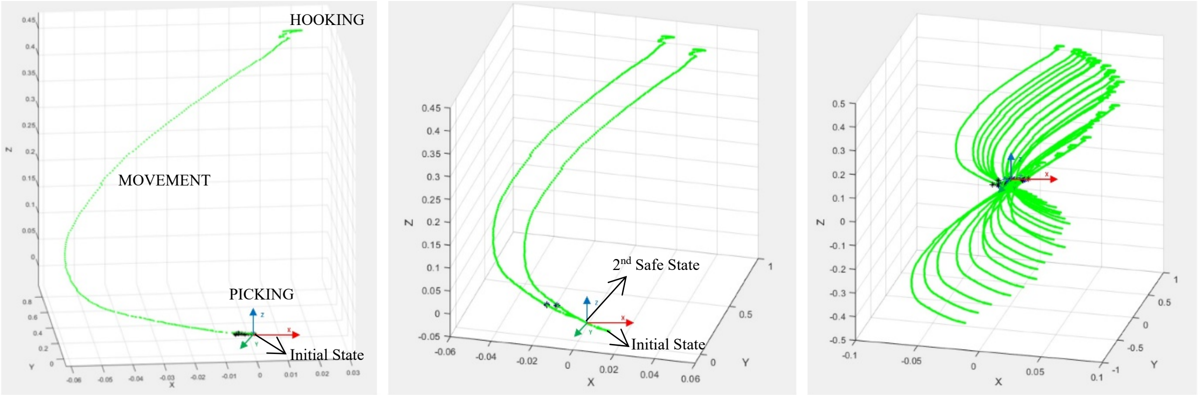

Fig. 4. (a) First learning step (first sub trajectory). (b) Trajectory seen from initial position and second position in the training sequence. (c) All learning points across the trajectory.

6.2. Evaluation

The work is tested in an experimental scenario shown in Fig. 5. In this scenario, an object is picked up from a table and hanged on the hook on the *Christmas tree*, while the VR system is tracking the human behavior. Since the meat on the process is presently not considered, a lightweight soft object was used.

The VR system tracks the human movement with high precision. It was observed that even the trajectory of the hooking action is clearly tracked. However, it was recognized that the VR system does not behave correctly if any shiny surfaces are present in the scene, which may impose tracking errors. The VR controller used is moved by hand. This is not practical for real scenarios, but other wearable trackers can be used instead in order to track the human worker while manipulating real meat in the slaughterhouse.

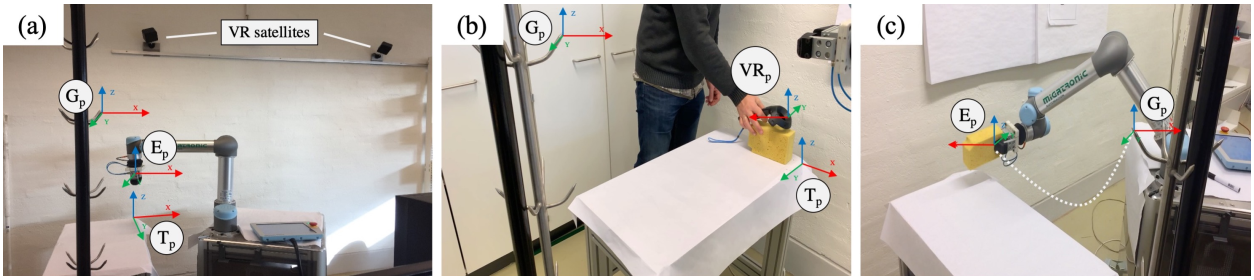

Fig. 5. (a) VR-system sensing the working environment. (b) Initial coordinate T_P (table) and goal coordinate G_P (*Christmas tree* hook). (c) Robot moving towards the final position G_P along the trained trajectory.

After the learning process using the VR information, the robot can go from the initial position T_P to the goal G_P. The learning time can significantly change from 86.39 minutes for a 1 mm robot step and 50 mm radius size of the learning zone to 1.15 minutes for a 5 mm robot step and 50 mm radius size of the learning zone. In order to simplify the problem, picking and placing action are not considered in this work.

