INFRASTRUCTURE FOR TRAINING AND VALIDATING AI ALGORITHMS IN SPACE APPLICATIONS BY MEANS OF DIGITAL TWINS

André Kupetz[(1)], Tobias Osterloh[(2)], Ulrich Dahmen[(3)], Jürgen Rossmann[(4)]

[(1)]RIF e.V., Joseph-von-Fraunhofer-Straße 20 – 44227 Dortmund, Germany, andre.kupetz@rt.rif-ev.de
[(2)]MMI, RWTH Aachen University, Ahornstraße 55 – 52074 Aachen, Germany, osterloh@mmi.rwth-aachen.de
[(3)]MMI, RWTH Aachen University, Ahornstraße 55 – 52074 Aachen, Germany, dahmen@mmi.rwth-aachen.de
[(4)]MMI, RWTH Aachen University, Ahornstraße 55 – 52074 Aachen, Germany, rossmann@mmi.rwth-aachen.de

ABSTRACT

A crucial challenge of transferring existing AI-based approaches to aerospace applications is the very limited availability of reference data.
This paper presents a novel infrastructure for the effective and efficient development, validation, and use of modern AI functionalities in space applications. By means of comprehensive Digital Twins, which incorporate holistic simulation models, the presented infrastructure allows to efficiently generate training data, to provide decision support methods for the selection and optimization of AI-based solutions, and finally supports the validation of the behavior of the overall system with representative data sets. In addition, the concept allows for a bidirectional technology transfer from terrestrial to space applications and vice versa. Consequently, the presented infrastructure constitutes an enabling technology for the sustainable development of AI-based systems in space applications.

ACKNOWLEDGMENTS

This work is part of the project "KImaDiZ", supported by the German Aerospace Center (DLR) with funds of the German Federal Ministry of Economics and Technology (BMWi), support code 50 RA 2015/2103.

1. INTRODUCTION

Modern terrestrial applications in automation and robotics reveal the tremendous capabilities and potential of AI-based approaches. As required by all machine-learning approaches, a huge amount of data for the development and validation of the respective AI-based functionality is necessary. A crucial challenge of transferring existing AI-based approaches to aerospace applications is the very limited availability of reference data. Real world reference data for space applications come at enormous cost and in many cases the data is not even available yet. Additionally, testing AI-based functionality in its operational environment (e.g., in orbit) increases the risk for a mission failure. Thus, before even thinking about the qualification of AI-based space functionalities, the persisting challenge of data deficiency for the development as well as for the validation must be taken up.

This paper faces this challenge by presenting a novel infrastructure which enables the effective and efficient development, the validation, and the use of modern AI functionalities in space applications. The key component that enables us to do this is the use of Digital Twins. By means of comprehensive Digital Twins [1], which provide holistic simulation models, the presented infrastructure allows to efficiently generate training data, to provide decision support methods for the selection and optimization of AI-based solutions, and finally supports the validation of the behavior of the overall system with representative data sets. Since the basic idea of the concept is not domain specific, it allows for a bidirectional technology transfer from terrestrial to space applications and vice versa. Thus, it promotes the integration of existing and validated terrestrial AI knowledge. Consequently, the presented infrastructure constitutes an enabling technology for the sustainable development of AI-based systems in space applications. Figure 1-1 illustrates the basic idea.

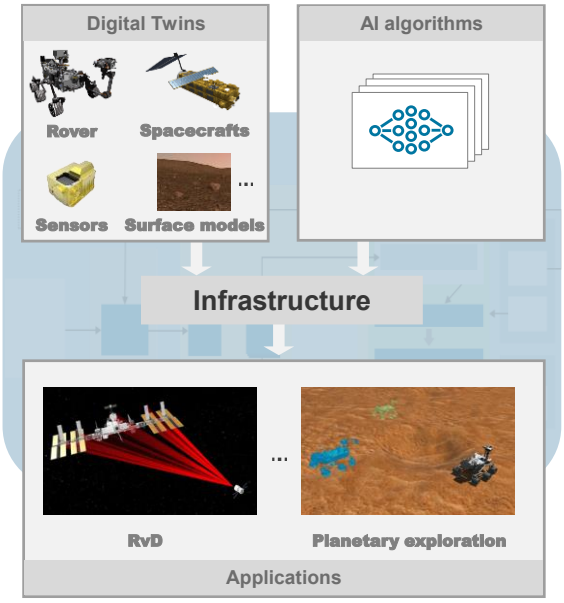

Figure 1-1: Idea of the presented approach

We will demonstrate the capabilities of the developed infrastructure in two use cases: the first one is in the context of training of an algorithm for AI-based pose

