1.3

Optimising Trajectories in Simulations with Deep Reinforcement Learning for Industrial Robots in Automotive Manufacturing

Noah Klarmann[1*], Mohammadhossein Malmir[1*], Josip Josifovski[1*], Daniel Plorin[2], Matthias Wagner[2] and Alois C. Knoll[1]

[1]Technical University of Munich, Germany
[2]AUDI AG, Germany
[*]These authors contributed equally to this work

Abstract

This paper outlines the concept of optimising trajectories for industrial robots by applying deep reinforcement learning in simulations. An application of high technical relevance is considered in a production line of an autmotive manufacturer (AUDI AG), where industrial manipulators apply sealant on a car body to prevent water intrusion and hence corrosion. A methodology is proposed that supports the human expert in the tedious task of programming the robot trajectories. A deep reinforcement learning agent generates trajectories in virtual instances where the use case is simulated. By making use of the automatically generated trajectories, the expert's task is reduced to minor changes instead of developing the trajectory from scratch. This paper describes an appropriate way to model the agent in the context of Markov decision processes and gives an overview of the employed technologies. The use case outlined in this paper is a proof of concept to demonstrate the applicability of reinforcement learning for industrial robotics.

Keywords: deep reinforcement learning, automotive manufacturing, simulation, industrial robotics, virtual learning platform, trajectory optimisation, motion planning, offline programming, robot learning.

Video: A video clip demonstrating the proposed methodology is available at: https://vimeo.com/562948911.

1.3.1 Introduction

A concept for the automatic generation of trajectories for the control of industrial robots is presented in this work. Data-based robotic control has the potential to address two major shortcomings of conventional robot programming [1]:

(i) The classic programming of industrial robots is done manually by a specialist who precisely specifies the trajectory of the robotic arm *Tool Center Point* (TCP). To this end, the programmer is in close contact with the responsible plant engineer as well as the product owner to fulfill all demands, restrictions, and requirements.

(ii) Conventional programming is deterministic and thus prevents the flexible adaptation of the robot to changing environments (like varying products). A variable control that adapts to different conditions cannot be realized with classical programming.

Both shortcomings are addressed in this work by introducing a self-taught and automated method for robot control programming. By adopting the reinforcement learning methodology [2, 3], the control is learned based on sampled experience from the interaction with a virtual environment. In this context, a predefined reward function is optimised that steers the action policy towards the desired outcome of the robotic manipulation. Reinforcement learning combined with non-linear function approximators (e.g., neural networks – referred to as *Deep Reinforcement Learning* DRL) can generalize action policies on experience to solve problems with large and complex state spaces (e.g., learning from unstructured data such as a camera signal). Further advancements in the field of reinforcement learning are algorithms that can deal with continuous action spaces enabling robotic control [4, 5]. These developments have recently led to interesting milestones, such as learning the locomotion of a four-legged robot [6] or robots that have learned to open doors [7]. Well-known achievements in the field of reinforcement learning are based on environments that allow the efficient generation of an abundant amount of experience (e.g., video games [8] or board games [9, 10]). However, robots require physical interaction with the environment whereas training the agent in the real world is exceptionally resource intense. As mentioned in [11], an agent that learns a simple

grasping task requires experience from 800,000 episodes. The application of reinforcement learning to industrial robotics requires a significant amount of data due to the following two characteristics: (i) complex spaces for the state (e.g., learning based on unstructured data such as images) and action spaces (e.g., controlling the torques of each axis), and (ii) industrial requirements regarding safety and precision of the robotic control.

An appealing alternative to real-world training is to simulate the agent-environment interaction and transfer the control to the real world. For the use case described by Rusu et al., a considerable speed increase by the factor of 50 could be achieved when a robot is trained in parallel virtual environments instead of the real world [12]. Another advantage in the use of virtual methods is the avoidance of accidents that could occur during the training of robots in the real world [13].

