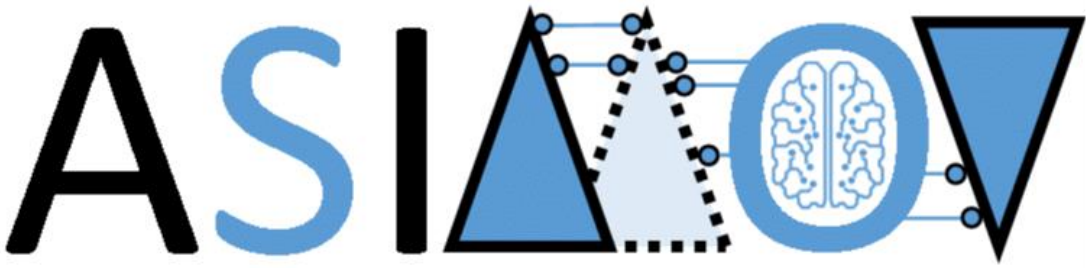
Proof of Concept Demonstration and Evaluation of Unmanned Utility Vehicle

[WP1; T1.3; Internal Report: D1.3 version 2]

public



AI training using Simulated Instruments for Machine Optimization and Verification

Document Information

Project	ASIMOV
Grant Agreement No.	20216 ASIMOV - ITEA
Deliverable No.	D1.3
Deliverable No. in WP	WP1; T1.3
Deliverable Title	Proof of Concept – Use case Unmanned Utility Vehicle
Dissemination Level	external
Document Version	Version 2
Date	2023-12-04
Contact	Niklas Braun
Organization	AVL Deutschland GmbH
E-Mail	Niklas.Braun@avl.com

The ASIMOV-project was submitted in the Eureka Cluster AI Call 2021
https://eureka-clusters-ai.eu/

Task Team (Contributors to this deliverable)

Name	Partner	E-Mail
Niklas Braun	AVL	niklas.braun@avl.com
Elias Modrakowski	DLR	elias.modrakowski@dlr.de
Andreas Eich	LiangDao	andreas.eich@liangdao.de
Lukas Schmidt	Norcom	Lukas.schmidt@norcom.de
Ezra Tampubolon	Norcom	Ezra.tampubolon@norcom.de
Thomas Kotschenreuther	RA Consulting	thomas.kotschenreuther@rac.de
Sebastian Moritz	Triangraphics	Sebastian.moritz@triangraphics.de

Formal Reviewers

Version	Date	Reviewer
2.0	2023-12-05	Pieter Goosen (TNO/ESI); Bram van der Sanden (TNO/ESI)

Change History

Version	Date	Reason for Change
2.0	2023-12-04	Initial M30 version

Abstract

The testing and calibration of diverse Unmanned Utility Vehicles requires a specialized set of driving tests, that challenge the vehicle. Driven by this idea, this deliverable gives an overview of the Components required to create such a system. It uses the methods and structure that has been proposed by other ASIMOV deliverables. A more detailed Introduction on vehicle testing, as well as unmanned utility vehicles is given before explaining the reasoning behind choosing the ASIMOV approach, including Reinforcement Learning and Digital Twins. After that, the general way of working inside the German ASIMOV Consortium is explained and the working results are presented. This is structured in 4 Workgroups, that have been established to face the various challenges in the UUV proof of concept. Hereby, the workgroups “Environment Simulation”, “Feature Engineering”, “Reinforcement Learning” as well as “Storage and Compute” are presenting their tasks, decisions and implementation results. The document closes with an overall status update and plans for the future.

Contents

Table of Figures

Table of Tables

1. Introduction

1.1 Unmanned Utility Vehicle

Unmanned Utility Vehicles (UUVs) offer a great possibility to deliver goods and lead to a sustainable alternative in public transportation while improving safety. In order to deploy a system of UUVs, for flexible people or goods transport, multiple different UUVs need to be developed and calibrated. Calibration reaches from UUV individual parameters for control units of the drive train up to parameters for communication between the UUVs or a teleoperator, which can act as Incident Management. Considering the large variety of possible UUVs, each designed with a specific purpose in mind, individual manual calibration becomes unfeasible when deploying a fleet of vehicles. To improve scalability of mass fleet deployment, as well as leading to a significant reduction in application cost during development, Digital Twins (DTs) and AI-based system optimization can be used as an enabler for smart mobility solutions. In Figure 1 such a vehicle, specialized for public transport can be seen in an urban environment.

Figure 1 - Unmanned Utility Vehicle example [1]

DTs and AI-based system optimization for UUVs offers a tool to significantly lower the need for testing on proving grounds and public streets and therefore lead to a reduction in cost. ASIMOV not only provides companies the possibility to scale in production, but also serves as a tool for development of alternative vehicle concepts in a more research orientated organisation.

Digital Twinning and AI-based parameter optimization is also not limited to vehicle parameters but can also be applied to the testing itself. The testing process as such can benefit in two ways from the ASIMOV idea. A well calibrated test bed leads not only to more accurate data but also to a wider accessible range of possible tests, including highly dynamic ones, which offer insight into vehicles driving characteristics in safety relevant scenarios. Additionally, optimizing the parameters of the relevant test scenarios leads to a more effective way of gathering meaningful measurement data and therefore reduces the required testing time in the lab and on the road.

The Unmanned Utility Vehicle Use case will focus on improving the testing itself by automatically creating a test plan that is best suited to test the vehicle.

As access to such a vehicle is rather limited, the use case will rely on virtual validation of the developed toolchain and methods. The autonomous driving (AD) stack itself will not be part of the optimization.

1.2 Scenario-based testing

Testing of Vehicles will be done by using Scenario-based testing methods. Formally, Scenario-based testing can be defined as "the enabling tool for the homologation of automated driving (AD) systems […]. This is mainly due to the aspiration of scenario-based testing for gaining understanding about the tested AD system, which provides support for arguments that shall convince ourselves about its capabilities" [2]. For this, a specific traffic situation is parameterized and used to estimate the vehicles behaviour in this specific situation. This allows the creation of test scenarios, that are interpretable, as they resemble real world situations. Its main advantage is to conduct systematically the verification, validation and accreditation (VV&A) in a controlled environment and putting emphasis on the rarest of events instead of testing the system out on the road.

Each traffic scenario uses a road network, described via the ASAM OpenDRIVE [3] standard and a scenario description, which defines the actors and the movement of the actors in an ASAM OpenSCENARIO [4] file.

1.3 Digital Twin

We aim for the use of DTs in this project as it is unfeasible to train a Reinforcement Learning Agent (RLA) directly on a vehicle testbed. Such a DT serves as a virtual playground for the RLA, where it can test actions and receive rewards and states, without having the cost-intensive operation of a vehicle testbed. The DT therefore has to be a realistic representation of its Physical Twin (PT), so that its interaction with the RLA is similar enough for the RLA to gain meaningful training data out of it. The digital representation of vehicle and environment will be further examined in the "Environment Simulation" Workgroup in 2.1.

1.4 Reinforcement Learning

Reinforcement Learning (RL) [5] is a goal-oriented learning paradigm alternative to the popular supervised learning approach. It consists of iterative cycles of observing – taking action – being rewarded or penalized. An agent gradually optimizes its actions by interacting with a training environment at discrete time steps, typically to maximize its cumulated rewards (return) in order to fulfil some given goals.

More formally, the RL paradigm is designed for an underlying discrete-time Markov decision process (MDP) [6] $\mathcal{M} = (S, \mathcal{A}, P, R, \gamma, D)$, where: $s \in S$ is a state; $a \in A(s) \subset \mathcal{A}$ denotes an action allowed in s; $P(s, a, s') = \mathbb{P}(s_{t+1} = s' | s_t = s, a_t = a)$ is the transition probability at any discrete time t (stationary w.r.t. time), which is only dependent on the current state and action (Markovian property); $r_t \sim R(s_t, a_t)$ is the probabilistic reward at time t; $\gamma \in [0,1]$ is a discount rate for the long-term return; $D(s_0)$ is an initial state distribution. Given this, the agent starts in an initial state $s_0 \in S$ and takes an action $a_t \in A(s_t)$ at each time step t. Then, the system makes a transition to $s_{t+1} \sim P(s_t, a_t)$ and the agent receives an immediate reward $r_t \sim R(s_t, a_t)$.

In order to maximize the return $\sum_{t=0}^{\infty} \gamma^t r_t$, RL aims at optimizing the agent's policy, modelled as a probability distribution $\pi(a \mid s)$, by use of state/state-action value functions, i.e., expected returns for following a given policy π. Common RL approaches include (Deep) Q-Learning [5] [7], SARSA [8], Temporal Difference Learning [5] and (direct) Policy Search methods [9].

