Available online at www.sciencedirect.com

ScienceDirect

Procedia CIRP 109 (2022) 287–292

www.elsevier.com/locate/procedia

32nd CIRP Design Conference

Overcoming the Sim-to-Real Gap in Autonomous Robots

Pascalis Trentsios[a], Mario Wolf*[a], Detlef Gerhard[a]

[a]*Digital Engineering Chair, Universitätsstrasse 150, 44801 Bochum, Germany*

* Corresponding author. Tel.: +49-234-32-222115 ; fax: +49-234-32-14443. *E-mail address:* mario.wolf@rub.de

Abstract

When designing and planning an autonomous, adaptive system utilizing the benefits of Artificial Intelligence (AI), both the specific field of AI and the desired use case must be addressed to achieve a reliable system. In the field of autonomous robotics, the authors focus on the aspect of training a robot not in reality, where training is reliant on real-time and the amount of hardware systems performing the training, but in a virtual environment, which can be parallelized and customized individually. In this paper, a functional simulation of a real robot was developed and trained using reinforcement learning in a virtual environment to successfully complete a specified task. The trained model was then transferred to the physical robot, which itself was able to complete the specified task in the real world, without the need of previous real-world training. Due to extensive observation and analysis of the physical system's characteristics, the presented direct transfer approach does not rely on domain randomization, which is usually applied in this field.

Keywords: reinforcement learning; machine learning; sim-to-real; simulation; Unity; robotics;

1. Introduction

In recent years, a megatrend in current Machine Learning (ML) research has emerged. Instead of programming the execution of a specified task, the ML system also referred to as agent, learns how to perform a task successfully by exploration and exploitation of its environment and targeted rewards or penalties for desired or undesired behavior given by an interpreter [11]. With such an approach, a product designer does not need a holistic understanding on how the product should execute its task, but rather needs to create a rewarding system based on goals. Through this, ML agents can be able to outperform classic problem-solving solutions. While ML applications require an enormous amount of qualitative data for sufficient results, gathering such data in the real world can act as a bottleneck and hinder development. The use of digital twins (DT) is widely established practice in industry, and with increasingly potent simulations the use of DT is becoming more attractive and efficient. Digital Twins can therefore act as an enabler for ML applications, with ML agents being trained in the simulation part of a DT and then transferred to the real twin in reality (sim-to-real). Autonomous robotics is one potential use case for such an approach, which is presented and discussed in this paper.

2. Related Work

Depending on the task, the use of a real environment for solving the control problem based on learning methods is not feasible, because reality comes with many limitations a virtual environment does not have. For such an approach, an interpreter is needed, which would need to reward or penalize the agent for its then-current behavior in real-time. Also, the training environment must be reset after a failed attempt and randomized to a certain degree to make the trained model more robust. The utilization of simulated, virtual environments can solve these and other problems by parallelization of the training environment and therefore accelerated training time to reach the number of required training steps faster.

The sim-to-real approach encompasses the transfer of the learned behavior of an agent trained in simulation to the real world and physical system [13]. Usually, the resulting behavior shows deviations between the simulation and real world, due to an imperfect representation of the real world in the simulation, which is referred to as sim-to-real gap [18].

2212-8271
10.1016/j.procir.2022.05.251

A common practice is to use the domain randomization approach [16, 2, 4] to overcome the sim-to-real gap, by randomizing selected, major parameters in the simulated domain. For example, the simulated physics behavior or the visual observation.

3. Concept

3.1. Methodology

The presented approach aims to overcome the sim-to-real gap by analyzing the characteristics of a real robot in detail and evaluate various approaches to create a simulation with sufficient precision in a virtual environment. This practice is commonly referred to as zero-shot or direct transfer [17, 5]. This approach should require less training time compared to the domain randomization approach while providing better results for the addressed physical domain. The method to the analysis of the physical domain, how it can be simulated precisely enough in the simulation and the presentation of relevant simulation parameters are also considered. To achieve this goal, the process shown in Fig. 1 is the basis to establish a detailed understanding of metrics of the real world, the development of a virtual environment and the transfer into reality.