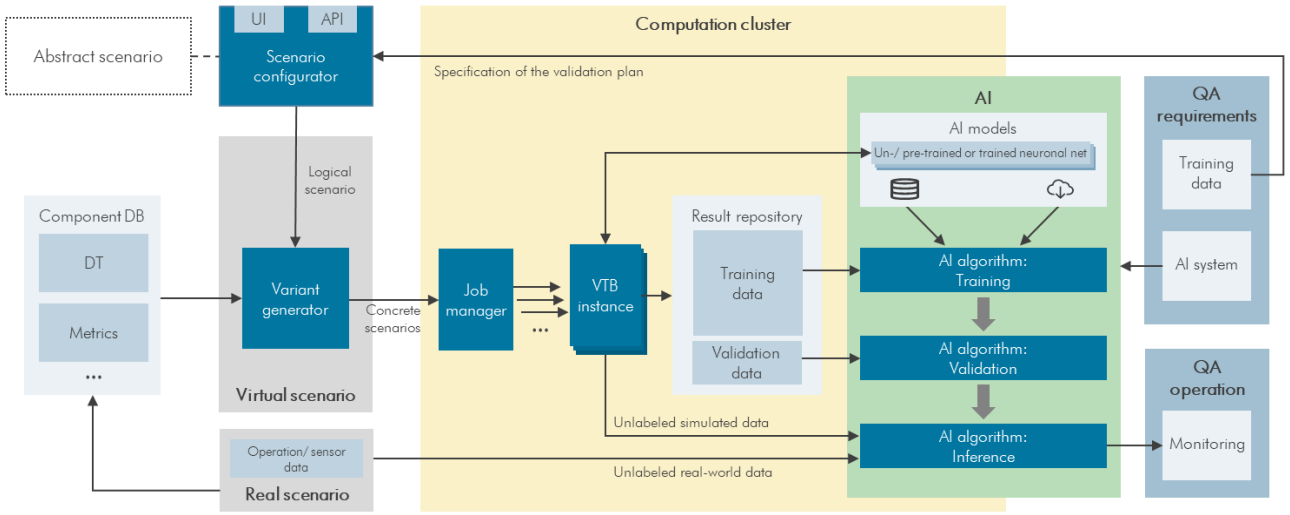

Figure 3-1: System architecture for the infrastructure

estimation in rendezvous and docking (RvD) scenarios, and the second one is in the context of automated reinforcement learning, analysis, and evaluation of different AI variants for the navigation of a rover for planetary exploration missions.

2. RELATED WORK

Nowadays, AI-based systems are established in a multitude of application domains (e.g., image and speech processing). Independent of any application, the development and validation of AI-based systems highly relies on large amounts of high-quality data. With systems becoming autonomous, this amount of data is ever increasing. Consequently, simulation technology and data generated by simulation is widely getting acceptance in the development of AI-based systems, filling the gap of the data not yet available. This applies in particular to the automotive industry and the development of advanced driver assistance systems (ADAS) [2]. But also, the European Cooperation for Space Standardization (ECSS) underlines the value of simulation in the development process of products [3].

Detached from the development and validation of AI-based system, the *ISO 21448 – Safety of the Intended Functionality* (SOTIF) faces a similar problem. The SOTIF process deals with the operation of systems in potentially unknown environments. Therefore, the SOTIF norm defines a procedure to identify unsafe operational scenarios of a system under test. Based on this, the result can be fed back to the development and minimize the number of known unsafe scenarios. This is depicted schematically in Figure 2-1.

This approach can be transferred to the development of AI-based systems. Our research focuses on scenario-based testing [3]. We developed a scenario-based framework, which allows to generate high quality training and validation data of AI-based systems. Therefore, our approach closely relates to formalized specifications to describe scenarios, like the OpenSCENARIO standard [4].

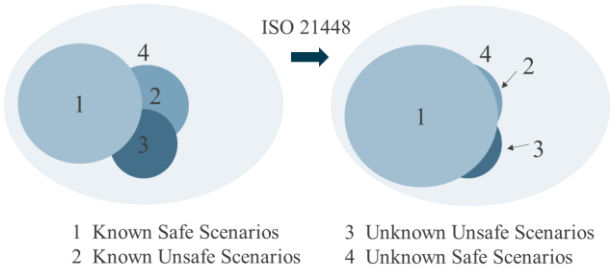

Figure 2-1: Goal of the SOTIF process

3. BASIC ARCHITECTURE OF THE INFRASTRUCTURE

Figure 3-1 illustrates the basic idea of the overall system architecture for the novel infrastructure.