2. Proposed workflow

The UUV1 use case will cover an automated adaptation of the traffic scenarios 3D environment, based on the vehicle's interaction with the environment. The resulting scenarios shall be critical and novel. This shall allow for the creation of an automated test plan generation, that tailors its test plan based on the individual weak spots of the tested vehicle. As the 3D environment is subject of optimization for the test plan, the focus of the test plans clearly lies on the sensor and perception of the vehicle. The more conventional testing of vehicle driving scenarios with different dynamic scenario parameters, e.g., target speed, distance offset, etc. could also be optimized in a similar way, but will not be changed in this use case. This further shifts the focus of this test plan optimization to perception and sensors.
To coordinate the work in the German UUV use case, four Workgroups were formed to cover all aspects of the envisioned solution. The workgroups are *Environment Simulation*, *Feature Engineering*, *Reinforcement Learning* and *Storage and Compute*. A visual representation of their connections and working fields can be found in Figure 2.

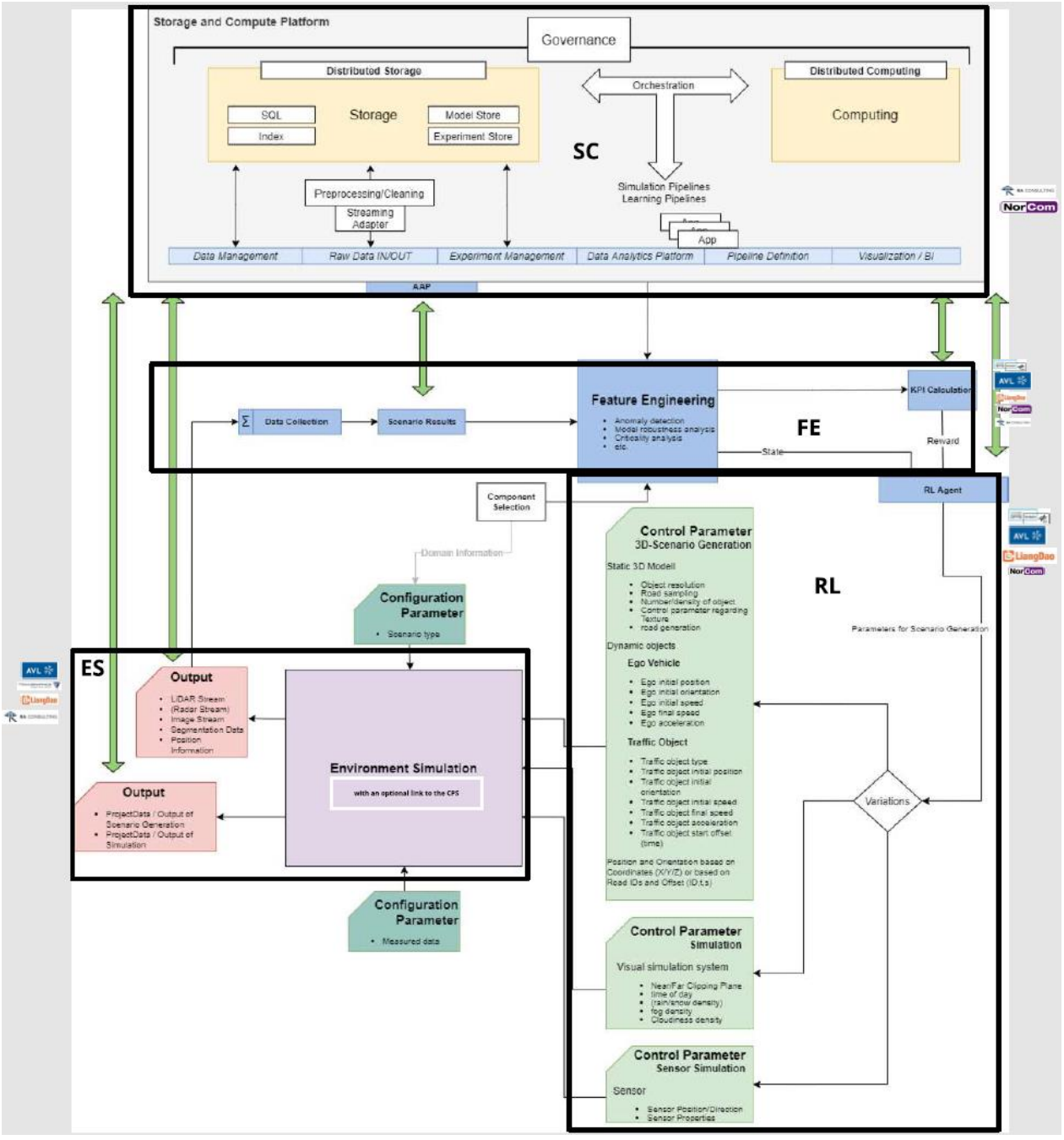


Figure 2 - Overview of the different Workgroups in the UUV Use Case

The next sections will describe the work of these groups in detail.

2.1 Workgroup: Environment Simulation

The environment simulation (ES) workgroup covers every aspect of the UUV Use Case, that is concerned with the simulation of the vehicle in its virtual environment using a co-simulation platform. It can be further split into two parts. One covers the execution and setup of the simulation toolchain, while the other focusses on the creation of a realistic 3D environment, that includes variable elements that can be configured by the RLA.

2.1.1 Environment Simulation using a Co-Simulation Platform

This section will describe every component, that is used to gather simulation data. This includes components that are used in the PT, as well as in the DT, such as the virtual 3D Environment and Scenario Playback, as well as components that are used for training purposes, such as the vehicle dynamics, driving function, perception and sensor. During operation, these components could however serve as models for the PT-connected DT and could be synchronized with their physical counterparts on the testbed. Further concepts on the synchronization of DT and PT can be found in the ASIMOV Deliverable D4.1.

2.1.1.1 Scenario Engine

Every traffic scenario relies on the definition of a certain road layout, as well as the description of how all traffic participants interact with each other. Both of these descriptions can be realized in standards provided by ASAM. The ASAM OpenX Standards rely on an xml-based description language. The definition of the road layout can be done using OpenDRIVE. Here, all the streets, lanes, junctions, etc. can be defined. In the UUV use case, the OpenDRIVE file is directly created together with the 3D environment itself. The positions of the vehicles and other traffic participants can be assigned to the roads and lanes, that are defined in the OpenDRIVE file. Triggers can be defined inside an OpenSCENARIO file as a set of conditions, that need to apply, for certain e.g., speed or lane changes to happen. With multiple of these conditions in place, complex traffic situations can be created.

To playback the behavior of all these traffic participants, a program is needed that can interpret OpenSCENARIO and OpenDRIVE files and move the traffic participants accordingly. In this project, we are using esmini [10] to serve this purpose. The creation of such a scenario that complies with the respective standards is supported by the AVL Scenario Designer, which was used in this project. The road network in form of an OpenDRIVE file is created during the 3D environment creation. Section 2.1.2 provides details on the creation process.

The creation and playback of an OpenSCENARIO on its own, would of course not be interactive, as all the movements are predefined by the triggers. The ego vehicle shall therefore not be controlled via the OpenSCENARIO description, but rather by a vehicle dynamics physics engine, that is controlled via an autonomous driving function. The position and orientation of the ego vehicle will therefore be overwritten. This serves the additional benefit, that the movement of all traffic participants, including the ego vehicle, can be tracked via the scenario engine. The log of all the movements is tracked in the Open Simulation Interface (OSI) format and serves as basis for the sensor engines vehicle movements as well as ground truth for calculating Criticality KPIs as well as comparing percepted and ground truth vehicle position.

2.1.1.2 Sensor Engine

The sensor models are directly integrated into the Unreal [11] environment of Carla [12]. Currently, a virtual camera for lane detection is attached to the vehicle. Its properties in terms of Field-of-View, focal length, aperture, sensor dimension and resolution represent a camera used in an AVL Test Vehicle. The camera sensor and its resulting video stream are used for lane detection in this use case. More complex applications, that also rely on sensor fusion would be possible, but are – as the focus of this Use Case is to optimize the testing and not the vehicle and its driving function – not necessary at this point.

Besides the camera, a LiDAR model is implemented, to create virtual point cloud data. The LiDAR implementation consists of a ray casting function, that creates datapoints when hitting nearby objects. The rays are sent out angular equally distributed. The resolution of the sensor can be configured by setting the angular distance and vertical line count. The model does not (yet) take unequal point distribution, or the reflection physics of air and different surfaces or the rotation of the sensor into account. A more detailed LiDAR model implementation, based on the Carla sensor model is currently being paired with an Object detection algorithm. Refer to section 4 for details.