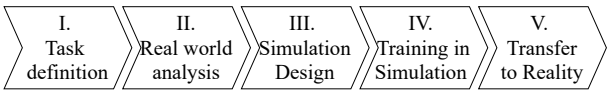

Fig. 1. Process for the presented approach

The first step is to define the task that the system must be able to fulfill. In the second step, the real components of the system are analyzed and recorded with sufficient accuracy, which is used in the third step to simulate the interaction of system components and the environment with sufficient accuracy including a virtual representation of the task with clear indicators of success. In the fourth step, the simulated system or respectively its machine learning agent starts its training in the simulation until success rates are satisfactory. Finally, in the fifth step, the direct transfer of the learned behavior to the physical system is performed and validated in reality.

3.2. Task Definition and Limiting Factors

The practical relevance and final goal for the presented approach are to be able to train driverless transport systems (DTS) in virtual environments based on digital factory (DF) or Building Information Modeling (BIM) derived virtual environments. The transport system should be able to pick and place various objects from one point to another, and also be able to avoid obstacles and find objects that are not in direct view. To reach this goal, a proof of concept implementation is conducted on a reduced scale, both in size of the participating robot and the complexity of the transportation task. This means, that the current task acts as a foundation level difficulty of the latter to come complex task. In this case, the robot has to simply find and reach a distinct object, which is randomly placed on a plane, a random distance away from the robot. The localization of the object acts as early proof of concept with the other tasks of a complete delivery process, like pick-up, transport, and drop-off not yet being considered. A modified mBot from the manufacturer MakeBlock serves as a demonstrator for the future DTs, as shown in Fig. 2 (left side).

The modified mBot is a differential wheeled robot, that has nonholonomic constraints, with two active wheels each driven by one DC motor and a third low-friction passive wheel. The motors are controlled by the mBot mCore (d), which in essence is a ATmega328 based board. The base mBot does not offer a camera or other sensors offering communication capabilities needed. Therefore, a Raspberry Pi Zero (a), a Pi HQ Camera (c) with a 6 mm wide angle lens and an additional 5000 mAh battery (c), were added to the mBot, resulting in the operational variant i).

3.3. Real World Measurements and Observations

In the following, characteristics of both the real robot and the real environment are documented in order to transfer them to the virtual environment.

The power output voltage of the mBot battery directly connects to the two outputs of the DC motors, which can be controlled in discrete steps. One step corresponding to 0 V output, and the other 500 steps are split into negative and positive output voltage, ascending from the minimum possible absolute voltage to the maximum possible appliable voltage. The maximum rpm without load on the motors is 200 rpm ± 10% for 6 V, as specified by the manufacturer. With a 3.7 V battery, attached wheel assembly but without load on the robot, 120 rpm were measured for one running motor. One can expect a further drop in performance during higher load while driving both motors in parallel.

When maximum voltage output is demanded, the measured voltage on the motor side fluctuated around 3,9 V, which is higher than the 3,7 V rated output of the mBot battery. This is due to the mBot's battery being charged through the USB connection with the Pi by the additional battery pack. It seems that the additional battery pack keeps the mBot battery voltage and thereby the output voltage more stable during a load scenario. While running, the mCore battery indicator showed a fully charged main battery.

The weight of each component of the robot, each assembly and sub-assembly were measured. The robot's dimensions were

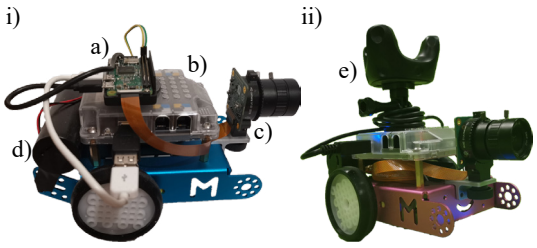

Fig. 2. Robot system used i) operational version, ii) measurement version

gathered from official data sheets and CAD-Models and were verified with suitable measurement methods on the real robot.

[8] exemplifies and describes the driving behavior of differential drive robots mathematically. However, to further understand and be able to compare the real robot to the simulated one, eight different hypothetical driving maneuvers with each combination between full throttle backwards, neutral and full throttle forwards on both motors are evaluated. The Steam VR tracking system, which has an accuracy of around 4 mm [1], was used for recording the real robot's movement. The position, the rotation in Euler angles and Quaternions, and the time passed since startup were measured with a sampling rate of 90 Hz. The Vive tracker (e) version 3.0 used for recording the movement was attached on the mCore cover of the robot system, resulting in the configuration as seen in Fig. 2, (right side). To compensate for the weight of the Vive tracker, and because the corresponding components do not play a role during the recording, the Pi Zero was temporarily removed. Every maneuver took 10 seconds and was repeated 10 times to compensate for outliers.

