Reinforcement Learning of Robot Behavior based on a Digital Twin

Tobias Hassel[a] and Oliver Hofmann[b]
Faculty of Electrical Engineering, Technical University Nuremberg, Keßlerplatz 12, Nuremberg, Germany

Keywords: Reinforcement Learning, Digital Twin, Unity.

Abstract: A reinforcement learning approach using a physical robot is cumbersome and expensive. Repetitive execution of actions in order to learn from success and failure requires time and money. In addition, misbehaviour of the robot may also damage or destroy the test bed. Therefore, a digital twin of a physical robot has been used in our research work to train a model within the simulation environment Unity. Later on, the trained model has been transferred to a real-world scenario and used to control a physical agent.

1 INTRODUCTION

Smart solutions for production processes are key for a competitive manufacturing industry. While the number of multipurpose industrial robots has significantly increased (Roland Berger Strategy Consultants, 2014), the adaption of robot behavior to frequently changing processes is still time-consuming.

Machine learning offers a promising approach to reduce the adaption effort. Due to the lack of new process training data, reinforcement learning algorithms have to be used to explore an efficient robot behavior strategy based on *trial and error*.

However, an *error* in an industrial shop floor environment may result not only in damaged goods and equipment, but also (possibly deadly) injuries. Thus, the exploration has to be done in a simulated environment by the robot's digital twin. The findings of that exploration lead step by step to a successively improved behavior model.

Later on, the trained behavior model can be used to control the physical robot with the best practices that have been discovered during the simulation and to avoid the detected misbehaviors. In contrary to the simulation domain, the robot's behavior is shifted from *explore* to *exploit*.

Within this paper, we show the soundness of this approach using a prototype based on the physical robot *Cozmo* that is usually used as an educational toy in student AI classes (Touretzky, D., Gardner-McCune, C., 2018). The digital twin was built and operated within the *Unity* development platform. While originally designed as a workbench for 3D game development, the platform has been evolving to a serious simulation tool. The open source *Unity ML-Agent Toolkit* provides the architecture to train digital twins (Juliani, A., et al., 2018).

As a testing scenario the robot's task is to follow a line which is printed on the floor. As long as the line can be detected the robot tries to follow it. Otherwise the attempt is reset and restarted.

2 ROBOT

2.1 Cozmo

Cozmo is a small toy robot produced by a company called Anki. It is equipped with a caterpillar drive, a lift and a small 128x64 pixel display. In addition, it has sensors like a camera, a cliff sensor and an accelerometer. The robot also creates a WIFI network for other devices (e.g. mobile phones) to connect.

Besides the physical robot, Anki also provides an appropriate open source Python SDK in order to interact with it. A few lines of code can be used to get the raw data from Cozmo's sensors or control its movement. (Anki, 2019a; Anki, 2019b)

[a] https://orcid.org/0000-0002-3135-4923
[b] https://orcid.org/0000-0002-9714-9869

Hassel, T. and Hofmann, O.
Reinforcement Learning of Robot Behavior based on a Digital Twin.
DOI: 10.5220/0008880903810386
In *Proceedings of the 9th International Conference on Pattern Recognition Applications and Methods (ICPRAM 2020)*, pages 381-386
ISBN: 978-989-758-397-1; ISSN: 2184-4313

2.2 Actions

Cozmo is primarily designed to work as a virtual companion. Therefore, its behavior creates the impression of a personality. For this purpose, it is for instance able to recognize different human faces and to differentiate between cats and dogs.

However, the most important actions for the approach of following a line are movement, taking pictures and setting the lift as well as the head height. While positioning the robot's lift and head is only needed at the start to get an optimal view of the scene in front of the robot, the real challenge is the permanent control of the movement.

In order to keep things simple, the movement was limited to a finite set of actions. Consequently, the robot was forced to use one of the predefined actions move forward, turn left, turn right or stop. Each of these options can only be carried out in a certain speed that is given by the physics of the engine and the geometry of the robot itself. Turn left and turn right lead to a short speed difference of the left and the right caterpillar resulting in a slight change of the robot's heading of only a few degrees.