After further discussions, some of these properties might be included. However, an all-encompassing representation of the sensor or of the environment is not needed and even counterproductive. E.g., already with a slight rotation of a LiDAR sensor, the resulting point cloud data is unusable due to

calibration loss. The physical sensor needs to be deactivated or ignored until realignment; a digital representation of this case is not necessary. Another example is the sensitivity of sensors. An increase of sensitivity of actual LiDAR sensors might not be pushed to higher resolutions or to distinguish (slightly) different reflective surfaces or variations due to different air properties. This would increase the amount of resulting data, which is already in the ballpark of several TB per hour, causing problems for data processing and storage. Therefore, including the mentioned properties into the digital twin would increase its complexity and consume resources without further benefit.

Furthermore, a radar sensor is implemented. It is configured in a similar fashion to the LiDAR sensor, but directly outputs the distance and velocity of the object it hits, instead of raw point cloud data. It is used as a data source for the Adaptive Cruise Control (ACC) and Emergency Braking (AEB) function, included in the Driving Function.

The 3D environment is loaded as modeled, with all dynamic traffic participants being continuously updated in their position and orientation according to the scenario engine.

2.1.1.3 Driving Function and Perception

The Driving Function and Perception can be seen as the two components that translate everything seen in the environment into concrete actions on how to move inside that environment. The Driving Function provides basic Adaptive Cruise Control (ACC), as well as Autonomous Emergency Braking (AEB) and Lane Keep Assist (LKA) functionality. These driving function elements provide enough functionality to test the desired scenario variation effects, while still being easy enough to allow for debugging. It requires relevant information about perceived vehicles, as well as information about the detected lanes to function properly and output steering, as well as accelerator and brake pedal output.

The perception is divided into two parts. ACC and AEB both require information about the current target vehicles' relative position to the ego vehicle, as well as its relative speed. All this information is extracted from object lists, generated by the virtual sensor setup, or – for debugging purposes – from the ground truth data provided by the Scenario Engine. The Perception therefore looks at the provided object lists and identifies which of the vehicles are either currently directly in front of the ego vehicle or are about to cross its current trajectory. Based on this approximate calculation, the most relevant vehicle is selected and its distance as well as speed are forwarded to the Driving Function.

The Lane Detection forms the second part of the Perception. It uses an image stream, provided by a virtual camera sensor, to detect the lane markings and calculate the relative distance of the ego vehicle to these lanes.

2.1.1.4 Vehicle Dynamics

Especially when the ego vehicle enters highly dynamic driving situations, like an emergency braking, a detailed model of its physically correct driving behavior becomes very important. In the ASIMOV project, we use a dedicated vehicle dynamics simulation as Physics Engine for that. It provides the possibility to define the kinematics of the vehicle's suspension, weight distribution, tire properties among other relevant parameters of the vehicle simulation model. Its outputs not only provide necessary input to calculate the grip of every tire, but also the resulting position, orientation, and movement of the vehicle chassis. This information is then used by the Scenario and Sensor Engine, which move the ego vehicle accordingly. As the sensor is mounted on that vehicle, it directly influences the perception.

2.1.1.5 Co-Simulation

The Co-Simulation environment is used to bring together all the different components that are necessary to simulate a vehicle together with its driving function, sensors, vehicle dynamics and environment. It has the purpose of managing interfaces between different tools and taking care of the timing, so that every part of the simulation can run with consistent data. Furthermore, the signals that are exchanged between different components can be logged and stored together, without having to depend on logging possibilities of individual tools.

An important aspect of the co-simulation environment is also the variety of tools that can be integrated in that co-simulation.

2.1.2 3D Environment

2.1.2.1 3D Environment Variation Pipeline

The goal of the 3D Environment Variation Pipeline is to create a 3D environment that can be used in the simulation environment CARLA. [12] The variations which are performed are controlled by a JSON file, which is provided by the RLA before the pipeline run begins. The 3D Environment Variation Pipeline consists of the following four steps, which can also be seen in Figure 3:

Figure 3 - 3D Environment variation pipeline

2.1.2.1.1 Step 1: Variations process

The aim of the 1st step of the variation pipeline is to create a Triangraphics (TG) project file that contains all the information for a particular varied 3D environment that can be used to create intermediate files. Therefore, a configuration file, a variation file and a TG template project file are passed to a variation process.

The configuration file contains information that does not change during the learning process and remains constant for all pipeline runs. This file can be used, for example, to set the level of detail of objects.

The variation file contains information about the changes that should be applied to the 3D environment for a pipeline run. The variation file is created by the RLA and its values describe a specific 3D environment. The variation file can be used at any time to reproduce the 3D environment.

The TG template project file, which serves as a template for the resulting project file. In the template file, for example, objects are defined that do not change during the variation (e.g. houses, road network).

2.1.2.1.2 Step 2: Intermediate data generation

The aim of step 2 is to generate intermediate format files from the TG project file, which will be used for further processing in the next step.

For the generation of these intermediate formats, the TG project file created in step 1 serves as input for a generation process. Which intermediate formats are generated is defined by the TG project file. In the context of ASIMOV, the intermediate formats OpenDRIVE and glTF are used.

OpenDRIVE is mainly used to describe the road network. The glTF format mainly contains information about the visual appearance of the 3D scene. This includes, for example, 3D objects as well as their material description.

2.1.2.1.3 Step 3: intermediate data import

In step 3, the generated intermediate data from the previous step are imported with the use of the Unreal Editor. The import is done through the Unreal Editor using script files called blueprints. During the import of the intermediate data, various optional adjustments can be made to improve the appearance of the 3D environment as well as to improve performance in the simulation. These include the merging of materials and the settings for level streaming and collisions.

2.1.2.1.4 Step 4: Destination data export

In step 4, the data imported in the previous step is exported through Blueprints using the Unreal Editor. The result of the export is a CARLA world file that can be read into the CARLA simulation environment. The CARLA world file contains all the necessary information for the varied 3D scene.

2.1.2.2 3D Environment Variation

The variation is controlled via a JSON file, which is generated by the RLA. The JSON file contains complex structures that can vary different aspects of the 3D environment. In the following, the structures that control the variation are examined in more detail. Figure 4 shows the content of an example JSON file.


```
{
    "StaticModel" : {
        "Variation" : {
            "MoveObjects" : [
                {
                  "ruleID":1,
                  "s":0.99,
                  "t":0.5,
                  "side":0.0,
                  "a":0.3,
                  "i":0.4
                }
            ],
            "Lantern" : [
                {
                  "ruleID": 1,
                  "start" : 0.3,
                  "end" : 0.3,
                  "number": 0.5,
                  "i": 0.4
                }
            ],
            "Trees" : [
                {
                  "ruleID": 1,
                  "start" : 0.3,
                  "end" : 0.3,
                  "number": 0.5,
                  "i": 0.4
                }
            ],
            "ObjectsDensity" : [
                {
                  "ruleID":1,
                  "i":0.6,
                  "density":0.3
                }
            ],
            "Patches" : [
                {
                  "ruleID":1,
                  "p":0.1
                }
            ],
            "Marking" : [
                {
                  "ruleID":1,
                  "v": 0.1
                }
            ],
            "Signs": [
                {
                  "ruleID":1,
                  "x":0.8,
                  "y":0.3,
                  "i":0.4
                }
            ]
        }
    }
}
```


Figure 4 - Content of an example JSON variation file

2.1.2.2.1 Object placement on the road

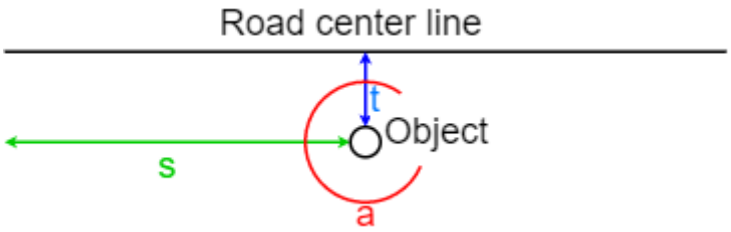


Figure 5 - Schematic representation of the placement of an object on the road

A schematic representation of the object placement with its parameters can be seen in Figure 5.
The structure for placing objects on the street contains the following parameters that can be used to control the variation:

Table 1 - Parameters for placing objects

Parameter	Description
ruleID	The ruleID indicates which predefined rule should be applied to a vector. A rule contains, for example, which car should be placed.
i	The parameter i determines a vector on which the variation will be applied. In the case of object placement, a vector is the center line of a road.
s	The parameter s determines the positioning of the object along the road.
t	The parameter t determines the lateral positioning of the object on the road.
side	The parameter side determines on which side of the street an object is placed.
a	With parameter a, the orientation of the object can be varied. This means that the object can be rotated around its center.

