Boosted deep reinforcement learning method of optimal trajectory planning for a five-axis manipulator

Artur Stefańczyk[1], Patryk Bałazy[2], Paweł Knap[3] and Tymoteusz Turlej[4]

Abstract— **Determining the optimal trajectory of a manipulator is a mathematically complex part of control due to the time-consuming determination of inverse kinematics, the complexity of which increases as the manipulator's degrees of freedom increase. This paper presents the results of training the RL agent with a custom-made reward function built to control the 5DOF manipulator. Unlike the classical approach to determining inverse kinematics, the solution presented in the article is based on the model's output signals. The agent determines the control policy based on the angular position, angular velocities of the manipulator actuators and the feedback that is the reward function. The reward function includes integral indicators of control quality, which allowed the neural network's learning process to be significantly accelerated. The agent was trained to reach the desired point from a fixed and random initial state of the robotic arm in optimal time and efficient energy requirements. In addition, the learning process is based on the environment, which is the manipulator. However, it is not a mathematical model of the manipulator but a three-dimensional representation designed in a CAD environment.**

I. INTRODUCTION

Artificial Intelligence, especially Artificial Neural Network, are used in various engineering fields. However, they are beneficial when it comes to high-dimensional, non-linear optimization problems [1], [2], [3], [4]. For instance, that feature can be used in robotics and mechatronics projects in planning a well-optimized motion for complex robotics mechanisms [5], [6]. One of the most common problems in the robotics control system is efficiently determining trajectories for manipulators with high degrees of freedom. With the increasing diversity of applications for the manipulator and the factors that can be considered when planning trajectories such as obstacle avoidance, the traditional method to solve those problems can be insufficient and time-consuming. Due to that causes, much research is trying to find innovative methods to solve this problem. Many of them use Artificial Intelligence-related algorithms [7], [8], [9]. One with the most promising [10], [11], [12], while very innovative method that propose solution to described problem use Reinforcement Learning (RL) algorithm [13], [14]. This solution replaces a complex control system with an RL agent that can execute chosen actions based on received observations from a virtual environment. In real environment it is a good practice to estimate free space with use of sensors and specialized algorithms [15]. This paper presents the results of the Reinforcement Learning method applied to evaluate the optimal trajectory for a 5-axis manipulator with an agent trained using Deep Deterministic Policy Gradient(DDPG) and Actor-Critic policy simulated in a virtual environment. Deep Reinforcement Learning is a relatively new approach to solving complex, multi-dimensional problems. Its key element is the reward function [16], which is being searched in the training process. Choosing a suitable reward function is extremely challenging as it directly influences later convergence of the whole training, not only on the required time but also on whether it convergences. The reward function should reflect the quality of chosen actions. In this case, it is directly connected to the distance between the manipulator end effector and desired point in the global frame of reference. Successful training should produce such an RL agent that can reach the desired point in space by producing a series of actions from any initial state. In this particular research, actions will represent the value of torque applied to a given joint of the manipulator.

To make training possible, it is necessary to collect feedback data from the virtual environment where the simulation is held. To determine optimal trajectory, it is compulsory to know the angle and angular velocity in each joint of the robotic arm, and those values will be called observation in the other part of this paper. The training process of Reinforcement Learning consists of episodes. Each episode in this particular problem will represent one entire simulation of manipulator movement. Length of single episode dependent on chosen quantity of steps and ending conditions value. In each step, the RL agent will decide which value of actions should be executed, and it will gain new experience. While this approach can lead to very successful results, it requires a lot of computing power, as Reinforcement Learning training usually consists of hundreds or thousands of episodes to reach sufficient policy parameters. As in our case, each episode is an individual simulation. Even with advanced hardware devices, it can take many hours to converge. Another flaw of this method is that even if the training result satisfies the expected presupposition, it does require further validation as Reinforcement Learning is very sensitive to many factors, such as seed for pseudo-random

*This work was supported by Faculty of Mechanical Engineering and Robotics, AGH University of Science and Technology in Krakow, Poland

[1]Artur Stefańczyk is with Faculty of Mechanical Engineering and Robotic, University of Science and Technology in Krakow, Poland. astefanczyk@student.agh.edu.pl

[2]Patryk Bałazy is with Faculty of Mechanical Engineering and Robotic, University of Science and Technology in Krakow, Poland. balazy@student.agh.edu.pl

Paweł Knap is with Faculty of Mechanical Engineering and Robotic, University of Science and Technology in Krakow, Poland. pknap@student.agh.edu.pl

Tymoteusz Turlej is with Faculty of Mechanical Engineering and Robotic, University of Science and Technology in Krakow, Poland. turlej@agh.edu.pl

number generator [17]. Therefore, the training process should be repeated at least a couple of times to justify the results' repeatability. Due to those causes, it takes days to acquire correctly established parameters for RL agents successfully. Another problem related to this field of Machine Learning is a massive amount of hyperparameters that have to be precisely established to get satisfying results. Especially gentle hyperparameters are weights in reward function, which wrongly determined can completely change training results.