A *virtual scenario* is a representation of a problem that should be investigated (*real scenario*). Applying comprehensive simulation technology, as provided by a virtual testbed (*VTB*), a virtual scenario allows for the generation of training and validation data for the development of the AI. In our case, we build upon the VEROSIM simulation platform [1].

The infrastructure distinguishes between an *abstract scenario* (an abstract description of the problem at the semantic level), a *logical scenario* (a detailed description with entity relationships and parameter ranges defining a parameter space) and a set of *concrete scenarios* (unique scenarios with fixed values in the given parameter space). An abstract scenario serves as template for the generation of a logical scenario (e.g., by manual configuration with a UI, or automatic configuration, generated via an API). Then, a randomization engine processes a logical scenario and generates a multitude of concrete scenarios, taking the given constraints of the logical scenario into account (*variant generator*). Finally, a *job manager*

distributes the concrete scenarios to multiple instances of a virtual testbed to enable a parallel simulation of each concrete scenario on a *computation cluster*. For this purpose, each concrete scenario is automatically translated into an appropriate simulation model, making use of Experimentable Digital Twins stored in a *component data base* [5]. The training and validation data generated by the simulation in the virtual testbed is managed in the form of a *result repository*.
The concept is accompanied by *quality assurance (QA)* measures that influence both, the AI system, and the training data to be generated. For this purpose, the generated data is constantly monitored regarding its relevance. Additionally, the performance of the deployed AI algorithms is evaluated. At the same time, QA monitors the operation to detect behavioral changes of the AI (e.g., due to changing environmental conditions in the real application) at an early stage. Based on the results of this continuous evaluation, it is then possible to fine-tune the scenario configuration and generation to continuously increase the informative value of the generated training and validation data.

4. INFRASTRUCTURE FOR AI-BASED APPROACHES SUPPORTED BY VIRTUAL TESTBEDS

One key component of our approach is to develop an infrastructure that enables training and validation of AI algorithms using the execution of Digital Twins within a virtual testbed. To this end, the infrastructure must enable an efficient generation of high-quality training data. Depending on the use case the training data may need to be labeled. A major advantage of a simulation-based data generation is the availability of ground truth data, which effortlessly can be used to automatically generate labeled training data. This typically is of great importance for the labeling of sensor data like cameras (e.g., assigning corresponding label-IDs to coherent objects). Therefore, our virtual testbed realization provides rendering-based sensor simulation capabilities, resulting in Digital Twins of cameras or LIDAR sensors [10]. The labeling process is supported by a template to define specific label requirements, for example required data formats or specifications concerning data dimensionalities or hierarchical structures. These requirements are passed to our developed labeling interface, which then labels the simulation data. Training data can immediately be reused from the virtual testbed, e.g., for training in a supervised learning application, as well as being externally stored for later use.

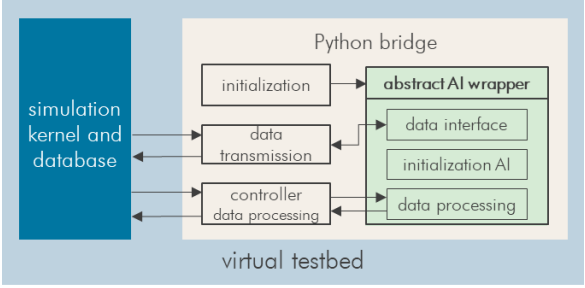

Figure 4-1: Schematic representation of the Python bridge for integrating AI algorithms

In order to integrate AI algorithms, the virtual testbed is able to serve as a Python interpreter. We developed a Python bridge, which allows to connect a Python-based AI algorithm with the simulation core of the virtual testbed. Therefore, the Python bridge allows the initialization and execution of an AI wrapper that is represented by a Python script too, and at the same time the Python bridge provides an interface to the simulation kernel and the database (Figure 4-1). Consequently, current simulation data can be immediately processed by the AI algorithm and vice versa.
Therefore, the abstract AI wrapper serves as a template and describes the required functions for the initialization and data processing, which then can be implemented regarding all application specific requirements. The implementation of the template is carried out independently of the interface for the simulation database and enables Python-based access to pre-existing AI frameworks such as PyTorch or TensorFlow. Finally, trained AI algorithms can be integrated back into the virtual testbed via the Python bridge, where the interactive in-the-loop validation for e.g., corner cases can be performed.