2.1.2.2.2 Tree row variation

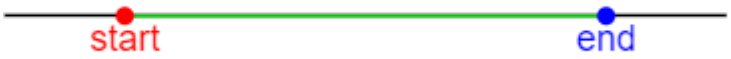


Figure 6 - Schematic representation of the variation of the tree row

Figure 6 shows a schematic representation of the placement of trees in a tree row. A certain number of trees are placed between start and end. The variation can be controlled with the following parameters:

Table 2 - Parameters for tree spacing

Parameter	Description
ruleID	The ruleID indicates which predefined rule should be applied to a vector. A rule determines, for example, which tree object should be used for the placement.
i	The parameter i determines a vector on which the variation will be applied. In this case, a vector is a reference line on which the trees are placed.
start	The start of the tree row can be varied with the parameter start.
end	The parameter end determines the end of the tree row
number	The parameter number controls how many trees are placed between start and end.

2.1.2.2.3 Tree area variation

The density of a forest can be varied by parameters. Trees are placed within an area described by a vector. The following parameters can be used to control the variation of trees within an area:

Table 3 - Parameters for tree density

Parameter	Description
ruleID	The ruleID indicates which predefined rule should be applied to a vector. A rule determines, for example, which tree object should be used for the placement.
i	The parameter i determines a vector on which the variation will be applied. In this case, a vector is an area in which the trees are placed.
density	The parameter determines the density of a forest within an area.

2.1.2.2.4 Lantern variation

Figure 7 - Schematic representation of the variation of lanterns

Figure 7 shows a schematic representation of the placement of lanterns. A certain number of lanterns are placed between start and end. The variation can be controlled with the following parameters:

Table 4 - Parameters for lantern spacing

Parameter	Description
ruleID	The ruleID indicates which predefined rule should be applied to a vector. A rule determines, for example, which lantern object should be used for the placement.
i	The parameter i determines a vector on which the variation will be applied. In this case, a vector is a reference line on which the lanterns are placed.
start	The parameter start determines where on a vector, which serves as a reference line, the placement of the lanterns begins.
end	The parameter end determines where the lantern placement ends.
number	The number parameter controls how many lanterns are placed between start and end.

2.1.2.2.5 Road sign variation

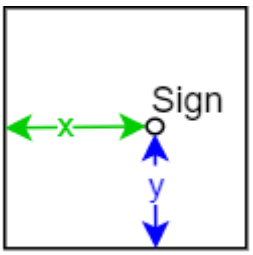


Figure 8 - Schematic representation of the variation of a road sign

Figure 8 shows the schematic representation of how a street sign is placed. A list of the parameters to control the variation can be seen in the table below:

Table 5 - Parameters for traffic sign placement

Parameter	Description
ruleID	The ruleID indicates which predefined rule should be applied to a vector. A rule determines, for example, how large the surrounding square is in which the traffic sign is allowed to move.
i	The parameter i determines a vector on which the variation will be applied. In this case, a vector is a point that represents a sign.
x	The parameter x is used to place a road sign along the x axis within a quadrate that restricts the movement space.
y	The parameter y is used to place a road sign along the y axis within a quadrate that restricts the movement space.

2.1.2.2.6 Road patches variation

Patches (e.q. manhole cover, tire tracks) placed on the street can be switched visible with the parameters of the following structure:

Table 6 - Parameters for patch placement

Parameter	Description
ruleID	The ruleID indicates which predefined rule should be applied to a vector.
p	The parameter p determines how many of the total number of patches are visible. A patch can be, for example, a manhole cover on the road or tire tracks.

2.1.2.2.7 Markings

The visibility of markings on the road can be varied with the parameters of the following structure:

Table 7 - Parameters for visibility of road markings

Parameter	Description
ruleID	The ruleID indicates which predefined rule should be applied to a vector.
v	The parameter v controls the visibility of the markings.

2.2 Workgroup: Feature Engineering

The feature engineering (FE) workgroup represents the bridge between raw simulation data and useful features for the RLA to base its decisions about future environment variations on. FE is furthermore used to guide the variations in a specific direction, depending on the desired focus of the tests. The process of creating these features can include expert knowledge.

As the goal of the Use Case is to provide safety critical and diverse Environment Variations, we can divide the FE into two main parts. One covers the Quantification of Safety Criticality via appropriate KPIs, while the other quantifies the diversity of the resulting simulation data.

2.2.1 Criticality KPIs

The criticality KPIs provide the means to quantify if a traffic scenario was critical or dangerous. This helps in identifying scenarios, which are interesting in terms of validating a driving function from a safety or comfort point of view and help to improve the vehicle and its driving function. As there are different types of KPIs, we identified two major safety groups, which we wanted to quantify. The driving function used in this project, can be roughly divided into a lateral and a longitudinal controller and the KPI groups reflect that. An overview of possible KPIs can be found in [13].

Based on the driving function and the testing scenarios, the multiple KPIs have been selected.

For **KPI Group 1 (AEB/ACC)** the following metrics can be used:

- Time to React
  - Minimal Time to Maneuver (TTM): Time to Brake (TTB), Time to Steer (TTS), Time to Kickdown (TTK)
- Deceleration to Safety Time
  - Required Acceleration to maintain a set safety time distance
- Required Longitudinal Acceleration => Required Acceleration
  - Required Longitudinal Acceleration, to avoid an accident
- Brake Threat Number
  - Required Amount of Deceleration in comparison to the vehicle's maximum capabilities

For **KPI Group 2 (LKA)** the following metrics can be used:

- Required Latitudinal Acceleration => Required Acceleration
- Area out of Lane
- Steer Threat Number
  - similar to Brake Threat Number in lateral direction

For each of the KPIs, a respective reference value is also added to normalize the resulting value. Multiple of these KPIs can then be aggregated via a weighted sum, in which the weights offer the possibility to decide how important a specific KPI is in a scenario.

An overview of the proposed workflow can be seen in Figure 9. This overview also includes the anomaly detection, which is presented in the next section.

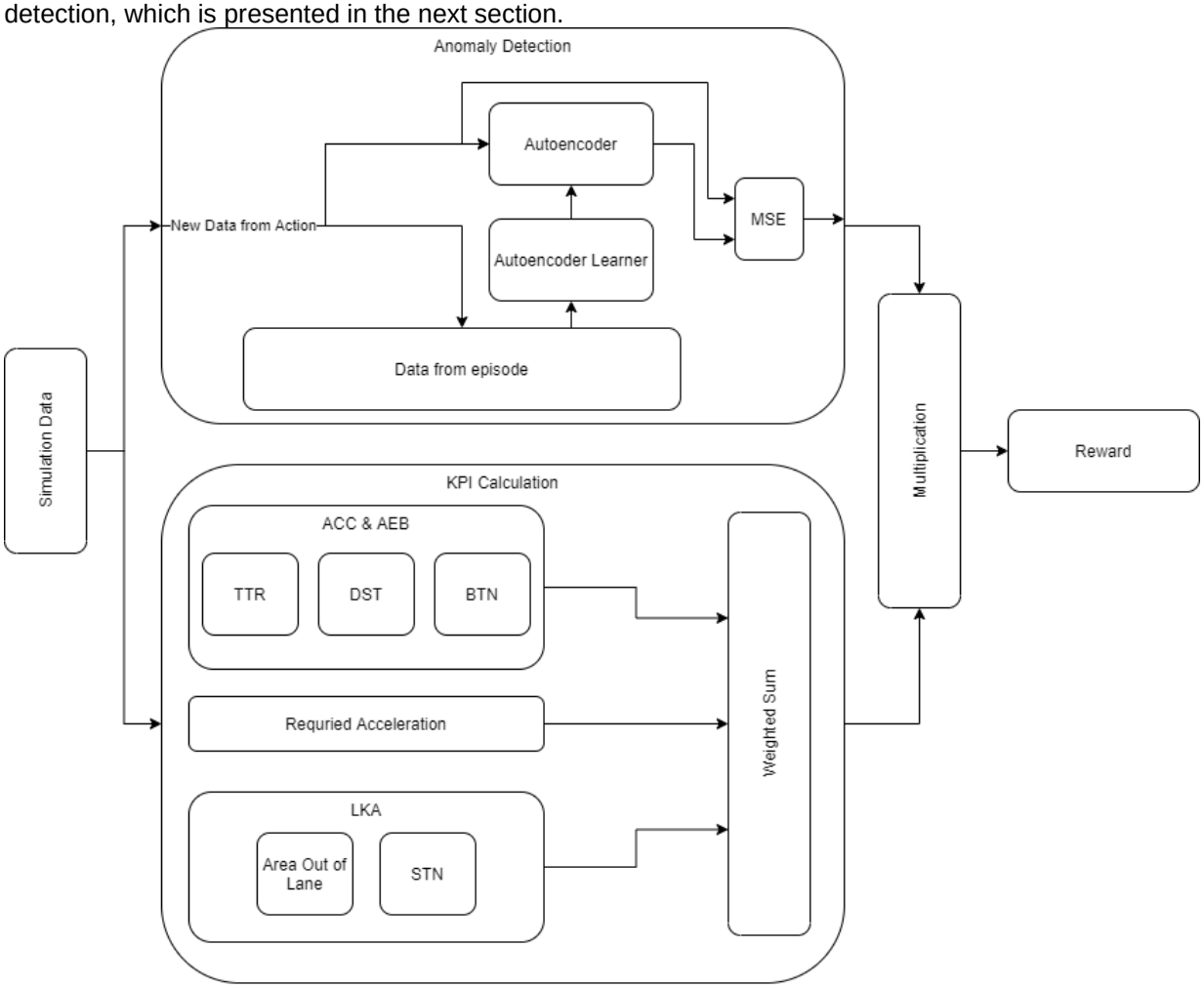


Figure 9 - Overview Feature Engineering in UUV Use Case

2.2.2 Diversity Quantification

Originally, it was planned to also include a quantification of the novelty into the state of the RLA. But as a built up of knowledge during the RL process leads to either a very large state space, or the violation of the Markov property, we have decided to exclude the novelty quantification from the state and reward. Instead, the Novelty Quantification can be used as a standalone tool, to analyze already recorded data and identify the most interesting sections. A Detailed example can be found below.