2.3 Observations

One of the robot's sensors is a camera, which can be found in the bottom half of its head and whose pictures are accessible via SDK after subscribing to the proper event.

The robot needs its camera to identify the line drawn on the ground in front of it. Accordingly, the head height must be set to the minimum to let the camera focus on the ground. Additionally, the lift is not allowed to cover the line and must be set to the maximum height.

As a result, the camera shows a 640x480 pixel image of the line on the ground, which can be used as state information for reinforcement learning.

3 DIGITAL TWIN

3.1 Unity

A 3D-model of the robot was used to visualize the digital twin in Unity. This scaled CAD-model was originally created in Tinkercad, an online CAD-program (Gearcortex, 2018).

After reducing the number of the model's polygons, it was imported into Unity. Based on this 3D-data, the virtual camera was placed. The position corresponds to the spot within the real robot. In addition, the Cozmo SDK was used to get the head angle from the physical robot in order to replicate it on the virtual camera.

A comparison between the image of the virtual and the real camera was made. For this purpose, an object was positioned right on the edge of the particular camera's field of view. Afterwards, the distance between the object and the specific camera was measured. As a result, the virtual camera was adjusted to match the real-world camera.

Finally, the Unity components *Boxcollider* and *Rigidbody* were added to the digital twin. These scripts ensure that the virtual Cozmo can be moved. Moreover, it can collide with other objects in the Unity scene (Fig. 1).

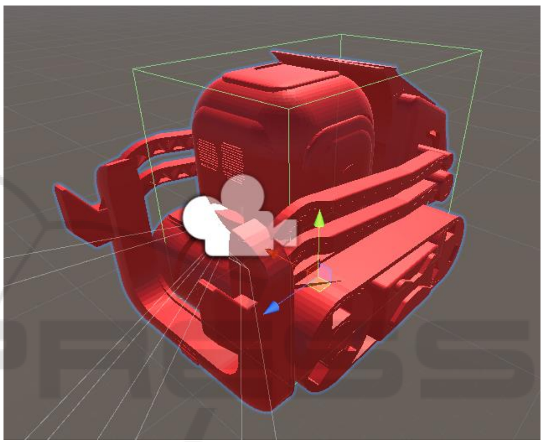

Figure 1: Model of the Digital Twin.

3.2 Actions

The virtual Cozmo uses the same discrete action space as the physical robot. The actions correspond with the movement states of the real robot, namely move forward, turn left, turn right and stop.

In order to reproduce the different possibilities in the virtual environment, the exact movement characteristics of the real robot must be known. The API can indeed be used to set the speed of the physical robot. However, to achieve proper results the actual behavior has been measured. This applies to both, the forward motion and the rotary motion.

In order to simulate this speed in virtual mode, a distance was defined. The times which the real as well as the virtual Cozmo needed to complete the distance were measured. By comparing the required times of the robots, the speed of the virtual robot was adjusted until the times coincided.

The same procedure was performed for the rotation speed. It is for this reason that a fixed rotation has been set for the real robot to complete. In addition, the required time was recorded. The virtual robot was then parameterized so that it could complete the same rotation at the identical speed.

3.3 Observations

The agent must be aware of the world's state. Therefore, the robot needs to keep an eye on the environment around it.

In our case, a visual observation was used that mimics the camera image of the real robot. With the help of a virtual camera, which renders its image into a texture, the state of the environment can be observed. In this case, the texture offers the same resolution as the real camera. Additionally, the camera must be aligned at the same position and angle to the ground as its real-world equivalent.

4 LEARNING APPROACH

4.1 Definitions

The sensors of the robot define a set of possible states of the environment. At each time t the robot receives information about the state S_t of the environment from its sensors.

$$S_t \in \mathcal{S}$$

The robot then has to select an action A_t out of the given set of actions

$$A_t \in \mathcal{A} = \{\text{forward, left, right, stop}\}$$

The selection of A_t is based on the reward R_{t+1} for the robot's actions A_t in a given state S_t leading to a new state S_{t+1} at time t.