The friction coefficient values of the rubberized wheels and the vinyl floor environment were considered relevant, but hard to record without special equipment, so they were taken from a tribology manual[3].

For the recreation of the CV-aspect of the robot, the camera's and lenses' specifications like aspect ratio, frame rate and angle of view as well as the lighting conditions of the environment are relevant. The specifications of the manufacturers were adopted. Additionally, a calibration board with a checkerboard pattern and a gray card was used for measurement. The calibration board was photographed with the robot system's camera from various positions with various camera resolutions. The pixel distance for each checkerboard edge was noted, used for later calibration of the simulated camera. The photographs made of the gray card were used for later creation of the simulated lighting conditions.

It was considered to simulate the real environment with 3D-Models and corresponding textures. However, due to simplicity and with the focus being on the simulation of the robot system in this prototype, a 360° spherical image of the real environment was taken and used for later simulation.

4. Implementation

The virtual environment is build using Unity v2021.2, with Unity's object-oriented 3D-Physics, which itself is based on Nvidia's PhysX Engine SDK v4.1. The approach is based on observations, and best practice guides and documentations provided by Nvidia [9] and Unity [12]. To create a simulation of the robot system and its environment, the authors focus on selected properties. The design and development relies on the prior mentioned measurements and observations in the real world. The key approach is to compare the real measurements and the simulated behavior for various aspects, creating a virtual environment that simulates the reality precisely. Therefore, the simulation was created in an iterative process where simulated and real behavior were constantly compared and adjusted to narrow down the sim-2-real gap. Overall, different baselines were created to simulate the environment and the real robot concerning the following, different components.

Physical Body. While in Unity, meshes can be imported as FBX, the meshes themselves are visually rendered but do not have physical properties. To affect an object with the physic engine, there are two different types of virtual "physical" body components that can be selected. One being rigid bodies, the other being articulation bodies. Either one can be applied together with a collider component. According to Unity, the articulation bodies are specifically designed for a more realistic simulation approach [15, 14]. Another difference is that by using the articulation bodies, the bodies of the articulation are automatically joined to each other, while for rigid bodies one must use additional joint components to combine bodies.

Collider Geometry. The colliders from which one can choose are the primitive geometries *spheres*, *capsules*, *boxes*, and *planes*, as well as *convex mesh colliders* which are, according to the Nvidia documentation [9], limited to a total of 255 polygonal faces and vertices. A special form of collider that can be used inside Unity, which is not part of the Nvidia PhysX SDK, is the *wheel collider*. The *wheel collider* has a two-dimensional circular form with a three-dimensional sphere on the contact point to the ground. The *wheel collider* however requires a *rigid body* and does not function in combination with an *articulation body*. Primitive colliders can be used to achieve less processor intensive calculation.

CDM (Collision Detection Mode). There are different collision detection modes available in Unity, each with different mathematical basics on how collisions between two physics objects are detected and calculated.

Friction. A collider component in Unity can be given a physics material, which describes the dynamic and static friction as well as the bounciness and how those coefficients values are calculated if two colliders interact with each other.

Geometric scale. The scale unit in Unity, roughly translates to meters. Therefore, it is feasible to use the real-world scale of the robot in the simulated world. However, it was observed, that the quality of the simulations is very sensitive to the scale factor. The real-world scale of the robot lead to unintended and inaccurate physical behavior of the robot system in the virtual world.

Weight scale. The weight scale unit in Unity is kg. Similar to the geometric scale, one can use the real-world weight of the robot. Testing various virtual weights, no significant variations were observed if the weight distributions remain the same. For higher weights, the force levels and break points of the used joints need to be scaled accordingly.