5. METHODOLOGY FOR AN AUTOMATED CREATION OF SIMULATION SCENARIOS

The developed methodology for the automated generation of simulation scenarios distinguishes three notions of scenarios and describes a formal process to connect and automatically derive the respective scenarios. Within our methodology, all scenarios are based on an XML specification that - in its core design - closely relates to the OpenSCENARIO standard from the automotive industry. The fundamental idea of the pursued approach is depicted in Figure 5-1.

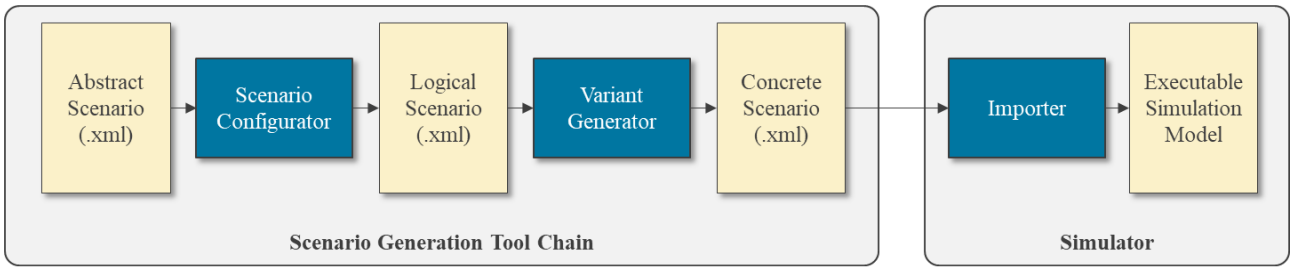

Figure 5-2: Generation of executable simulation models by generating concrete scenarios from abstract scenario specifications

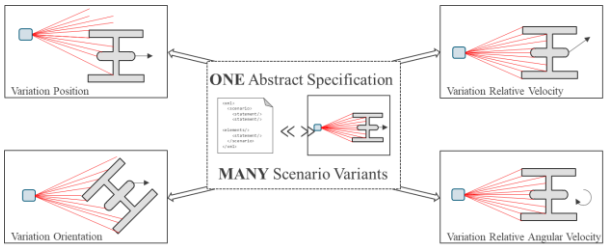

Figure 5-1: One abstract scenario specification is transformed to many different scenario variants

Abstract Scenario: The abstract scenario is a formalized representation of e.g., a test or a training case. The abstract scenario defines all relevant entities, their initial configuration and their interactive behavior. Additionally, it specifies all parameters that can be altered.

Logical Scenario: The logical scenario enriches the abstract scenario by the definition of parameter ranges and parameter constraint. The logical scenario spans the entire scenario space and allows to purposefully constrain certain regions in the potential high dimensional vector space.

Concrete Scenario: The concrete scenario is a parametrized version of an abstract scenario. It fulfils all constraints defined by the logical scenario. It is possible to automatically translate concrete scenarios to executable simulation models.

As shown in Figure 5-2, the different scenario versions are generated and connected via different tools/software modules. The first stage of the scenario generation tool chain is formulated independent of any simulator. Only after creating a concrete scenario, an executable simulation model is generated. Since concrete scenarios are defined based on an XML-file, they are usually much smaller and thus better to handle than simulator specific executable simulation models.

Scenario Configurator: The scenario configurator is a tool, which is implemented specifically for certain applications and allows to automatically generate a logical scenario, e.g., based on user defined parameter ranges and constraints.

Variant Generator: The variant generator is a generic software module, allowing to create an arbitrary number of concrete scenarios based on the description of the logical scenario. Therefore, the variant generator realizes a Markov-Chain-Monte-Carlo based stochastic process, generating and evaluating random parameter samples [6]. The variant generator solely relies on a random process. Theoretically, it would be possible to traverse the entire parameter space with a brute force approach, but many unnecessary scenarios without new information will be generated, making the process inefficient. We rely on the stochastic process exploring the parameter space, and if interesting sub-spaces in the overall parameter space are found, it is possible to specifically generate new scenarios in this sub-space.