To achieve such novelty quantification, an auto encoder neural network shall be used. The autoencoder network comprises an encoder and decoder part. The encoder compresses high dimensional input data into lower dimensional compressed data, while the decoder uses this compressed data to try to reconstruct the original high dimensional data, with as little deviation as possible. Encoding and decoding happens through multiple layers of artificial neurons. Encoding starts with an input layer, where the number of neurons is defined by the input vector size. Following “deeper” layers contain successively less neurons to achieve the information compression effect. The decoder then reverses that technique and contains multiple layers with an increasing number of neurons. The output layer then has the same number of neurons as the input layer.

A typical use case for autoencoders is therefore data compression or feature extraction from high dimensional inputs such as images. To extract the most relevant features, a diverse, but representative set of training data is required. The more diverse the dataset is, the harder it becomes for the autoencoder

to find a common compression technique, resulting in bigger losses, when reconstructing the input data in the decoder.

The sensitivity of the reconstruction loss to unseen and fundamentally different data compared to training data, makes the autoencoder suitable to quantify data similarity. In the case of the UUV use case, data from multiple simulation runs can be seen as a sequence of independent vectors, with the size equaling the number of recorded data channels. A novelty quantification can therefore not only be provided for an entire simulation run, but rather for the individual timesteps of such a run. The overall novelty value can then be calculated by using the sum, average or any other aggregational method across all timesteps of the simulation run.

After the quantification is done, the neural net needs to be adapted, so that it incorporates the new information, provided by the latest dataset. The new data points therefore are added to the common pool of training samples. A retraining of the neural network using samples from this pool then incorporates the new information into the autoencoders internal structure. The limited number of neurons and therefore limited capacity of the neural network itself, leads to generalization. The difference in reconstruction error between “expected” and “unexpected” samples is however still great enough to easily identify novel datapoints.

2.2.2.1 Novelty Quantification Demonstrator

For the UUV use case, a small demonstrator for quantification of these novelty values has been created. For demonstration purposes, demo data has been gathered in a vehicle dynamics simulation. It features the following simulation runs, each containing variable simulation duration and 56 data channels:

- 1 Lap around the AVL test track in Gratkorn (two left-hand corners connected via two straights)
- 1 Lap around the Hockenheimring racetrack
- Driving in a circle with increasing velocity
- Acceleration followed by travel at constant speed
- Acceleration followed by left-hand turn
- Acceleration followed by right-hand turn

The three acceleration simulations are similar during the acceleration phase of the simulation run. This offers the possibility to easily identify if similar phases of simulation runs are distinguishable from novel parts.

The order in which the novelty quantification evaluates the datasets has been varied in the different examples to show the effect on the quantification. The autoencoder was always trained until convergence on the Hockenheim racetrack dataset, to provide a foundation that covers a wide variety of dynamic driving situations.

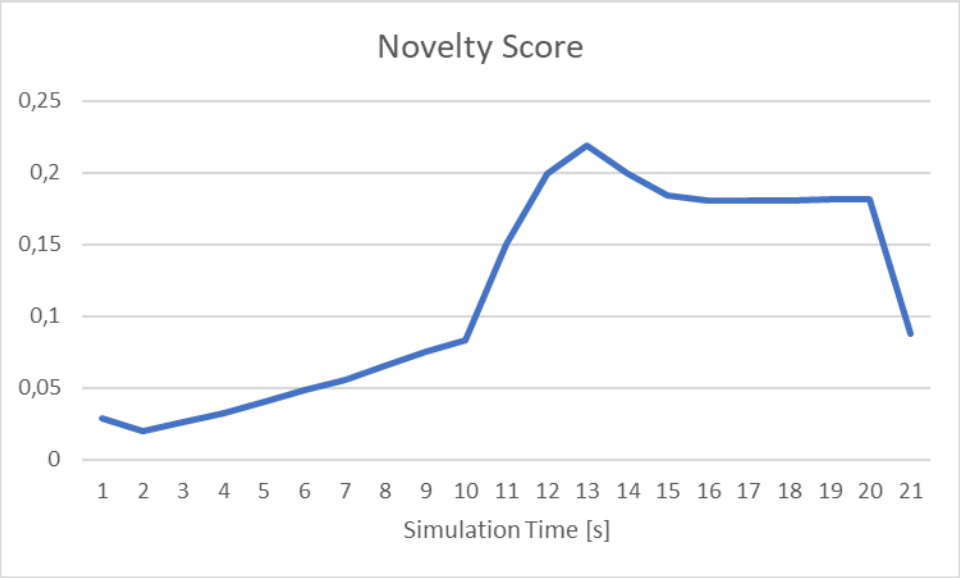


Figure 10 - Novelty Detection - Holding Speed Dataset

When evaluating the first dataset of the vehicle accelerating and holding a constant speed, it can clearly be seen, that the acceleration phase until the 10 second mark, features a lower novelty value than the phase of constant speed after that. This can be seen as a result of the racetrack basis dataset, where traveling at constant speed was not performed.

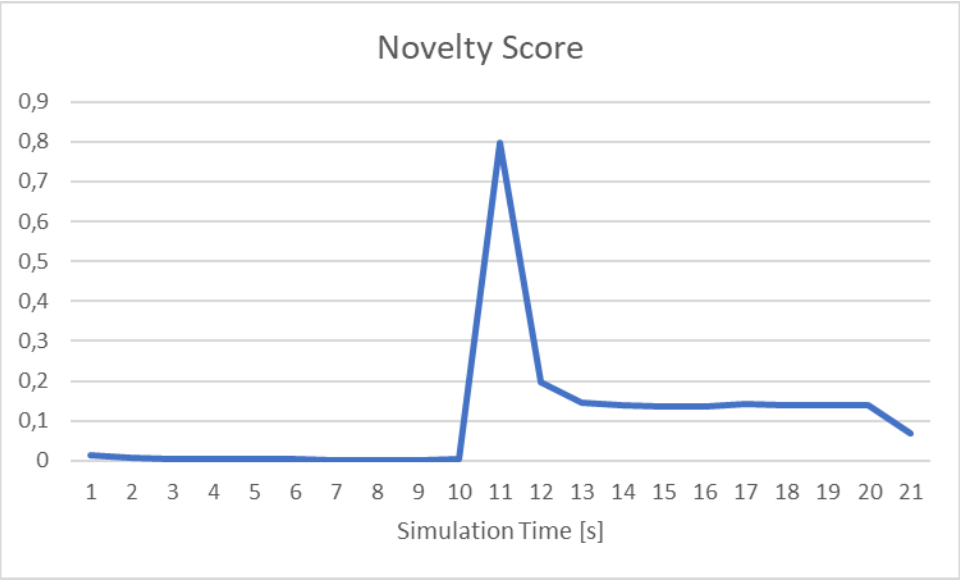


Figure 11 - Novelty Detection - Sharp Turn Dataset

The next simulation run contained an acceleration, followed by a sharp turn. It can be seen, that the acceleration phase has very low novelty values, as it is similar to the one seen in the sample before. As soon as the sudden turn in happens, the novelty value rises, but reduces again, during the constant steering phase. The novelty values correlate with the subjective opinion on which part of the simulation is interesting quite well.