In this work, DRL is employed to find the optimal trajectory of a robot that is applying PVC sealant on a door frame of a car body to prevent water intrusion and hence corrosion. Moreover, the robotic trajectory will be optimised with respect to the following aspects: (i) providing smooth velocity of the end effector, (ii) ensuring the optimal orientation of the end effector's nozzle to the car body surface, and (iii) avoiding collisions. Related work from other groups in which DRL is used for trajectory planning exists. In [14], a DRL agent learns to control a robot with six axes to solve the hot wire game that is seen as the first step towards industrial applications like welding, gluing, or cutting. A more practical application can be found in [15] where the authors propose a simulated environment for learning the optimal trajectories for applying paint on a car body. Besides learning a concrete task, DRL can also be used for the online optimisation of trajectories for robots with unknown/partially known dynamics that usually lead to control jumps [15–17].

Beyond the application to manipulators, DRL can also be used to generate trajectories for mobile robotics. The path planning for mobile robots with a known map can be found in [18], whereas the navigation with simultaneous map generation is proposed in [19–21]. The application of DRL to optimise the data collection for an agent that explores the environment can be found in [22]. Moreover, DRL to evaluate optimal trajectories for *Unmanned Air Vehicles* (UAVs) providing access points to end-users is presented in [23].

A unique feature of this work is the advanced simulation environment that can simulate the car body and robot with detailed geometry considering realistic physical behavior and a high-end rendering pipeline. In addition, the close collaboration with the industrial partner imposes high industrial

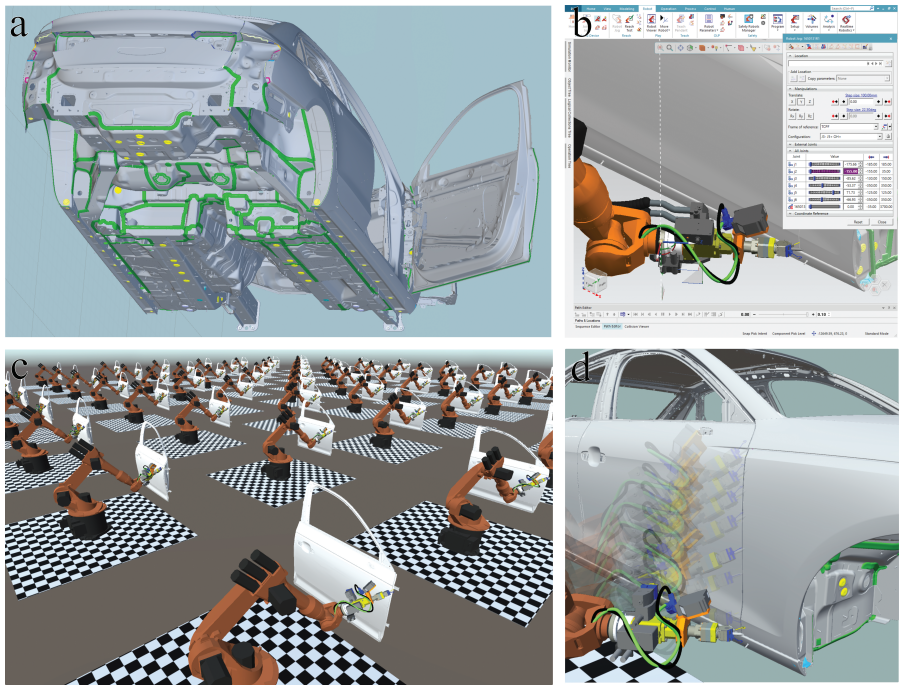

Figure 1.3.1 (a) Manually predefined geometric model of the seam, (b) the path editor and the robot jog in tecnomatix process simulate for manual trajectory programming, (c) multiple training environments running in parallel and optimising the trajectory for a car door, and (d) the robot following the learned trajectory.

requirements on this approach, and thus exceeds the usual academic proof-of-concept state.