Importer: Only in the last stage of the tool chain, a simulator specific executable model is generated. Of course, this importer is realized specifically for the desired simulator. Typically, this translation process relies on a repository of Digital Twins (i.e., preconfigured and validated simulation models of all relevant entities). Based on this, the simulation model is created, the initial values are assigned, and a scripting of the desired behavior of all systems is integrated [7].

Once, the executable simulation model is available, the simulation can be carried out and the training and validation data for an AI-based system can be generated and collected in a database.

6. DECISION SUPPORT FOR AI-BASED APPLICATIONS

It is commonly accepted that it takes large amounts of data and test cases to train and validate AI-based applications. Due to this increasingly growing number, it is important to automatize the analyses of simulation results, in order to support experts in the development and validation of AI-based systems.

Our developed process for the decision support directly ties in after the simulation of a concrete scenario, see Figure 6-1. During simulation all relevant properties of the Digital Twins are logged to an SQLite database.

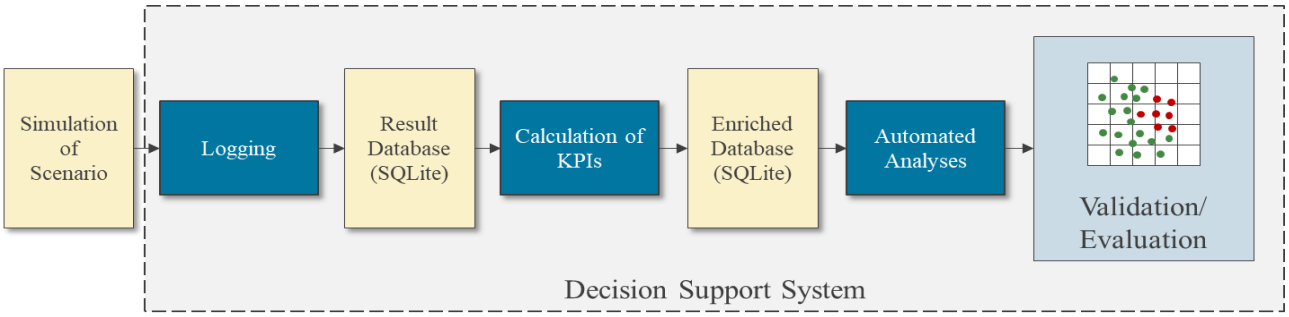

Figure 6-1: Data driven decision support for AI-based applications

Therefore, either white-list or black-list logging approaches can be chosen. Both of which have specific advantages and disadvantages with regard to their performance and analyses capabilities.

Once the initial result database is generated, additional KPIs and metrics can be computed in a post-processing step, resulting in an enriched database. Based on these numeric values, it is possible to execute automated evaluations of the simulation results. If sensible, it is possible to calculate certain metrics or KPIs while simulating the concrete scenario, facilitating the post-processing of the simulation results. It is important that the post-processing is more flexible since it directly operates on the database and does not require to carry out new simulations if new KPIs are identified and should be calculated.

Finally, this database can be used for statistical analyses, e.g., to identify critical operational scenarios of an AI-based system. Additionally, the comprehensive analysis allows to directly replay the simulation, empowering experts to intuitively analyze simulation results and drawing conclusions in the context of decision support, see Figure 6-2. These results can now iteratively be fed back, in order to derive more meaningful logical scenarios, creating new relevant scenarios either for training or validation of the AI-based system.

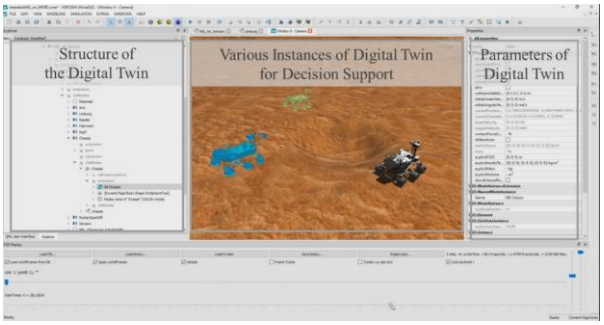

Figure 6-2: Using multiple instances of a Digital Twin (represented by their ghosts) in context of decision support

7. EXAMPLE APPLICATIONS

The following applications demonstrate the capabilities of the developed architecture and its fundamental design concepts.