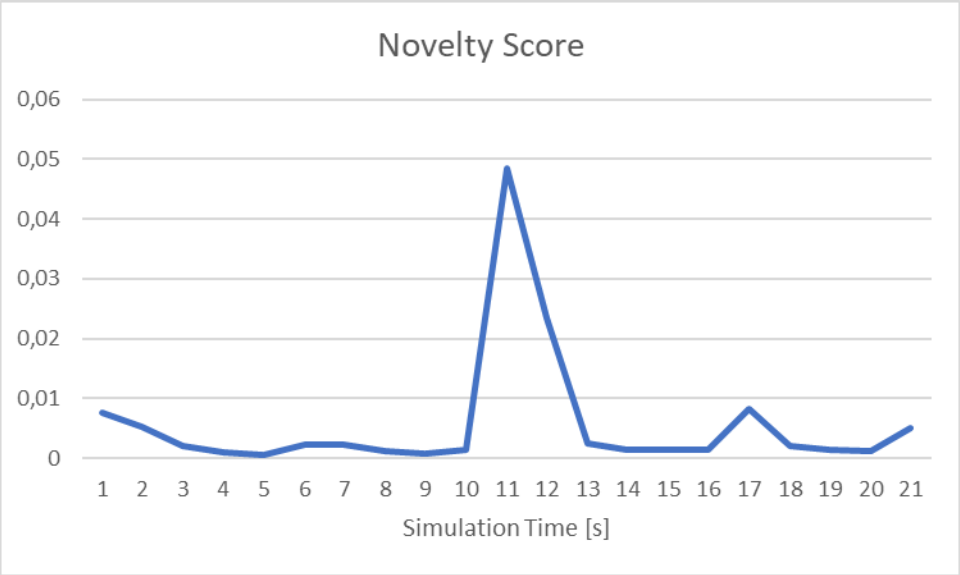


Figure 12 - Novelty Detection - Sharp Turn Dataset 2

Providing the autoencoder with the same sample again, shows again a spike at the 10 second mark, but overall provides much lower novelty values across the whole simulation time.

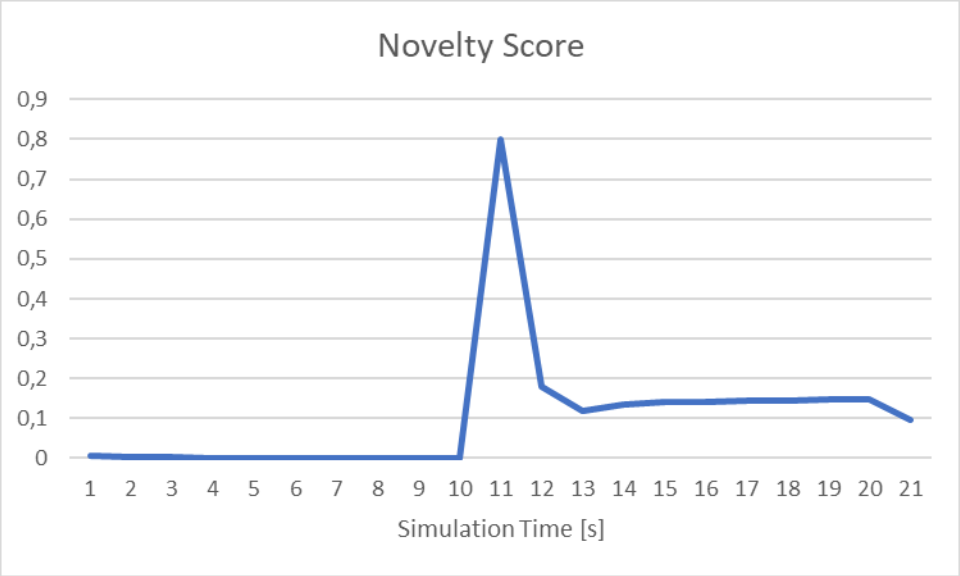


Figure 13 - Novelty Detection - Sharp Turn Different Direction

Changing the direction of Turn in from a left- to a right-hand turn, also quantifies in the expected way. The phase of constant acceleration has a novelty value of almost 0, while the values after turn in, are quite similar to the first turn in example.

When checking the anomaly results per data channel, a more detailed analysis can be done. A visualizing diagram is shown in Figure 14.

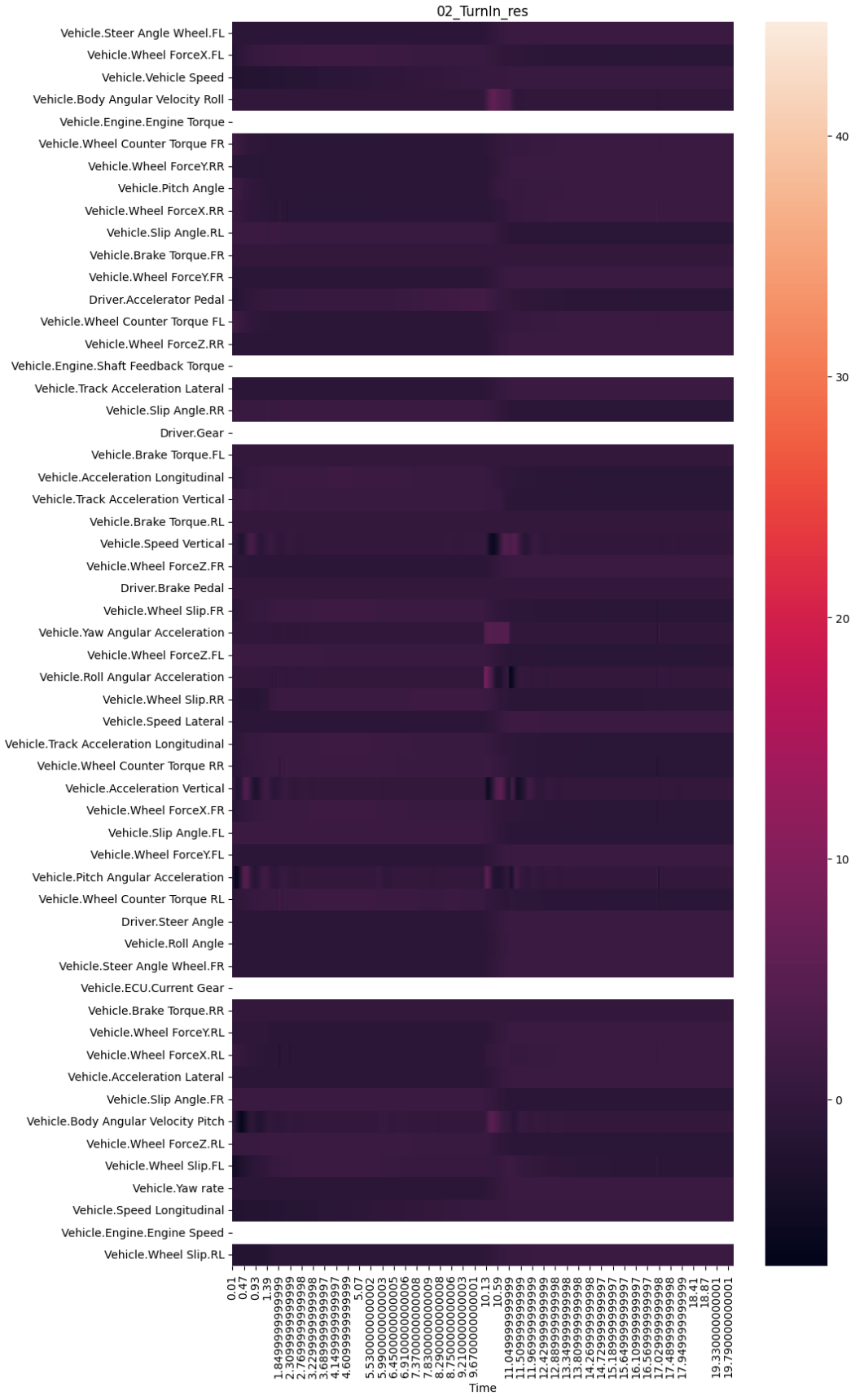


Figure 14 - Channel-specific analysis of contribution to anomaly score

2.3 Workgroup: Reinforcement Learning

The workgroup “Reinforcement learning” is concerned with the development of the RLA which implies its training and application. From a RL standpoint of view and as already introduced in 1.4, the RLA interacts with an “environment” by state, rewards, and actions (see [5]). It received the two previous from the Feature Engineering while sending the latter to the Environment Simulation (Scenario Generator). In the following the current status of RLA’s progress is shown.
The development follows an iterative process. First, requirements are elated based on the needs and possibilities given by the Use Case and the general ASIMOV solution. In continuation an algorithm is selected which meets these requirements and a first prototype is implemented. Based on knowledge that is gained by testing the initial version, refinements and additions will be made.

2.3.1 Requirements elicitation

The following requirements were elated by continuous discussion between the partners involved. In several sessions, questions that had been raised over the course of the project were collected, addressed, and answered. In the following the resulting requirements are categorized into (I) System/functional requirements, (II) external interface requirements, and (III) non-functional requirements. If all are met, the RLA should correctly interact with the adjacent system, produce good optimization suggestions, in a reasonable way.