7. Discussion

It was observed that a simple task such as placing a piece of meat on a hook is composed of several skills: (i) picking up a piece of meat, (ii) its displacement, and finally, (iii) the placement of the meat on a hook. The present work takes its outset from the second skill, i.e. the displacement of a piece of meat. By use of Reinforcement Learning, the robot was enabled to imitate the trajectory made by the human, captured by a VR system, through a trial and error strategy. Training time varied and was very dependent on the radius of the search and differential movements distance of the robot. In our tests, the training time was between a few minutes to an hour. The reason behind this is due to the number of states generated in each configuration varies significantly, ranging from approximately 30,000 to 154,000. Regardless, this should be considered a fast learning process compared with other alternatives that use a Gazebo simulator or real robots, where the robot is able to perform only 2-4 movements per second. The Q-learning technique is characterized by its simplicity, but also by its limitation when it comes to increasing the number of variables, which defines the state-space of the system. Currently, we are working with strategies within more recent developments in the Reinforcement Learning domain such as *Deep Q-Networks* (DQN) and *Deep Deterministic Policy Gradient* (DDPG), given that these techniques allow for increasing the number of input features and actions.

8. Conclusion

In this paper, a concept for an industrial self-learning robot agent was presented to strive for an optimal policy when moving heavy non-rigid objects in the food industry by use of Reinforcement Learning and Q-learning. The work aims to study the use of machine learning techniques to facilitate the programming of robots by non-experts in real manufacturing use case scenarios. The use case selected consists of transferring manipulation knowledge from a human worker to the robot i.e. the behavior of moving and hooking meat onto hooks in a slaughterhouse. The presented method was applied, which enabled the agent to learn and control a six-axis industrial robot purely from recordings of human behaviors. The training time achieved in this work is practical for industrial use, considering that the learning is performed without any human need.

The presented approach can be generalized to other use cases that share similarities in movement and execution e.g. painting, welding, polishing. This work has the potential to extend beyond the manufacturing industry into e.g. health care. For all these kind of use cases, time is the crucial factor when programming advanced robot systems. With stronger techniques and innovative applications of Reinforcement Learning, it is the first step, which allows us to gain new unseen possibilities in real-world robotics in the future for the greater good of companies and human workers.

Additional future work includes learning picking and hooking skills. We consider that the information of the force (payload) should also be part of the state representation as this can potentially affect the system dynamics drastically.

Acknowledgements

This work was partially supported by the Danish national funded research project MADE Digital, the Basque Government Mobility program (MV _2018_1_0018) and Malgurob (KK--2018/00114) research project.

References

[1] Pedersen MR, Nalpantidis L, Andersen RS, Schou C, Bøgh S, Krüger V, et al. Robot skills for manufacturing: From concept to industrial deployment. Robot Comput Integr Manuf 2016;37:282–91. doi:10.1016/j.rcim.2015.04.002.

[2] Schou C, Andersen RS, Chrysostomou D, Bøgh S, Madsen O. Skill-based instruction of collaborative robots in industrial settings. Robot Comput Integr Manuf 2018. doi:10.1016/j.rcim.2018.03.008.

[3] Tesauro G. TD-Gammon: A Self-Teaching Backgammon Program. Appl. Neural Networks, Boston, MA: Springer US; 1995, p. 267–85. doi:10.1007/978-1-4757-2379-3_11.

[4] Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, et al. Human-level control through deep reinforcement learning. Nature 2015;518:529–33. doi:10.1038/nature14236.

[5] Van Hasselt H, Guez A, Silver D. Deep reinforcement learning with double q-learning. Thirtieth AAAI Conf. Artif. Intell., 2016.

[6] Nair A, McGrew B, Andrychowicz M, Zaremba W, Abbeel P. Overcoming exploration in reinforcement learning with demonstrations. 2018 IEEE Int. Conf. Robot. Autom., 2018, p. 6292–9.

[7] Dyrstad JS, Mathiassen JR. Grasping virtual fish: A step towards robotic deep learning from demonstration in virtual reality. 2017 IEEE Int. Conf. Robot. Biomimetics, 2017, p. 1181–7.

[8] Meyes R, Tercan H, Roggendorf S, Thiele T, Büscher C, Obdenbusch M, et al. Motion planning for industrial robots using reinforcement learning. Procedia CIRP 2017;63:107–12.

[9] Wiering M, van Otterlo M. Reinforcement Learning: State-of-the-Art. vol. 12. Springer Science & Business Media; 2012.

[10] Sutton RS, Barto AG. Reinforcement learning: An introduction. MIT press; 2018.

[11] Brockman G, Cheung V, Pettersson L, Schneider J, Schulman J, Tang J, et al. Openai gym. ArXiv Prepr ArXiv160601540 2016.

[12] Borges M, Symington A, Coltin B, Smith T, Ventura R. HTC Vive: Analysis and Accuracy Improvement. 2018 IEEE/RSJ Int. Conf. Intell. Robot. Syst., 2018, p. 2610–5.