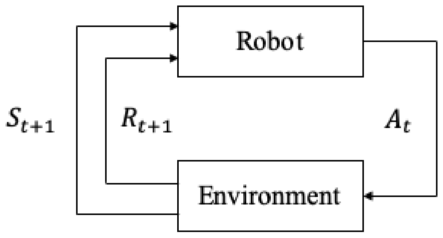

Figure 2: Robot-Environment Interaction.

4.2 Reinforcement Learning

As long as the environment's behavior can be depicted with a known deterministic function

$$f: \mathcal{S} \times \mathcal{A} \longrightarrow \mathcal{S} \times \mathbb{R}$$
$$f(S_t, A_t) \doteq (S_{t+1}, R_{t+1})$$

The action A_t can be easily chosen as that action which leads to the maximum sum of all future rewards. But in reality, this function is often neither known nor deterministic.

To keep it simple, in the robot domain we postulate a deterministic environment, i.e. an action A_t at a state S_t will always result in a specific subsequent state S_{t+1} with a reward R_{t+1} given by a well-known reward function r:

$$r: \mathcal{S} \longrightarrow \mathbb{R}$$
$$r(S_{t+1}) \doteq R_{t+1}$$

Further, we expect the reward R_{t+1} to deliver a proper indication of all future rewards due to the lack of hidden shortcuts in our domain. Therefore, the function f only has to estimate the reward R_{t+1}, because no follow up calculation of future rewards based on S_{t+1} is needed.

However, at the beginning of the robot's training, we do not know the deterministic outcome of any action A_t at a state S_t. In order to maximize the sum of all future rewards, the robot has to predict the reward of any action A_t at hand. Therefore, the unknown function f is approximated by a neural network that is trained with the experiences which have been made during the ongoing trial-and-error attempts.

At first glance it seems obvious for the robot always to select that A_t which has the expected max sum of future rewards. In doing so, we *exploit* the knowledge of our trained model. The physical robot should follow this policy to avoid damage and injuries.

During the training of the estimating neural network, this behavior is not optimal, because undiscovered actions are then less attractive than actions providing a low positive reward. As a consequence, the search for best solutions is narrowed to already tested actions. Thus, the digital twin has the permission to select from time to time unreasonable actions in order to *explore* new behavioral patterns (Sutton, R., Barto, A., 2018, p. 3).

4.3 Implementation

4.3.1 System Architecture

For the approximation of the function f we use a learning model. As input we provide the current state of the robot given by the image of the robot's camera. During the training phase, the virtual robot performs an action and the resulting new state is rated. This calculated reward is the expected outcome of the learning model.

After a sufficient amount of training the model can estimate the expected reward in a given state for any action. At this time, the model can be used to control the "real" physical robot. Therefore, a standardized interface to both robots is applied (fig. 3).

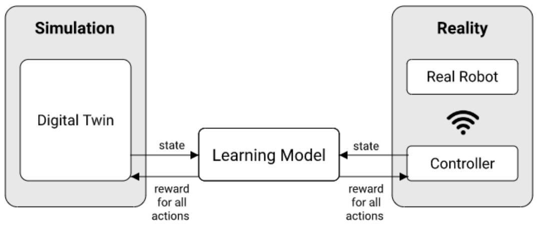

Figure 3: System Architecture.

The learning model is implemented as a neural network in TensorFlow, an open source machine learning library. In addition, the training process can be visually displayed and monitored with TensorBoard. This program allows weak points in the choice of hyperparameters for training to be identified and subsequently adjusted. (Unity Technologies, 2019a)

4.3.2 Data Preprocessing

The robot's state is given by the camera image. To reduce the learning effort the image is preprocessed to shrink the image's complexity and to focus on the essential data. These preprocessing operations must be performed for both the real and the virtual camera image.

First, the image is cropped. Therefore, the upper and the lower part of the image are cut off. As a result, the unimportant information of the image is omitted. For example, curves that are far away but still visible to the camera are neglected, because they would otherwise affect the robot's path.