2.3.1.1 System/functional requirements

Table 8 – Functional Requirements on the State

Req. number	Name	Description
G.1	Episodicity	The application of the RLA is structured in episodes. The characteristics of the vehicle change slightly, but we stay inside the ODD.
G.2	Complex environment	We need to assume that the system the RLA shall optimize is highly complex. Thus, the algorithm needs to be capable to deal with this level of complexity.
G.3	Distributed learning	Ideally, the algorithm is capable of being trained in parallel.

2.3.1.2 External interface requirements

Table 9 - External Interface Requirements on the State

Req. number	Name	Description
S.1	State definition	The state shall consist of criticality metrics (S.3), and basic vehicle configurations (S.4).
S.2	State transition	In each iteration, the criticality values and anomaly values of the new scenario are summed to the old state values.
S.3	Criticality values	Each criticality value represents one unique criticality metric. Which are used is to be determined in the FE workgroup.
S.4	Basic vehicle configurations	Basic vehicle configurations are used to give the RLA some basic knowledge about the vehicle. This information shall be as restricted as possible. Examples are vehicle weight, …
S.5	Number of states	The state space is continuous.

S.6	Initial state	The first state is blank (values = 0) except basic vehicle configurations.

Table 10 - Requirements on the Reward

Req. number	Name	Description
R.1	Input	The RLA shall be able to take one numerical variable as the reward.
(R.2)	Weighted sum	The reward shall be a weighted sum out of all KPIs.
(R.3)	KPIs	The KPIs is first a criticality metric. Later it is a combination of criticality metrics and anomaly value.

Table 11 - Requirements on the Action

Req. number	Name	Description
A.1	One action = one scenario	In the first step, one action of the RLA shall be the input for one scenario.
A.2	Systematic scenario errors	The possible actions shall be designed in such a way that any parameter combination produces a valid scenario.
A.3	Parameters	First: Static environment parameters (time of day, …) Later: More parts of the static environment + parts of the dynamic environment.
A.4	Output type	The actions shall support both discrete and continuous values.

Table 12 - Requirements on Diagnostics

Req. number	Name	Description
D.1	Basic diagnostics	The implementation should be capable of logging the main KPIs to track training progress.

2.3.1.3 Non-functional requirements

Table 13 - Non-functional Requirements

Req. number	Name	Description
N.1	Language	The RLA shall be written in python for easy implementation and access to state-of-the-art RL packages.
N.2	First step I/O	The RLA shall store a file with its actions and receive the state and reward as files in a folder as well.
N.3	Data efficiency	The algorithm shall be data efficient.

2.3.2 Introduction to selected algorithm

From all requirements, three stand out the most for being restrictive to the selection of algorithms: (I) Data efficiency, (II) Distributed Learning, and (III) support for discrete and continuous actions. While there is a multitude of RL algorithms and many would be capable of fulfilling the requirements, this section

introduces the selected algorithm. For a more thorough explanation of the decision process consult deliverable D3.3 of the project.

Degrave et al. [14] recently introduced a deep RL-designed magnetic controller for tokamak plasma (nuclear fusion technology) which is learned by interacting with a simulated environment and subsequently applied to the physical system of the tokamak in a zero-shot fashion. The approach comprises an actor-critic framework to learn appropriate voltage control commands, based on the current plasma state and control targets.

While the applications are not the same on first sight, it shows parallels to the UUV UC. On one hand a system needs to be controlled/optimized that is highly complex. On the other hand, due to the high costs of running a fusion reactor, a simulated environment is needed for training. The most relevant aspects of the approach are the following:

- Using an asymmetric actor-critic framework with large recurrent critic neural network (NN) to compensate for the non-Markovian properties of the environment and relatively small feedforward actor NN;
- Learning loops for episodic RL, using a distributed architecture with a single learner instance and several actors each running an independent instance of the simulator;
- Applying the Maximum a Posteriori Policy Optimization (MPO) algorithm by [15] as RL approach, and possibly combining this with relative entropy regularized policy iteration [16].

This indicates that MPO is a good candidate for fulfilling the requirements given by the UC except of providing native support for both discrete and actions. In order to fulfill this, the MPO algorithm is expanded by the work outlined in [17]. Without going much into detail, it is not based on a paradigm conversion of e.g., discrete to continuous actions but rather sees the policy as a combination of a policy with discrete and continuous actions.

$$\pi(a|s) = \pi^c(a^C|s)\pi^d(a^D|s)$$

The evaluation and training of both “sub”-policies happens simultaneously (see in following sections below).

2.3.3 Step-by-Step RLA training with MPO and implementation

In the following the MPO algorithm is explained by showing the implementation process step by step. For a more formal and in-depth explanation consider the aforementioned literature [16] [15] and [17]. In general, the training works as seen in Figure 15 in multiple iterations.

1. In each iteration step of “training iteration”, a number of trajectories are sampled and stored in a replay buffer.
2. Experiences are sampled from the replay buffer.
3. With these experiences, the critic is updated (considered step 1 according to [16]).
4. The E-step and M-step (both forming step 2 according to [16]) are performed.
5. Either the learning steps 2,3 and 4 are rerun for further improvement of the actor and critic or the improved actor and critic are set to be the target actor and critic. The target actor/critic is their respective version from the last training iteration. It is used as a benchmark for the improvement.

In the following, the steps are explained in more detail.

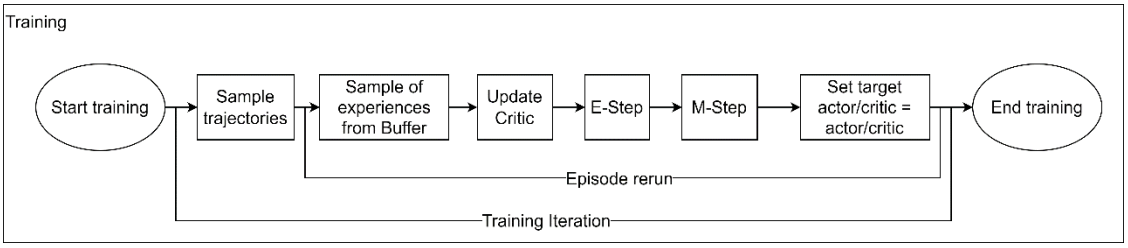


Figure 15 - Process diagram of the RLA's training

2.3.3.1 Sample trajectories

A trajectory is a sequence of experience, where the actor reiteratively suggests an action based on the state of the environment which consequently transitions to a new state and is evaluated with a reward. A trajectory is synonymous to an episode. The sampling of trajectories is straight forward and can be seen in Figure 16.

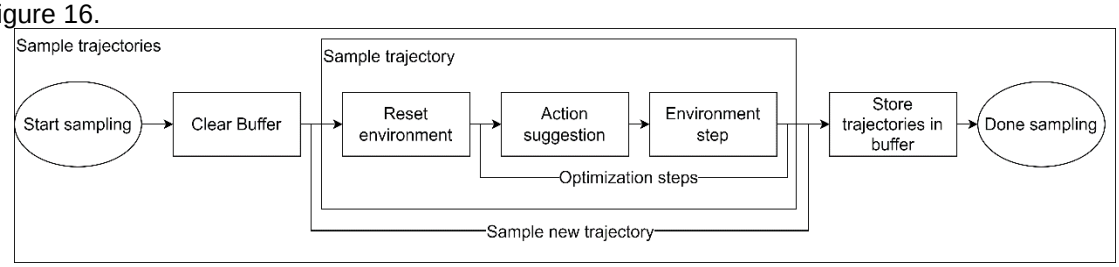


Figure 16 - Process diagram of the sampling of trajectories

The sampling of trajectories happens as follows:

1. At the beginning, the replay buffer is cleared in order to have only trajectories using the newest actor. This is actually not required by the algorithm since the off-policy is capable of dealing with samples from older RLA iterations but it is beneficial to the training time since not all trajectories need to be stored over all training iterations.
2. Sampling a number of trajectories is simply their execution in serial.
    a. The environment is reset to an initial state.
    b. Based on the given state, an action is sampled from the target actor which is the parameterized $\pi(a|s)$. Whether the actor or target actor is used, does not matter at this point since just previous the sampling of trajectories the actor and target actor have been set to be equal.
    c. In light of the action and previous state, the next state and reward of the last action is returned from the environment. If the goal of the optimization is not reached the next action is suggested.
3. All trajectories are stored in the episode buffer as experiences. In line with [18] we see the tuple s_t, a_t, r_t, s_{t+1} as an experience of timestep t.

2.3.3.2 Sample experiences from Buffer

In continuation to the sampling of trajectories, within each “episode rerun” B experiences are sampled from that buffer.

2.3.3.3 Update critic

The critic represents the parameterized approximation of the Q-function $Q(s,a)$. Its update is explained in more detail as the “Policy evaluation” (Step 1) in [16]. In essence, the critic’s NN’s parameters are optimized via gradient descent in such a way that the squared difference between the actual Q-value of the sampled actions and states and the estimated Q-value calculated by the critic is minimized. This implementation as shown in Figure 17.