The select specifics are only a fraction of the possible things to simulate. There are infinite ways to measure the reality as well as to simulate it. The simulation model used by Unity and

the Nvidia PhysX engine focuses on performance and stability of simulation, and therefore does not necessarily present a close approximation of real-world physics. Some specific values used in the simulation might even be a multiple of the one measured in the real world, but would result in a more realistic simulation. In practice, it is only necessary to simulate the reality sufficiently accurate to achieve a closing of the sim-2-real gap.

4.1. Virtual Actuators

To create the simulation of the robot system and the object to be collected, matching 3D-models in FBX format were imported into the Unity engine. A functional assembly of the robot was made inside Unity using articulation bodies. The physical bodies of selected individual robot components were simulated with multiple primitive collider objects for a more realistic weight distribution, and set as a single root body of the assembly. Approximations for the wheels were created with a primitive sphere and capsule colliders as well as with a mesh collider with: 8, 16, 32, 64 and 84 polygonal faces. Observations showed, that the spherical wheels match the real driving behavior best. Due to a high friction coefficient for the wheels, no considerable slipping of the wheels was observed. The used physical bodies indicated by the colliders outlined in green are shown in Fig. 3 on the lower right side.

4.2. Virtual Sensors

The camera was simulated with Unity's camera component. The physical camera option was used, with the application of the optical specification of the real Pi Cam, considering sensor size, focal length and the real measurements done with the calibration board. The camera resolution was set to 64×64×RGB pixels. The virtual camera's image is shown in Fig. 3 on the upper right side.

4.3. Environment

The 360° image of the real environment is applied to a capsule geometry When getting closer to the capsule's surface, the 360° image gets narrower. In simulation, there was one directional light source to replicate the real one, with soft shadows and minor reflections. Reflections were based on a skybox with the 360° image of the room. A plane collider was used as floor, with an associated physics material. To limit the environment space, wall borders were added ad the edge of the 360° image.

The simulation including the virtual environment, the robot and the object is shown in Fig. 3 on the left side.

4.4. Training in the Simulation

The robot, a.k.a. the agent, is trained using reinforcement learning, which allows the fulfillment of complex tasks without the need for a predetermined mathematical model for the Markov decision problem (MDP) [7]. For the training of the agent Unity's ML-Agents Toolkit [6] version *Release 18*, with

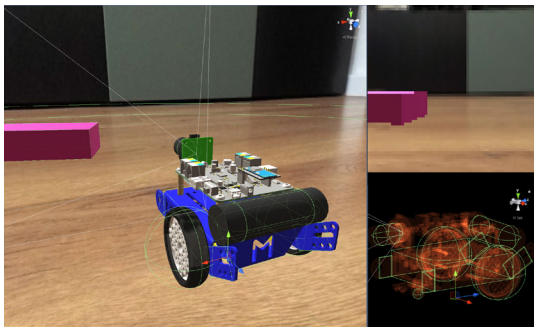

Fig. 3. Virtual environment with robot and desired object (left), virtual camera input (upper right), robot with the colliders, green lines (lower right)

Python package version *0.27.0* and Unity package version *2.1.0* was used. The ML-Agents Toolkit includes versatile tools for the creation of custom machine learning scenarios, various algorithms are provided, from which one can choose as wells as a brought set of pre-defined API and the possibility to combine the given solutions thus custom approaches can be created.

The Proximal Policy Optimization algorithm [10] with the simple Convolutional Neural Network function for the visual observation was used for training. The agents were trained with a decision request of 2.5 Hz. To reduce the total training time, training was done with 16 instances of the agents being trained simultaneously, each running in custom environments and going through individual training episodes. A training episode starts with the agent spawning in the origin of its associated environment, using the RGB camera feed as input tensor and the control of each wheel's rpm as continuous output action. The 360° images of each environment would be randomly rotated with each new episode and object collected for a more robust learning.

To enforce the learning for the performance of the specified task, the agent would need to be rewarded and penalized according to its current behavior. The task completes when the desired object has been touched by the agent. For every step made, the interpreter therefore observes if the agent's collider has touched the desired object or the environment boundaries. With each beginning episode, a timer resets with every new episode or when the agent touches an object.

A reward is given for each object touched, for the percentage of time remaining, and for a successful run. A successful run requires the agent to touch all the objects without a single timeout. Penalties are applied when the environment borders are touched or a timeout occurs. In either way, the current episode will end.