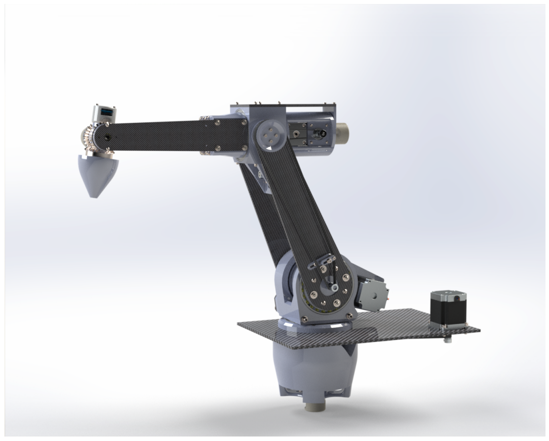


Fig. 1. 5DOF manipulator model

II. MATERIALS AND METHODS

A. Deep Deterministic Policy Gradient

The reinforcement Learning method consists of two primary elements: RL Agent and environment. Relation between those components is visible on Fig 2. The environment in this particular problem represents the manipulator simulation. The RL agent decides on the actions executed inside the environment. To successfully train the RL agent, a particular group of algorithms is called Policy Gradient Methods(PGM). PGM has been created to improve off-policy methods, such as Q-learning. In the off-policy method, action will be executed to maximize state value. The correctly implemented off-policy method can determine which action should be executed in the given state. The weakness of that method is that it can solve only problems with a discrete space of actions. PGM solves this problem, and as a training result, it returns network trained concerning policy.

In Reinforcement Learning, the assumptions are that agent is trained in the environment of a given state s_t, an element of the space of states S, which can be discrete or continuous. In this environment agent executes action at based on policy dependent of value s_t, a_t and parameter θ $\pi(a_t|s_t,\theta)$, which is an element of space of actions A, which also which can be discrete or continuous. As the results of executed action a_t state s_t switch to state s_{t+1}; state s_{t+1} has to be determinate by the action a_t, so it has to fulfill Markov Decision Process requirements. Based on the quality of executed action reward r_{t+1} is being assigned. The goal of an agent is to maximize return from all the states in set S, by training optimal policy function. The process of training policy function can be understood as maximizing some reward function $\xi(\theta)$, which measures the efficiency of the training concerning parameter θ. Searching for the global maximum is possible by calculating the gradient descent. This method can solve problems with discrete and continuous spaces of actions. However, in this research, continuous action space was used. For continuous action, space policy is given as the sample of Gauss distribution:

$$\pi(a_t|s_t,\theta) = a_t \ (\mu(s_t), \sigma^2(s_t)) \tag{1}$$

Mean μ and deviation σ can be represented as functions with s_t as the argument depended from the hyperparameters θ_μ, θ_σ, therefore finding optimal policy function may be regard as finding optimal hyperparameters θ_μ, θ_σ. If the reward function is continuous and differentiable, its gradient can be calculated by using the following equation:

$$\nabla\xi(\theta) = E\pi[\nabla\theta \cdot \ln \pi(a_t|s_t,\theta) Q\pi(s_t,a_t)] \tag{2}$$

The downside of this method is that the training process is complicated as this approach is very sensitive to local minimums. Another flaw is that deviation of cumulative reward is very high throughout the entire process of training.