7.1. Automated Rendezvous and Docking

The first application focuses on relative pose estimation required for automated Rendezvous and Docking (RvD) maneuvers. We assume that the pose estimation is performed by an AI-based system, which needs to be trained and validated. Therefore, we build upon a calibrated Digital Twin of a space LIDAR that generates point clouds during the approach (i.e., the point clouds are distorted due to the relative dynamics of target and chaser).

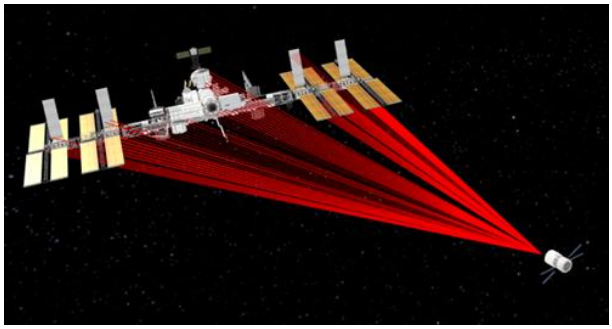

Figure 7.1-1 Generation of training and validation data using a Digital Twin of a space LIDAR

First, we begin by defining the abstract scenario. Table 1 highlights the relevant entities within the scenario, as well as the parameters that should be considered during variation. The relevant entities of the scenario are provided in a Digital Twin repository (i.e., the target can be changed easily, by referencing a different entity from the repository). The Digital Twins of the LIDAR is already integrated to the Digital Twin of the Chaser. For clarity reasons, we do not describe the scenario based on the developed XML-scheme.

Table 1: Abstract Scenario of RvD use case

Relevant Entities	Chaser Target
Variable Parameters	Pose Chaser $\vec{p}_{Chaser}, \vec{\varphi}_{Chaser}$ Vel. Chaser $\vec{v}_{Chaser}, \vec{\omega}_{Chaser}$ Pose Target $\vec{p}_{Target}, \vec{\varphi}_{Target}$ Vel. Target $\vec{v}_{Target}, \vec{\omega}_{Target}$

Subsequently, the logical scenario is specified, defining the parameter space of the scenario. For simplicity reasons, Table 2 only highlights some of the individual components of the variable parameters. As shown in the logical scenario, the parameters can either be distributed uniformly or based on a Gaussian distribution. The exemplary parameter constraint in the logical scenario enforces an approach of the chaser towards the target (i.e. an approach in x-direction is assumed).

Table 2: Logical Scenario of RvD use case

Parameter Ranges	$p_{Chaser,x} \in [0, 500]$ $v_{Chaser,x} \in [0, 1]$ $p_{Target,y} \in \mathcal{N}(0, 10)$ $v_{Target,x} \in \mathcal{N}(0, 0.01)$
Param. Constraints	$v_{Chaser,x} - v_{Target,x} > 0$

The variant generator can generate an arbitrary number of concrete scenarios, forming specific test cases for either training or validation of the AI-based functionality. Thus, a concrete scenario is a specific instance of an abstract scenario, fulfilling all constraints from the logical scenario.

Table 3: Concrete Scenario of RvD use case

Concrete Parameters	$p_{Chaser,x} = 250$ $v_{Chaser,x} = 0.5$ $p_{Target,y} = 1$ $v_{Target,x} = 0.01$

Finally, the concrete scenario can be translated to an executable simulation model. Details about the generation process can be found in [7]. Figure 7.1-1 depicts an impression from the simulation that is used to generate training and validation data. Currently, the training and validation process is in progress.

7.2. EXPLORATION

The general scope of the second application is, that a rover (a Digital Twin of the rover "Perseverance") learns to get from random start points to random target points autonomously on a 50m x 50m model of a Mars-like surface. The surface contains rocks and crater that the AI should learn to circumvent.

The particularly interesting part of this reference application is that a multitude of variants of an AI-based pathfinding system can be trained, evaluated and compared automatically. Therefore, we use the workflow described in section 5 to create concrete scenarios, which form the basis for different AIs. This means, here each concrete scenario is used to train a different variant of an AI system for a pathfinding problem. The entire process is based on reinforcement learning [8]. Since, all variants can be handled separately, this process can easily be parallelized (see Figure 7.2-1).