The curriculum learning approach speeds up the training time. With this approach, the task is split up into multiple lessons with increasing difficulty, where the first level should provide an early reward for the desired behavior. The task consists of 10 curriculum lessons. The curriculum would switch to a higher lesson, when 420 Episodes of the current lesson reached a mean reward of at least 0.9. For the first 6 lessons only one object spawns per episode, resulting in a theoretical maximum reward reachable of 3, however due to considerable drive time towards the desired object, the practical maximum

reachable reward is less according to the necessary drive time. For lesson 7, the object would respawn a second time after being touched and for lesson 8 to 10, the object would respawn 10 times in total, respawning to a new random position each time. In the first curriculum lesson, the desired object would spawn close to and in view of the agent, making it much more likely to touch the object, resulting in early rewards given for behavior that results in touching the object. With every lesson, the range in which the object could spawn would increase significantly to the previous lesson, with the agents always spawning in the origin of the environment. The y-axis rotation of the desired objects was randomized with each spawn.

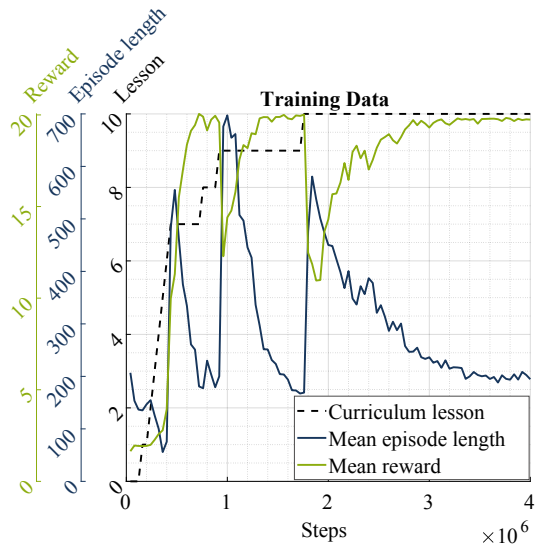

Fig. 4. Training results

The training results in Fig. 4 display the mean reward per agent and episode, the mean episode length per agent, and the current curriculum lesson over the total steps taken. Data logs are taken every 40,000 training steps. The 4 million total training steps taken for the training of the neural network correspond to a total training time of 3 h 14 min 55 s. The thresholds for the first 7 curriculum lessons are met relatively fast in the first 440,000 steps made. While for lesson 9 alone, over 700,000 steps are needed to reach a sufficiently high mean reward to switch to lesson 10. In lesson 9 and 10 the distance to the desired object can be that high, that it might only be perceived by the camera as a few single pixels, making the detection and localization much harder. After 3.5 million steps, the reward function as well as the episode time seem to not further improve but stabilize around 19.8 for the mean reward and 200 decision requests for the mean episode length.

4.5. Transfer to Reality

With the training completed, the execution of the trained neural network model, also referred to as inferencing, can be done inside Unity with the ML-Agents toolkit. For the transfer from the simulation to the reality, the camera feed of the real robot was streamed into Unity as an M-JPEG live stream and was used as input for the trained neural network model. The average latency measured was 400ms. The continuous actions space in the simulation being in a range from [-1, 1] was converted to a range with which the robot's DC motors are controlled. The outputs were streamed over TCP to the Raspberry Pi, which redirects the commands over serial bus to the connected mBot, controlling the DC motors accordingly.

4.6. Evaluation

Fig. 5 shows selected results of three different 64×64×RGB camera inputs, the result of one of the 16 outputs of the first convolutional operator, the associated output of the LeakyReLU activation operator and the resulting output action for each motor. High activation values are reached where the desired object is located in the input image, represented by a darker red color. For the first column, with no object located in the camera view no major activation can be seen, resulting in an output of M1 = 1, M2 = -1, which corresponds to the robot system turning on spot clockwise. With this behavior, the robot effectively searches all the environment for the object. For the input image of the second column, where half the object can be seen on the left side, the output actions consequences in a slight left turn, towards the object. With the object located central, as seen in the third column, the robot would drive forward. The robot system is therefore able to fulfill the specified task, with only simulated data and the direct transfer approach.