After the image has been cropped, the Canny algorithm is applied. This algorithm is used for edge detection and converts the result to a binary image. The edges are displayed as white pixels while the rest remains black. This step will be needed later for the reward function.

Finally, the resolution of the resulting texture is reduced so that the final image consists of 80x30 pixels.

The same processing steps are carried out for the camera image of the real robot as for the visual observation of the simulation.

4.3.3 Rewards

The robot has to receive positive feedback for following the line. Based on the preprocessed image a reward function was defined. The binary image is used to calculate the Center Of Gravity (COG) of the white pixels (fig. 4).

In a binary image, the COG can be computed efficiently. The average of the white pixels' X- and Y-coordinates in the image is calculated. The resulting point with the average X- and Y-coordinates is called COG. (Mallick, 2018)

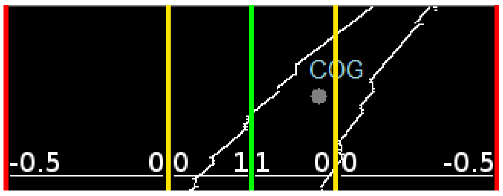

Figure 4: Center Of Gravity (COG).

If the COG is further to the right of the image, we know that there is a right curve in front of the robot. The same is true for the other direction. If COG is located exactly in the middle of the image, a positive reward of 1 is passed on to the robot. The further away the COG is from the center of the image, the lower the reward will be. If the position of the COG reaches one of the turning points marked in yellow in Figure 4, the reward changes from positive to negative. Again, the reward changes as the COG approaches the edge of the image. However, the negative reward gets bigger and bigger and rises up to -0.5 at the edge of the image. If the COG is outside the image, i.e. no longer recognizable, a fixed reward of -1 is awarded. In addition, the virtual robot is reset to its training start point.

Further, small rewards were awarded for different actions. So, for every decision the digital twin makes, a small negative reward was forwarded to the agent to tempt him to find the fastest way. In addition, the move forward action was given a positive reward, while all other actions were given small negative rewards. The reason for this was that the agent should not learn to turn from right to left on the spot in order to maximize his reward.

4.4 Evaluation

In order to evaluate the trained model, a test track was created on a 70cm x100cm paper sheet (Fig. 5).

The test track intentionally lacks sharp curves due to the robot's limited vision capabilities. The evaluation is based on several attempts with changing starting position and direction (clockwise/counterclockwise).

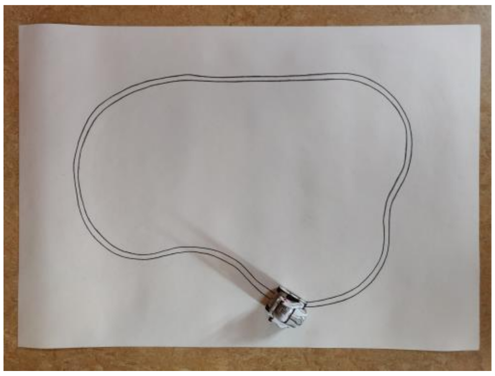

Figure 5: Test Track.

4.4.1 Line Tracking

Based on our approach the physical robot was able to follow a line using a learning model that was trained with its digital twin.

The robot's movement was quite slow due to the selection of the *stop* action which took more time in the real robot than in the simulation and consequently influenced the other actions. Therefore, the action was taken out, so that the robot only had the possibility to move or rotate but not to stop.

However, a kind of trembling was observed in the real Cozmo. Too often a left turn came after the decision to drive right and the other way around in order to mitigate the effects of the last action.

4.4.2 Improvements

This is why a so-called *action mask* was introduced for the next training. With the help of such a mask it is possible to forbid the robot specific actions after a previous one was executed. This method was used to give the robot a kind of *memory* about its last actions. If there is a decision to move to the left among its last actions, it is not allowed to move to the right.

Finally, four models with different memory sizes were trained and compared. The following illustration shows a training graphic of the models supplied by Tensorboard.