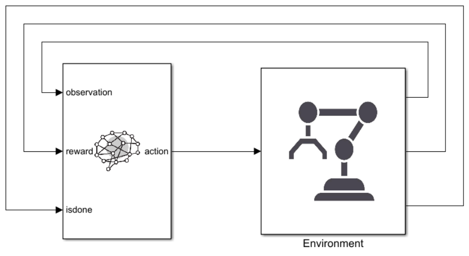


Fig. 2. Diagram show relation between RL agent and environment

B. Actor-Critic method

Algorithm 1 Actor-Critic method in pseudo-code

```
for $step$ in $episode$ do
    sample action a $\sim\pi(a|s,\theta)$

    evaluate estimation of state value
      $\delta = r + \gamma V(s',\theta_v) - V(s,\theta_v)$

    calculate value of a gradient
      $\nabla V(\theta_v) = \gamma_t \delta_\theta V(s,\theta_v)$

    increase the gradient $\theta_v = \theta_v + \alpha V(\theta_v)$

    calculate reward function gradient
      $\xi(\theta) = \gamma_t \delta_\theta ln\pi(a_t|s_t,\theta)$

    s = s'
end for
```

An actor-critic method is a subset of a bigger group of approaches called REINFORCE method. The main idea of this method is to create two neural networks. One of them is Policy Network, which will be referred to as Actor Network in the further part of this paper. This neural network decides which actions should be executed in a given state. The second neural network is called the Critic Network; it is Value Network and is responsible for evaluating decisions made by an Actor. Critic estimate state value $V(s, \theta_v)$ and compare it whit gained reward r and next state value $\gamma V(s', \theta_v)$. Difference δ is written as:

$$\delta = r + \gamma V(s', \theta_v) - V(s, \theta_v) \tag{3}$$

Since predicting future rewards is nearly impossible in the long run, this approach must be limited to the nearest future. This technique is widely known as a bootstrapping method. In every step of an episode, both networks are being trained. The actor-critic method can be represented as pseudo-code, as presented above on Algorithm 1 [18].

C. *Manipulator Simulation*

The robotic arm simulation has been made using Simscape and a multi-body toolbox from MATLAB. The inventor's plug-in, MultibodyLink, has been used to upload required STEP files. The neural network training process also has been held in MATLAB using the Reinforcement Learning toolbox. For the training purpose, other options were considered, such as using Python modules, for example, the well-known Tensorflow or PyTorch. However, Python does not have the well-developed required libraries for robotic or dynamic systems simulations. Therefore MATLAB has been chosen for this research. As this paper focuses on the Deep Reinforcement Learning method, restrictive limits have been set on robotic arm joints, which can be seen in the Table I.

TABLE I
JOINTS LIMITS

Joint number	angular limit [deg]	speed limit [rpm]
Joint 1	0 - 360	260
Joint 2	0 - 120	205
Joint 3	30 - 150	300
Joint 4	0 - 100	390
Joint 5	Not controlled	315

Setting limits allowed to extensively improve the training process due to the much smaller exploration needed during training. At the same time, it provides a sufficient work field for given task execution. Joint 5^{th} is not controlled as it does not directly affect the distance between the desired point and the manipulator's end effector.

All simulations, models, and programs are open-source and available on GitHub repository:
`https://github.com/SzachTech/5DOF_RL`.

III. METODOLOGY

A. *Reward function*

To solve the problem stated in this paper, we had to develop an efficient reward function to make the training process convergent. Our reward function is a sum of three components, two of them are non-linear. The reward function is given with the following equation:

$$\xi(\theta) = \alpha_1 \cdot e^{(-\alpha_2 \cdot \theta^2)} - \alpha_3 \cdot N - \alpha_4 \cdot \Sigma u^2 \tag{4}$$

Where values α_1 α_2 α_3 α_4 are hyperparameters that need to be optimized. In the case of this research θ represents an absolute distance between the end effector and desired point. Value N represents the number of steps required to end the episode. Value N is initially fixed and can only be changed by reaching an end condition. In this case, N will be dependent on the absolute distance θ, as ending condition value will be assigned as true when the end effector reaches the desired point with an insignificant margin of error. Therefore it is beneficial for the RL agent to reach the desired point in the lowest number of steps, consequently in the quickest time. The third component is necessary to minimize the energy required to accomplish a given task. Value u represents the action value in the previous step. The last component minimizes the actuator's effort and prevents jittering.