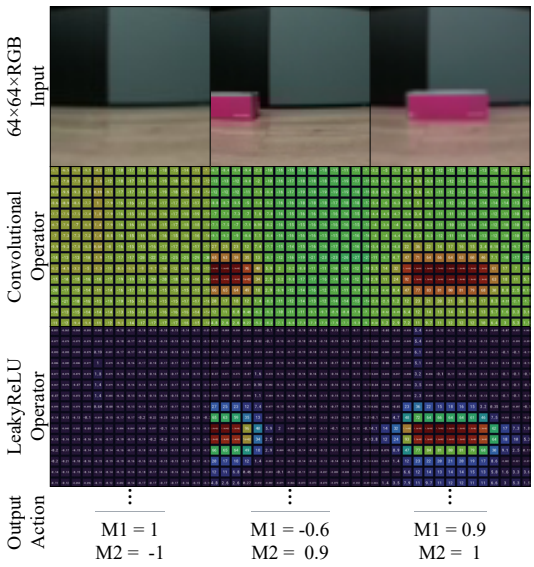

Fig. 5. Three different RGB inputs and the resulting output actions

The robot with its one input sensor and only two action vectors does not have many degrees of freedom, and therefore there is not much space for deviation on how the specified task can be performed. So even a physic-wise poorly simulated robot should take the same actions, driving in expedient maneuvers to reach the desired object. This assumption was tested by handicapping one of the motor output action to only provide 50% of the actual output force. The robot movement was overshooting

and in general not as sufficient as before, but still was able to reach the desired object in time.

Even though many parameters were measured beforehand, some may not be considered as relevant. One of those parameters was the frame rate of the camera. Due to fast rotations around its own axis, the robot moves that fast, that the input image is significantly blurry, making the detection harder, especially when the object is further away from the robot. This can be addressed in various ways, the frame rate can be set higher, the decision requests per second could be increased, or the motion blur effect can be simulated and the model retrained. Since, Unity is capable of simulating motion blur and other camera specific effects. The already trained model could be used and optimized to cope with the motion blur problematic. Retraining the whole model would not be necessary, since the motion blur could be added in simulation as an additional curriculum lesson.

5. Conclusion & Outlook

The presented direct transfer sim-to-real approach was successfully tested with a simple localization and movement problem. Considering that the training took already 4 million steps for the direct transfer approach, the resulting training time for the same task using domain randomization would be considerable higher. Then again, the direct transfer approach is much more prone to fail, when domain shift happens or the sim-to-real gap was greater for example, due to poor simulation parameters. But even in those cases, retraining of the model could be considered. Therefore, the lack of robustness can be specifically addressed by adjusting the virtual environment accordingly and continue the training process without the need to restart the whole training. As virtual environments are generally more easily adapted than real environments or can even be procedurally created, such changes are neither cost- nor time-intensive.

For future work, it should be analyzed how much deviation between the simulated and real domain is tolerable for a desired task to be performed successfully. At least for the presented approach, some measurements and observations collected proved to be irrelevant or had only a negligible influence on the performance. A robust training process in simulation would therefore be possible with even less or less reliable information.

In general, the stated approach was set as an early concept and will be further researched, with the goal to elaborate a deeper understanding of the technique and a general methodology which can be applied to varying use cases. Additionally, the authors plan to apply the lessons learned on a quadruped robot, where the physical adaptation of the real robot should have more impact on the actual trained behavior.

References

[1] Bauer, P., Lienhart, W., Jost, S., 2021. Accuracy Investigation of the Pose Determination of a VR System. Sensors 21, 1622. URL: https://www.mdpi.com/1424-8220/21/5/1622, doi:10.3390/s21051622.

[2] Chu, Y.J.R., Wei, T.H., Huang, J.B., Chen, Y.H., Wu, I.C., 2020. Sim-To-Real Transfer for Miniature Autonomous Car Racing. arXiv:2011.05617 [cs] URL: http://arxiv.org/abs/2011.05617. arXiv: 2011.05617.

[3] Czichos, H., Habig, K.H. (Eds.), 2015. Tribologie-Handbuch: Tribometrie, Tribomaterialien, Tribotechnik. 4., vollst. überarb. und erw. aufl ed., Springer Vieweg, Wiesbaden. doi:10.1007/978-3-8348-2236-9.