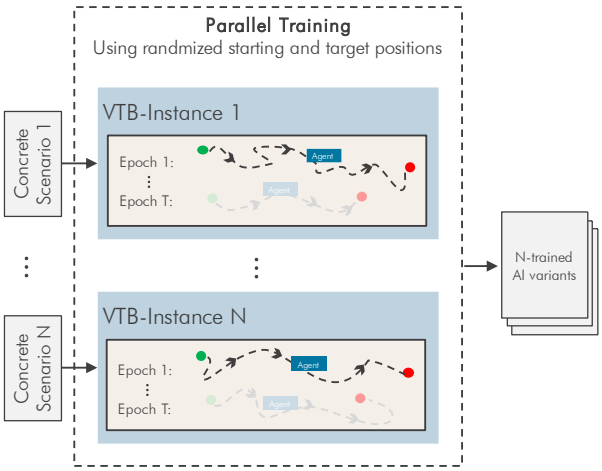

Figure 7.2-1: Parallel Training of AI variants for various concrete scenarios

Finally, the performance of all AI variants is evaluated and compared automatically with the help of quality indicators globally defined in the abstract scenario.
In contrast to approaches where the goal is to develop one highly sophisticated AI, we focus on coping with the multitude of adaptation possibilities within the complete application. This includes of course variations of the hyperparameters, reward function, but also variations of type and location of sensors used by the rover, or which types of movement actions the rover should offer. The definition of the variation possibilities is supported by a scenario configurator and stored in our XML-based logical scenario.
Instead of focusing on a single setup that may lead into a dead end, our infrastructure allows to investigate a wide range of variants to avoid such risks and leads the user into the right direction for his application setup.
Applying our approach, we implemented a common neural network, based on a deep Q-learning variant [9] as shown Figure 7.2-2. It is important to note that the variation capabilities also include arbitrary configurations of the sensor system of the rover (e.g., RGB camera, a time-of-flight camera, or a laser scanner). In any case the Python wrapper described in section 4, will preprocess this data and pass it to the neural network.

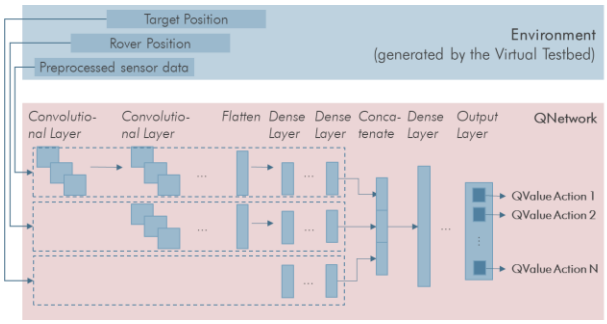

Figure 7.2-2: Layout of the neural network

First, the sensor data is processed by a set of convolutional layers, aiming for the detection of craters and rocks and thus the realization of a sensor-based perception of the environment. The final output layer returns a q-value for each action, which has a significant influence on the selection of the next action to be executed.
The following Figure 7.2-3 illustrates a snapshot during the training process of one variant. The rover is equipped with an RGB camera with a resolution of 64 x 64 pixel. The Python wrapper then turns the image data into luminance values for the neural network. A cutout of specified reward areas is illustrated in Figure 7.2-3 by means of the red (reward -1) and green (reward +5) metaphors. The complete reward function is defined as follows:

$$R_t = \left| Pos_{Rov}(x,y) - Pos_{Trg}(x,y) \right|_t \qquad (1)$$

$$R = MapRev(x,y) + R_{t-1} - R_t \qquad (2)$$

R_t represents the distance between the rover and the target at a time t. Since in the long run the distance of the rover to the target should reduce, $R_{t-1} - R_t$ represents a positive reward for getting closer to the target. Finally, the reward value for the rover's current position on the map (given by the reward areas) is added on top. This gives the total reward for the action that has just been performed.
Regarding the configuration of the hyperparameters, only the values of the three most important hyperparameters are outlined in Table 4.

Table 4: Excerpt of the hyperparameters

Parameter	**Value**
Gamma	0.99
Learning rate	0.001
Epsilon	0.4

Then, we limited the number of steps per episode to 100. Each step correlates to one action performed by the rover. This means if the rover is making too many detours, the episode is stopped. In consequence, we prefer to train AIs that will reach its destination quickly.

Finally, we defined the following four movement actions the rover can execute: move forward, move backward and turn on the spot clockwise and counter-clockwise.