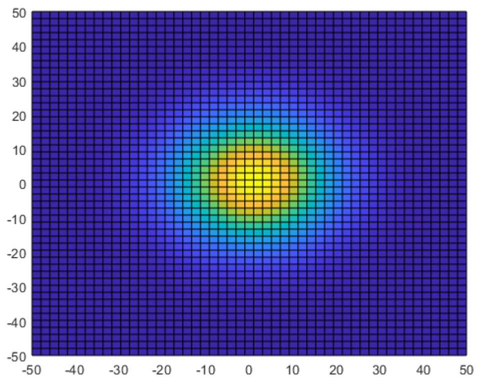


Fig. 3. Heatmap represents reward function value, when desired point is in position $(0, 0)$

B. *Neural Network architecture*

In the Actor-Critic method, there are two kinds of neural networks. The first is the actor-network, also known as Policy Network. The second one – Critic Network, acts as Value Network. The Actor-Network is essentially just Multi-Layers Perceptron (MLP) in our case. On the input layer, normalized observations are given. Then there is a specified quantity of hidden Fully-Connected Layers. ReLu function is used as an activation function. A hyperbolic tangent was used as an activation function on the output layer. The output value from the Actor Network is interpreted as an action executed by the RL agent. Actor-Network architecture is shown on Fig. 4. The Critic Network is more complex as it has two input layers. On the first branch of the Critic Network, observations are given. On the second branch, executed actions. Then both branches are united, and the Output layer is on the common path of both branches. Each one of the paths can be

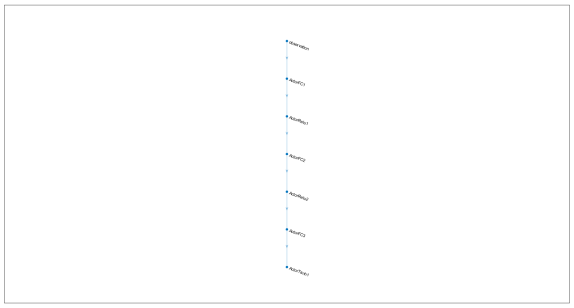


Fig. 4. Actor Neural Network architecture

considered as an individual Multi-Layers Perceptron. Critic Network architecture is shown in Fig. 5.

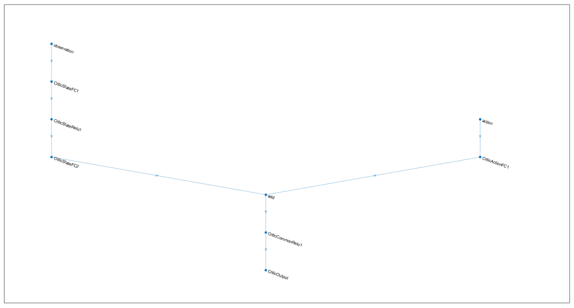


Fig. 5. Critic Neural Network architecture

IV. NEURAL NETWORK TRAINING

Two problems have been posed: training the RL agent to reach the desired point from a fixed initial state and training it in such a manner that it can reach the desired point from a randomized initial state. Both of those problems have been covered. However, firstly it is necessary to determine reward function hyperparameter values, which can be seen in equation (4). Their selected values can be found in Tab. II. Training of RL agent has been conducted on i7 Intel Core parallel on eight workers.

TABLE II
REWARD FUNCTION HYPERPARAMETERS

α_1	20
α_2	$3.328 \cdot 10^{-3}$
α_3	800
α_4	$3.125 \cdot 10^{-3}$

A. Fixed initial state

Initially, the RL agent has been trained to find the optimal trajectory from the initial set-up state to the stationary desired point. The learning rate for Actor has been adopted to 10^{-4} and for the Critic 10^{-3}. Each episode's time has been set to 2 with a fixed number of steps per episode equal to 80. The end condition was met when the manipulator's end effector was at a distance of 0.1 inch from the desired point. After 1482 episodes, it has successfully convergent with an average reward 254.174 and elapsed time was $2789s$. The convergence curve can be seen in figure 6.

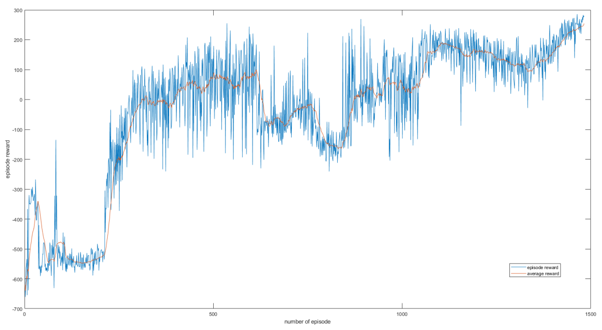
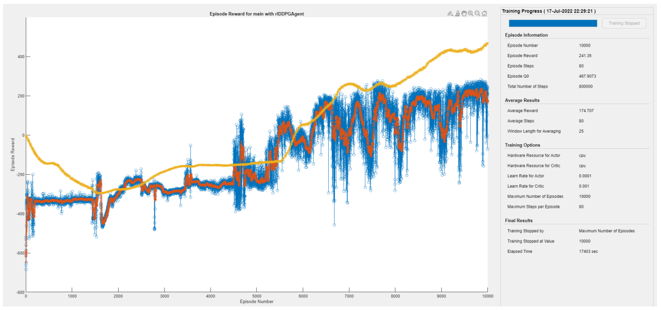