[4] Ding, Z., Lepora, N.F., Johns, E., 2020. Sim-to-Real Transfer for Optical Tactile Sensing. arXiv:2004.00136 [cs] URL: http://arxiv.org/abs/2004.00136. arXiv: 2004.00136.

[5] James, S., Johns, E., 2016. 3D Simulation for Robot Arm Control with Deep Q-Learning. arXiv:1609.03759 [cs] URL: http://arxiv.org/abs/1609.03759. arXiv: 1609.03759.

[6] Juliani, A., Berges, V.P., Teng, E., Cohen, A., Harper, J., Elion, C., Goy, C., Gao, Y., Henry, H., Mattar, M., Lange, D., 2020. Unity: A General Platform for Intelligent Agents. arXiv:1809.02627 [cs, stat] URL: http://arxiv.org/abs/1809.02627. arXiv: 1809.02627.

[7] Kaelbling, L.P., Littman, M.L., Moore, A.W., 1996. Reinforcement Learning: A Survey. arXiv:cs/9605103 URL: http://arxiv.org/abs/cs/9605103. arXiv: cs/9605103.

[8] Mathew, R., Hiremath, S.S., 2016. Trajectory Tracking and Control of Differential Drive Robot for Predefined Regular Geometrical Path. Procedia Technology 25, 1273–1280. URL: https://linkinghub.elsevier.com/retrieve/pii/S2212017316305758, doi:10.1016/j.protcy.2016.08.221.

[9] NVIDIA Corporation, 2021. NVIDIA PhysX SDK 4.1 Documentation. URL: https://gameworksdocs.nvidia.com/PhysX/4.1/documentation/physxguide/Index.html.

[10] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O., 2017. Proximal Policy Optimization Algorithms. arXiv:1707.06347 [cs] URL: http://arxiv.org/abs/1707.06347. arXiv: 1707.06347.

[11] Tokic, M., 2013. Reinforcement Learning mit adaptiver Steuerung von Exploration und Exploitation. Dissertation. Universität Ulm. URL: https://oparu.uni-ulm.de/xmlui/handle/123456789/2544, doi:10.18725/OPARU-2517. accepted: 2016-03-15T09:03:57Z.

[12] Unity Technologies, 2021. Unity - Manual: Unity User Manual 2021.2. URL: https://docs.unity3d.com/2021.2/Documentation/Manual/index.html.

[13] Vrabič, R., Škulj, G., Malus, A., Kozjek, D., Selak, L., Bračun, D., Podržaj, P., 2021. An architecture for sim-to-real and real-to-sim experimentation in robotic systems. Procedia CIRP 104, 336–341. URL: https://linkinghub.elsevier.com/retrieve/pii/S2212827121009550, doi:10.1016/j.procir.2021.11.057.

[14] Yakovlev, A., Greene, C., 2020. Prototype your industrial designs using Unity's new ArticulationBody feature | Unity Blog. URL: https://blog.unity.com/manufacturing/use-articulation-bodies-to-easily-prototype-industrial-designs-with-realistic-motion.

[15] Yakovlev, A., Navarro, A., 2021. Simulate robots with more realism: What's new in physics for Unity 2021.2 beta | Unity Blog. URL: https://blog.unity.com/technology/simulate-robots-with-more-realism-whats-new-in-physics-for-unity-20212-beta.

[16] Yan, M., Sun, Q., Frosio, I., Tyree, S., Kautz, J., 2020. How to Close Sim-Real Gap? Transfer with Segmentation! arXiv:2005.07695 [cs] URL: http://arxiv.org/abs/2005.07695. arXiv: 2005.07695.

[17] Zhao, S., Li, B., Reed, C., Xu, P., Keutzer, K., 2020. Multi-source Domain Adaptation in the Deep Learning Era: A Systematic Survey. arXiv:2002.12169 [cs, stat] URL: http://arxiv.org/abs/2002.12169. arXiv: 2002.12169.

[18] Zhao, W., Queralta, J.P., Westerlund, T., 2021. Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: a Survey. arXiv:2009.13303 [cs] URL: http://arxiv.org/abs/2009.13303, doi:10.1109/SSCI47803.2020.9308468.. arXiv: 2009.13303.