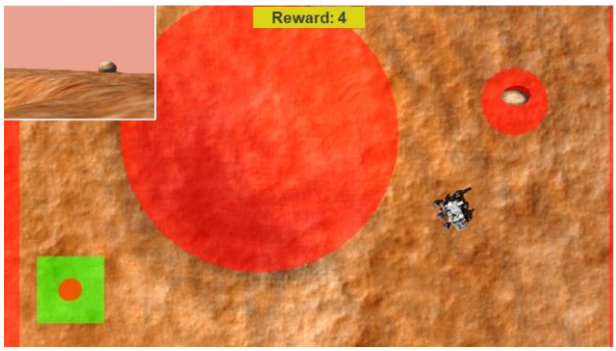

Figure 7.2-3: Snapshot during the training process

In a first test run, a configuration of the logical scenario was performed which resulted in 36 different concrete scenarios – respective AI variants to be trained. Figure 7.2-4 illustrates the current intermediate result of the test run. In all these 36 variants, we used an RGB camera and set the number of steps per episode to 100.
The color coding of Figure 7.2-4 shows how often an episode was aborted prematurely with respect to the target position that the rover was supposed to approach. For example, if the rovers target was within the red area in the lower left corner of the map, no matter where the rover's starting point was, it was quite unlikely that it would reach its destination before exceeding the maximum number of steps for an episode. The color-coded map is the result of all 36 variants.

Figure 7.2-4: Training status of a first test run

Currently, the training and validation process is in progress and different sensors and rover actions will be considered, as well as a wider range of values for the hyperparameters will be used. In addition, different metrics like shortest path or fastest path will be used to enable a comprehensive evaluation of the AI variants.

8. RESULTS AND CONCLUSION

In this paper, we presented a novel architecture to generate training and validation data for the development of AI-based systems. The inherent methodology relies mainly on three scenario definitions: the abstract scenario, the logical scenario and the concrete scenario. The development process that is aligned with the architecture, allows to purposefully use the results for QA purposes of AI-based systems, and at any point provides feedback opportunities, in order to adapt the defined scenarios. In turn, this results in a highly flexible data generation process, resulting in high-quality training data. In addition to data generation tasks, the architecture suits the requirements of reinforcement learning approaches, enabling to directly train agents in simulation. Currently, we apply the developed approach to space applications, but due to the generalized concept, we intend to apply the underlying fundamentals to terrestrial applications as well.

9. REFERENCES

1. Dahmen, U., Osterloh, T. & Roßmann, H.J. (2022). Verification and validation of digital twins and virtual testbeds. *International journal of advances in engineering sciences and applied mathematics* 11(1), ijaas. pp.47-64.

2. Fremont, D.J., Kim, E., Pant, Y.V., Seshia, S.A., Acharya, A., Bruso, X., Wells, P., Lemke, S., Lu, Q. & Mehta, S. (2020). Formal scenario-based testing of autonomous vehicles: From simulation to the real world, in *IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. pp1-8.

3. ESA-ESTEC, (2020). ECSS-E-ST-40-07. Online at *https://ecss.nl/get_attachment.php?file=2020/04/ECSS-E-ST-40-07C(2March2020).pdf*

4. Association for Standardization of Automation and Measuring Systems, (2021). ASAM OpenSCENARIO® Standard., Online at *https://www.asam.net/standards/detail/openscenario*

5. Schluse, M., Priggemeyer, M., Atorf, L. & Rossmann, H.J. (2018). Experimentable Digital Twins - Streamlining Simulation-Based Systems Engineering for Industry 4.0. *IEEE Transactions on Industrial Informatics*, vol. 14, no. 4, pp1722-1731.

6. Maqbool, O., Roßmann, H.J. (2022). Formal Scenario-driven Logical Spaces for Randomized Synthetic Data Generation. *10th International Conference on Model-Driven Engineering and Software Development.*

7. Dahmen, U., Osterloh, T. & Roßmann, H.J. (2021). Generation of Virtual Test Scenarios for Training and Validation of AI-based Systems. *IEEE International Conference on Progress in Informatics and Computing (PIC)*, Shanghai, China, pp64-71.

8. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. & Riedmiller, M. (2013). Playing Atari with Deep Reinforcement Learning. *DeepMind Technologies*. Online at *http://arxiv.org/abs/1312.5602*.

9. Arulkumaran, K., Deisenroth, M.P., Brundage, M. & Bharath, A.A. (2017). Deep Reinforcement Learning: A Brief Survey. *IEEE Signal Processing Magazine*, Volume: 34, Issue: 6, pp26-38.

10. Thieling, J., Roßmann, H.J. (2020). Scalable and Physical Radar Sensor Simulation for Interacting Digital Twins. *IEEE sensors journal* 21(3), pp3184-3192.