1.3.2 Background

AUDI AG employs a fully automated process for applying the sealant material on car bodies using industrial manipulators. To program the manipulator, a conventional procedure is adopted in which fixed trajectories are specified by an expert. To this end, the creation of trajectories for the automation of a new car body requires the following steps: (i) A three-dimensional model of the seam that is supposed to be applied to the car body is designed manually (see the green markers in Figure 1.3.1a). (ii) The expert programs the final robot trajectories in a way that the manually predefined geometric model of the seam will be followed by the manipulator. It is worth

noting that there is a multitude of different nozzles and end effectors that vary according to the task and need to be specified before starting to program the robot. To create the robotic trajectory, the offline programming expert manually defines a sequence of parameterized motions using software tools (see Figure 1.3.1b). Two different ways are typically used to specify the movement of the end-effector along a free or constrained path towards a specific target. The first type is called *Point-to-Point* (PTP) motion that is used when the target pose should be reached as fast as possible by maximizing joint-level rotational speeds without specifying the TCP path. The second option is referred to as *Linear* (LIN) motion and is used whenever the target pose should be reached along a straight line with a specified velocity and acceleration. PTP motions are mainly used in two scenarios: (i) The robot's TCP should be moved to an initial pose configuration that is a suitable starting point and (ii) whenever there is a need to relocate or reorient the robot's TCP between the seam segments. The LIN motions are used whenever the nozzle tool is applying the sealant material as the priority is on accurate motions to guarantee a straight and uniform line.

It is worth noting that manual programming is a tedious and resource-consuming process as time-optimal and collision-free trajectories are complex and rely on many different parameters such as (i) quantitative metrics (e.g., how much time it takes to complete the seaming process), (ii) qualitative aspects (e.g., straight and uniform appearance of the seam), or (iii) expert-subjective considerations that are based on experience.

To reduce the complexity and workload of the task, the programmer can start with an existing trajectory from a different but similar scenario in which a solution already exists. An existing case from which the trajectory can be reused is called a *brown field*. By adopting a brown field, the manual effort is typically reduced tremendously as only minor modifications are necessary in most cases. In the so-called *green field* scenario, the car body is significantly different from any existing scenarios with the implication that the programmer must manually define the trajectories from scratch. In the following section, a procedure is described in which brown fields are generated by a DRL agent.

1.3.3 Methodology

The described problem is addressed by a DRL agent that finds an optimised trajectory in simulation. Figure 1.3.2 depicts the agent's state and action spaces as well as the reward function in the context of a *Markov Decision*