The first trained model without any action mask is visualized by the orange plot in Fig. 6. The model with adapted action mask and a memory size of one is represented by the dark blue plot. The colors red and light blue show the training process for the last trained models with the memory sizes three and five respectively.

Figure 6 shows the cumulative reward of each environment step. A run consists of a maximum of 1500 single steps, which means that the maximum cumulative reward per episode is 1500.

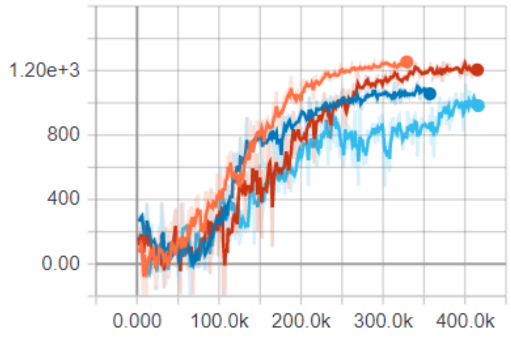

Figure 6: Cumulative Reward.

It can be seen that the model trained first achieves the highest cumulative reward fastest, while the light blue plot with memory size five scores the weakest. The only serious competition measured by cumulative reward is the model that considers the last three actions.

Based on several tries in reality, the training results of the Unity simulation have been approved: the introduction of an action mask based on a memory of its last actions downgrades the robot's ability to drive curves. The robot can drive curves worse the larger its memory is designed.

Overall, it turned out that very sharp curves were a problem for the robot, no matter which model was used.

5 CONCLUSION

In summary it can be said that it is possible to use Unity as a simulation environment and to transfer the resulting model to the real robot. The training can thus be made more cost-effective and less time-consuming than it would be without a simulation. The project demonstrates that it is also possible to use reinforcement learning with simple means for realistic application cases.

In addition, the project can be further improved, for example by using a continuous action space for the model. The hyper parameters can also be adjusted to make the training process more effective.

Further, the current status of the project uses the frame rate of the robot's camera as the decision frequency. This means that 30 decisions are made per second on how to proceed. This frequency can be reduced to achieve better results and to mitigate the trembling.

Currently, the calculation of the reward is based on a single image. In further work, the reward could be calculated using a moving average of the COG based on the last several images. This approach avoids the usage of action masks and memory, mitigates trembling but also allows subsequent turns at sharp curves.

REFERENCES

Anki, 2019a. Die Technik. Retrieved 2019-09-15 from https://anki.com/de-de/cozmo/cozmo-tech.html

Anki, 2019b. The Future of Consumer Robotics -Now. Retrieved 2019-08-31 from https://developer.anki.com/

Gearcortex, 2018. Anki Cozmo. Retrieved 2019-09-24 from https://www.tinkercad.com/things/diQp4OfpRl6

Juliani, A., et al.., 2018. Unity: A General Platform for Intelligent Agents. *arXiv:1809.02627 [cs.LG]*

Mallick, S., 2018. Find the Center of a Blob (Centroid) using OpenCv (C++/Python). Retrieved 2019-09-28 from https://www.learnopencv.com/find-center-of-blob-centroid-using-opencv-cpp-python/

Roland Berger Strategy Consultants, 2014. Industry 4.0, The New Industrial Revolution: How Europe Will Succeed. *International Conference The Next Industrial Revolution Manufacturing and Society in the XXI Century,Turin*

Sutton, R., Barto, A., 2018. *Reinforcement Learning*, MIT Press, Cambridge, 2nd edition.

Touretzky, D., Gardner-McCune, C., 2018. Calypso for Cozmo: Robotic AI for Everyone. *49th ACM Technical Symposium on Computer Science Education, Baltimore*

Unity Technologies (2019a). Background: TensorFlow. Retrieved 2019-09-23 from https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Background-TensorFlow.md

Unity Technologies (2019b). Learning-Environment-Design-Agents. Retrieved 2019-09-16 from https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Learning-Environment-Design-Agents.md