Fig. 6. Convergence line, where average is calculated from 25 episodes

B. Randomize initial state

All the parameters stayed unchanged. However, initial conditions have been randomly selected from a possible joint angle, with respect to table I. Each episode has been initialized from a random state. After 10000 episodes, it successfully convergent and ended the training process with an average reward 174. Elapsed time was $17403s$. The convergence curve can be seen in Fig. 7.



Training Progress (17-Jul-2022 22:29:21)

Episode Information

Episode Number	10000
Episode Reward	241.35
Episode Steps	80
Episode Q0	467.9073
Total Number of Steps	800000

Average Results

Average Reward	174.707
Average Steps	80
Window Length for Averaging	25

Training Options

Hardware Resource for Actor	cpu
Hardware Resource for Critic	cpu
Learn Rate for Actor	0.0001
Learn Rate for Critic	0.001
Maximum Number of Episodes	10000
Maximum Steps per Episode	80

Final Results

Training Stopped by	Maximum Number of Episodes
Training Stopped at Value	10000
Elapsed Time	17403 sec

Fig. 7. Convergence line, where average is calculated from 25 episodes

V. RESULTS

Plot of trajectory from fixed initial conditions can be seen on Fig. 8.

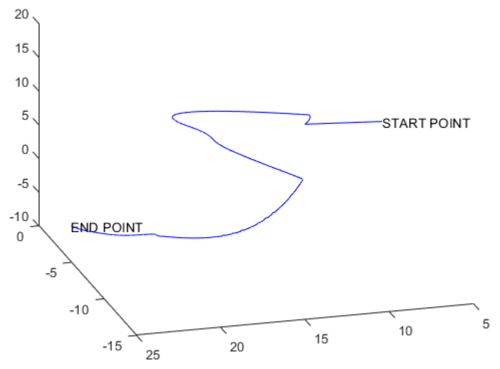


Fig. 8. Computed trajectory from fixed initial state.

Plot of trajectory from randomized initial conditions can be seen on Fig. 9.

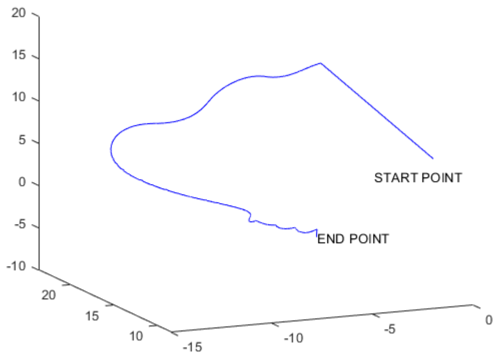


Fig. 9. Computed trajectory from random initial state.

VI. CONCLUSIONS

This paper uses Deep Reinforcement Learning to raise the problem of sufficient computing trajectory of 5 degrees of freedom manipulator. This problem has been overcome using the Actor-Critic method and Deep Deterministic Policy Gradient. Two cases were considered: an environment with a fixed initial state and an environment with a randomized initial state. Both of those cases have been successfully solved in relatively short computing time. The obtained results fulfil the efficiency requirements, and the end effector reached the desired point with a given margin of error.

The proposed method using the representation of the environment as a three-dimensional CAD model accelerated the learning process of the algorithm. The steps taken guarantee very similar if not identical performance on a real object. Such an approach shortens the dynamic system modelling and provides much more specific results. The model part can be modified, and the rest of the programs can be reused for training RL models for a completely different manipulator.

In future research, we plan to test developed algorithms in real-life scenarios. The manipulator who will allow performing the tests is currently under construction. Test results will allow proving the results. Furthermore, the solution can be extended to a multi-agent method, i.e. train the network in a way that allows a pair of manipulators to work in a standard working field. This will also be the focus of our future research.