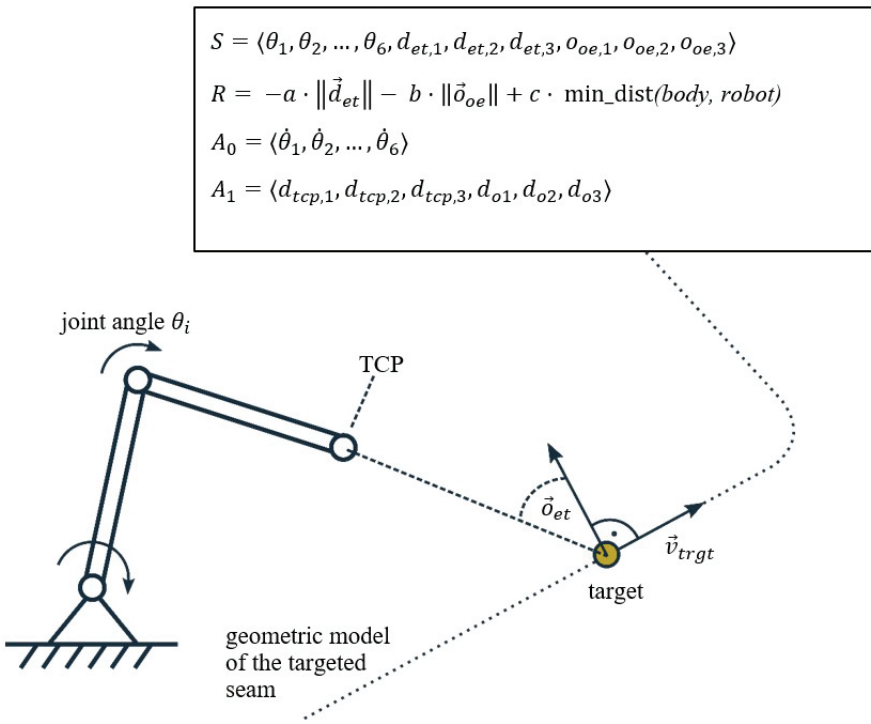

Figure 1.3.2 DRL agent in the MDP formalization for optimising a TCP trajectory by minimizing the distance to a predefined moving target.

Process (MDP). The geometric model of the targeted seam is given in the process as described in the previous section. An imaginary target location is modeled that travels along the predefined path at a specified velocity $\vec{v}_{trgt}$. The first term of the reward R is modeled as the negative distance from the TCP to the target location $\left\|\vec{d}_{et}\right\|$. By maximizing this term, the agent controls the robot's TCP to follow the target. When applying the sealant, the predefined target velocity $\vec{v}_{trgt}$ is required to be at a specific constant velocity to guarantee an optimally applied sealant. The application of the sealant is required to occur in an orientation that is orthogonal to the surface while aligned with the direction of the sealant line. Based on this, a reference end-effector orientation can be defined; the second term of the reward function penalizes the difference between the actual and the reference orientation of the end-effector (namely, the orientation error $\vec{o}_{oe}$, calculated based on the orientation of the target

surface normal w.r.t. the end-effector $\vec{o}_{et}$). The third term of the reward R measures the minimum distance of the manipulator to the car body to avoid collisions. Moreover, three weights (a, b, c) are used to balance each term individually.

Two different ways to define the action space are considered: (i) The robot is controlled on the joint level with the agent controlling the speed (or torque) of each joint individually. This results in a six-dimensional action space (action space A_0) for a robot that has six *Degrees of Freedom* (DoF). (ii) By employing kinematics, the TCP can be controlled directly in the operational space, e.g., by specifying a target delta for each Cartesian space dimension $d_{tcp,1}, d_{tcp,2}, d_{tcp,3}$ and for each orientation axis d_{o1}, d_{o2}, d_{o3} (see action space A_1). The state description comprises the relative position and orientation of the end-effector with respect to the current target point and the angles of all joints $\theta_1, \theta_2, \ldots, \theta_6$ of the manipulator. To simulate the robot interacting with the environment, a simulation developed with the commercial game engine *Unity3D* [24] is used (Figure 1.3.1c and d). *Unity3D* is a suitable choice for the planned undertaking as it provides advanced rendering capabilities (to potentially learn from pixels), the opportunity to write user-defined functions, and it comes with an efficient GPU-accelerated physics engine (*Nvidia PhysX* [25]).

To solve the MDP described above, the algorithm *Proximal Policy Optimization* [26] (PPO) is employed. PPO is a policy gradient method that trains both an actor and a critic function, whereas the policy update gradient is clipped to prevent stability issues. An important feature of PPO is the support of continuous action spaces that is achieved by training probability density functions instead of discrete actions. In this work, the PPO implementation of the *Stable Baselines* library is used (referred to as PPO2 [27]) that allows running multiple workers updating the same policies. Each simulation instance runs several robots at the same time (Figure 1.3.1c), whereas policy gradient updates are gathered in batches for the periodic update of the two policy networks.

The proposed methodology is applied to create brown fields. In a second step, the final control can be derived from the brown fields by the offline programmer. As mentioned before, a significant reduction of the workload can be achieved when the final programming is done based on a brown field provided by the agent. The simulation starts from scratch without any knowledge (starting from a green field). A disadvantage in taking a previous solution into account (e.g., starting the simulation from an existing but incompatible brown field) is a potential bias regarding the solution and might

lead to a local minimum. However, starting from another brown field comes with the high potential to reduce the training time and can be investigated in the future.