The materials provided in the GitHub repository allow for easy reuse and verification of the research results.

ACKNOWLEDGMENT

This project was possible due to the second edition of the IDUB grant support, which was created to support Polish students' research groups in participation in international events. This project has been developed with New-Tech students' research group located in Faculty of Mechanical Engineering and Robotic, AGH University of Science and Technology in Krakow, Poland.

REFERENCES

[1] Oludare Isaac Abiodun, Aman Jantan, Abiodun Esther Omolara, Kemi Victoria Dada, Nachaat AbdElatif Mohamed, and Humaira Arshad. State-of-the-art in artificial neural network applications: A survey. *Heliyon*, 4(11):e00938, 2018.

[2] Yu-chen Wu and Jun-wen Feng. Development and application of artificial neural network. *Wireless Personal Communications*, 102(2):1645–1656, 2018.

[3] Ahmad Azharuddin Azhari Mohd Amiruddin, Haslinda Zabiri, Syed Ali Ammar Taqvi, and Lemma Dendena Tufa. Neural network applications in fault diagnosis and detection: an overview of implementations in engineering-related systems. *Neural Computing and Applications*, 32(2):447–472, 2020.

[4] Mohaiminul Islam, Guorong Chen, and Shangzhu Jin. An overview of neural network. *American Journal of Neural Networks and Applications*, 5(1):7–11, 2019.

[5] Jorge Ribeiro, Rui Lima, Tiago Eckhardt, and Sara Paiva. Robotic process automation and artificial intelligence in industry 4.0–a literature review. *Procedia Computer Science*, 181:51–58, 2021.

[6] SWETHA Danthala, SEERAMSRINIVASA Rao, KASIPRASAD Mannepalli, and Dhantala Shilpa. Robotic manipulator control by using machine learning algorithms: A review. *International Journal of Mechanical and Production Engineering Research and Development*, 8(5):305–310, 2018.

[7] Ahmed A Hassan, Mohamed El-Habrouk, and Samir Deghedie. Inverse kinematics of redundant manipulators formulated as quadratic programming optimization problem solved using recurrent neural networks: A review. *Robotica*, 38(8):1495–1512, 2020.

[8] Xuan Khoat Ngo and Manh Dung Ngo. A review of artificial intelligence algorithms used for robotic manipulator.

[9] Rongrong Liu, Florent Nageotte, Philippe Zanne, Michel de Mathelin, and Birgitta Dresp-Langley. Deep reinforcement learning for the control of robotic manipulation: a focussed mini-review. *Robotics*, 10(1):22, 2021.

[10] Timothy Rupprecht and Yanzhi Wang. A survey for deep reinforcement learning in markovian cyber-physical systems: Common problems and solutions. *Neural Networks*, 2022.

[11] Farzin Piltan, SH Tayebi Haghighi, N Sulaiman, Iman Nazari, and Sobhan Siamak. Artificial control of puma robot manipulator: A-review of fuzzy inference engine and application to classical controller. *International Journal of Robotics and Automation*, 2(5):401–425, 2011.

[12] Serkan Dereli and Raşit Köker. A meta-heuristic proposal for inverse kinematics solution of 7-dof serial robotic manipulator: quantum behaved particle swarm algorithm. *Artificial Intelligence Review*, 53(2):949–964, 2020.

[13] Jung-Jun Park, Ji-Hun Kim, and Jae-Bok Song. Path planning for a robot manipulator based on probabilistic roadmap and reinforcement learning. *International Journal of Control, Automation, and Systems*, 5(6):674–680, 2007.

[14] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. Scalable deep reinforcement learning for vision-based robotic manipulation. In *Conference on Robot Learning*, pages 651–673. PMLR, 2018.

[15] Marek Szlachetka, Dariusz Borkowski, and Jaroslaw Was. Stationary environment models for advanced driver assistance systems. In *2020 Signal Processing: Algorithms, Architectures, Arrangements, and Applications (SPA)*, pages 116–121, 2020.

[16] Andrew Y Ng, Stuart Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, page 2, 2000.

[17] Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C Courville, and Marc Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in neural information processing systems*, 34:29304–29320, 2021.

[18] Antonio Gulli, Amita Kapoor, and Sujit Pal. *Deep learning with TensorFlow 2 and Keras: regression, ConvNets, GANs, RNNs, NLP, and more with TensorFlow 2 and the Keras API*. Packt Publishing Ltd, 2019.