One might ask why brown fields are evaluated rather than directly learning the final robotic programming. While it is indeed possible to let the agent define the final trajectory, the human is kept in the loop as the approximative nature of DRL as well as the difference between the real world and the simulation (reality gap) occasionally lead to undesired control behavior.

1.3.4 Conclusion and Outlook

The concept of optimising trajectories with DRL for industrial robots in simulations is outlined in this paper. To this end, a possible MDP formalization of the agent that has the potential to considerably reduce the amount of the manual work that is involved in the offline programming of the industrial robots is presented. For the further course of this undertaking, an adoption of the methodology in three steps is envisioned: (i) The experts are performing plausibility checks by comparing hand-crafted trajectories with the solution from the DRL agent. (ii) The agent is used in production by creating brown fields from which the expert derives the final solution. (iii) The agent finds final robotic trajectories and the human experts verify the solution without modifying it.

Beyond the specific application that is outlined in the paper, an end-to-end learning platform is envisioned that satisfies the industrial requirements of industrial robotic applications. A high degree of generalization is targeted to address a wide variety of different tasks that are typical for manufacturing.

Acknowledgements

This work has been financially supported by AI4DI project. The project has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 826060. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Germany, Austria, Czech Republic, Italy, Latvia, Belgium, Lithuania, France, Greece, Finland, Norway. The authors would like to thank Khedr Kaddour and Luke Pugin for their help and technical support in preparing this publication and Bare Luka Zagar for reviewing the content.

References

[1] Knoll, A., and Walter, F., "Teleported AI," [online], 2020, mediatum.ub.tum.de/attfile/1575022/incoming/2020-Sep/221826.pdf.

[2] David Silver, "Reinforcement Learning: UCL Course," https://deepmind.com/learning-resources/-introduction-reinforcement-learning-david-silver, [retrieved 17 April 2020].

[3] Sutton, R. S., and Barto, A. G., Reinforcement learning. An introduction, 2nd edn., The MIT Press, Cambridge, Massachusetts, 2018.

[4] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D., "Continuous Control with Deep Reinforcement Learning," 10 Sep. 2015, http://arxiv.org/pdf/1509.02971v6, [retrieved 16 November 2020].

[5] Mahmood, A. R., Korenkevych, D., Vasan, G., Ma, W., and Bergstra, J., "Benchmarking Reinforcement Learning Algorithms on Real-World Robots," 2018, http://arxiv.org/pdf/1809.07731v1, [retrieved 10 May 2021].

[6] Levinson, J., Askeland, J., Becker, J., Dolson, J., Held, D., Kammel, S., Kolter, J. Z., Langer, D., Pink, O., Pratt, V., Sokolsky, M., Stanek, G., Stavens, D., Teichman, A., Werling, M., and Thrun, S., "Towards Fully Autonomous Driving: Systems and Algorithms," IEEE Intelligent Vehicles Symposium, 2011, pp. 163–168.

[7] Yahya, A., Li, A., Kalakrishnan, M., Chebotar, Y., and Levine, S., "Collective Robot Reinforcement Learning with Distributed Asynchronous Guided Policy Search," 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2017.

[8] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D., "Human-Level Control Through Deep Reinforcement Learning," Nature; Vol. 518, No. 7540, 2015, pp. 529–533.

[9] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D., "Mastering the Game of Go With Deep Neural Networks and Tree Search," Nature; Vol. 529, No. 7587, 2016, pp. 484–489.

[10] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., and Hassabis, D., "Mastering the Game of Go Without Human Knowledge," Nature; Vol. 550, No. 7676, 2017, pp. 354–359.

[11] Levine, S., Pastor, P., Krizhevsky, A., and Quillen, D., "Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection," 7 Mar. 2016, http://arxiv.org/pdf/1603.02199v4, [retrieved 15 February 2021].

[12] Rusu, A. A., Vecerik, M., Rothörl, T., Heess, N., Pascanu, R., and Hadsell, R., "Sim-to-Real Robot Learning from Pixels with Progressive Nets," 14 Oct. 2016, http://arxiv.org/pdf/1610.04286v2, [retrieved 16 November 2020].

[13] Peng, X. B., Andrychowicz, M., Zaremba, W., and Abbeel, P., "Sim-to-Real Transfer of Robotic Control with Dynamics Randomization," 18 Oct. 2017, http://arxiv.org/pdf/1710.06537v3, [retrieved 16 November 2020].

[14] Meyes, R., Tercan, H., Roggendorf, S., Thiele, T., Büscher, C., Obdenbusch, M., Brecher, C., Jeschke, S., and Meisen, T., "Motion Planning for Industrial Robots using Reinforcement Learning," Procedia CIRP; Vol. 63, 2017, pp. 107–112. doi: 10.1016/j.procir.2017.03.095.

[15] Ota, K., Jha, D. K., Oiki, T., Miura, M., Nammoto, T., Nikovski, D., and Mariyama, T., "Trajectory Optimization for Unknown Constrained Systems using Reinforcement Learning," 2019, http://arxiv.org/pdf/1903.05751v2, [retrieved 10 May 2021].

[16] Akrour, R., Neumann, G., Abdulsamad, H., and Abdolmaleki, A., "Model-Free Trajectory Optimization for Reinforcement Learning," Proceedings of The 33rd International Conference on Machine Learning, Vol. 48, PMLR, New York, New York, USA, 2016, pp. 2961–2970.

[17] Tassa, Y., Erez, T., and Todorov, E., "Synthesis and stabilization of complex behaviors through online trajectory optimization," 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 07/10/2012 - 12/10/2012, pp. 4906–4913.

[18] Zhang, J., Springenberg, J. T., Boedecker, J., and Burgard, W., "Deep reinforcement learning with successor features for navigation across similar environments," 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 24/09/2017 - 28/09/2017, pp. 2371–2378.

[19] Botteghi, N., Sirmacek, B., Mustafa, K. A. A., Poel, M., and Stramigioli, S., "On Reward Shaping for Mobile Robot Navigation: A Reinforcement Learning and SLAM Based Approach," 2020, http://arxiv.org/pdf/2002.04109v1, [retrieved 20 May 2021].

[20] Tai, L., Paolo, G., and Liu, M., "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation," 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 24/09/2017 - 28/09/2017, pp. 31–36.

[21] Yu, J., Su, Y., and Liao, Y., "The Path Planning of Mobile Robot by Neural Networks and Hierarchical Reinforcement Learning," Frontiers in neurorobotics, published online 2 Oct. 2020; Vol. 14, 2020, p. 63.

[22] Kollar, T., and Roy, N., "Trajectory Optimization using Reinforcement Learning for Map Exploration," The International Journal of Robotics Research; Vol. 27, No. 2, 2008, pp. 175–196.

[23] Harald Bayerlein, Paul De Kerret, and David Gesbert, Trajectory Optimization for Autonomous Flying Base Station via Reinforcement Learning, IEEE, Piscataway, NJ, 2018.

[24] "Unity3d," https://unity.com, [retrieved 29 April 2020].

[25] "Nvidia PhysX," https://www.nvidia.com/de-de/drivers/physx/9_16_0318/physx-9-16-0318-driver-de/, [retrieved 29 April 2020].

[26] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O., "Proximal Policy Optimization Algorithms," 20 Jul. 2017, http://arxiv.org/pdf/1707.06347v2, [retrieved 16 November 2020].

[27] Hill, A., Raffin, A., Ernestus, M., Gleave, A., Kanervisto, A., Traore, R., Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., and Wu, Y., "Stable Baselines," GitHub, 2018 GitHub repository.