1) Sample from the target actor I alternative actions based on the B next states s_{t+1} drawn from the buffer.
2) Get expected Q-value Q_{Enext}. From the target critic of state-action pairs which are the product of the B next states and I actions drawn in step 1. In addition, the mean over all Q-values from one next state is doneresulting in a vector of length B which represents the expected Q-value of the next states called
3) Calculate the “actual” Q-value Q_a of the performed actions a_t and states s_t by
$$Q_a = r + \gamma \cdot Q_{Enext}$$
with reward r and discount factor γ.
4) Use the critic to calculate the estimated Q-values Q_{est} of the state s_t and action a_t.

5) Calculate the loss of Q by the squared difference between Q_a and Q_{est} and apply gradient descent to minimize it.

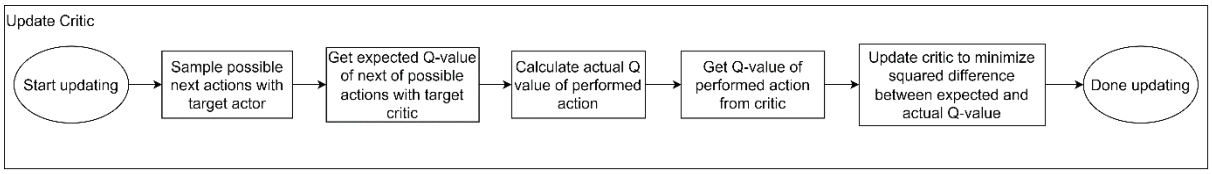


Figure 17 - Process Diagram of updating the critic

2.3.3.4 E-Step

The E-Step can be seen as the preparation for the following M-step. It is referred to by [16] as "finding action weights". Section 4.1 in [16] and section 3.2 in [15] explain this step in more detail, but essentially "we want to first re-adjust the probabilities for the given actions in each state such that better actions have higher probability. These updated probabilities are expressed via the weights q_{ij}…" [16]. As proven in [16], q_{ij} can be expressed by the soft-max over Q-values adjusted by the temperature η:

$$q_{ij} = q(a_i, s_j) = \frac{\exp\left(Q^{\pi^k}(s_j, a_i)/\eta\right)}{\sum_i \exp\left(Q^{\pi^k}(s_j, a_i)/\eta\right)}$$

With $i = 1 \dots N$, $j = 1 \dots K$, and Q^{π^k} being the Q-function (target critic). The temperature η which is a term introduced by solving the constraints on the weights which limits the change in policy (see Appendix D of [15]) can be found by solving the convex dual function:

$$\eta = \text{argmin}_\eta \, \eta\epsilon + \eta \sum_j^K \frac{1}{K} \log\left(\sum_i^N \frac{1}{N} exp\left(\frac{Q(s_j, a_i)}{\eta}\right)\right)$$

The heat factor η is a term introduced from solving the constraints on the weights which limits the change of the policy (see Appendix D of [15]). The equations are calculated as shown in Figure 18.

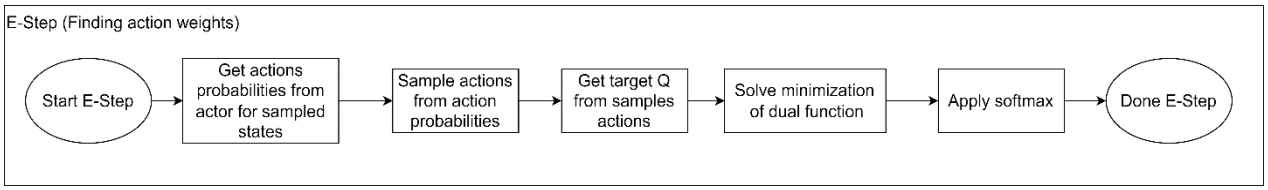


Figure 18 - Process diagram of the E-Step

1. K states s_j are sampled from the replay buffer and the distributions are obtained from the actor. Namely for continuous values, the vector of means μ of size equal to the number of continuous actions, the continuous actions covariance in form of A as the triangular matrix via its Cholesky decomposition, and the probability of options of the discrete actions p_{probs}.
2. Sampling of N actions a_i from the given probabilities.
3. Get Q-values for each pair of (a_i, s_j).
4. Find solution to η by minimizing dual convex function (see above).
5. Apply soft-max to by η adjusted Q-values (see step 3).

2.3.3.5 M-Step

The M-Step of the policy update is the iterative fitting of the policy represented by the actor to the re-weighted action probabilities. This is synonymous to the minimization of the Kullback-Leibler divergence (KL) between the two said probability distributions (sample-based distribution from the E-Step and the parametric policy represent by the actor).

$$\pi^{k+1} = \arg\max_{\pi_\theta} \sum_j \sum_i q_{ij} \log \pi_\theta(a_i|s_j)$$

As pointed out in [16]: “sample based maximum likelihood estimation can suffer from overfitting”. Thus, the change of the parametric policy (the actor) needs to be limited.

$$\sum_j \frac{1}{K} KL\left(\pi^k(a|s_j)||\pi_\theta(a_i|s_j)\right) < \epsilon$$

As shown in [16] this optimization in combination with a constraint can be solved using Lagrangian Relaxation. However, as we deploy an actor capable to provide continuous and discrete actions, the Lagrangian equation needs to be adapted according to [17]. Here the change of the parametric policy is limited in three separate ways: once for the categorical distribution of the discrete actions and for the means and covariance of the multivariate distribution from which the continuous actions are sampled each.

$$L(\theta, \eta_{c,\mu}, \eta_{C,A}, \eta_d) = \sum_j \sum_i q_{ij} \log \pi_\theta(a_i|s_j) + \eta_{c,\mu}(\epsilon_{c,\mu} - \mathrm{T}_{c,\mu}) + \eta_{C,A}(\epsilon_{c,A} - \mathrm{T}_{c,A}) + \eta_d(\epsilon_d - T_d)$$

with $\eta_{C,\mu}, \eta_{C,A}, \eta_d$, $\epsilon_{c,\mu}, \epsilon_{c,A}, \epsilon_d$, and $\mathrm{T}_{c,\mu}, \mathrm{T}_{c,A}, T_d$ corresponding to the Lagrangian multipliers, the allowed expected change of the policy in KL divergence, and the sample-based KL divergence respectively for continuous mean and covariance $(c, \mu \; or \; c, A)$ and discrete (d) actions. It is then solved as

$$\max_\theta \min_{\eta_{C,\mu}>0, \eta_{C,A}>0, \eta_d>0} L(\theta, \eta_{C,\mu}, \eta_{C,A}, \eta_d)$$

Iteratively solving the inner and outer optimization independently. This process is implemented as shown in Figure 19 as follows:

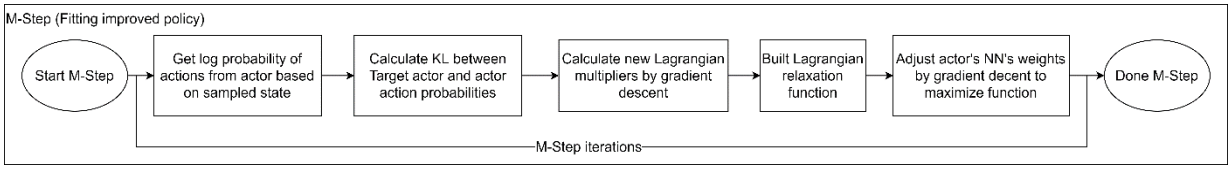


Figure 19 - Process Diagram of the M-Step

1. Get logarithmic probability (log prob) of each action from the current actor by getting μ, A and p_{probs}, convert them into torch distribution and obtain log prob.
2. Calculate average KL divergence of discrete actions (categorical KL) and continuous actions (Gaussian KL). The categorical KL is given by

$$\mathrm{T_d} = KL_D(p_1||p_2) = \frac{1}{K}\sum_{k=1}^{K}\sum_n {p_1}_{n,k} \log\left(\frac{{p_1}_{n,k}}{{p_2}_{n,k}}\right)$$

with K being the number of sampled states, n the number of options per discrete action, and p_1 and p_2 being the probabilities given by the actor and target actor respectively. The KL divergence of two multivariate Gaussian distribution, as proven in [19], can be expressed as follows:

$$KL_C(p1(\mu_1, A_1)|\,|p_2(\mu_2, A_2)) = \frac{1}{2}\left(\log\left(\frac{det\, A_2}{det\, A_1} - n + tr(A_2^{-1}A_1) + (\mu_2 - \mu_1)^T A_2^{-1}(\mu_2 - \mu_1)\right)\right)$$

Note that in this case, A is the covariance matrix instead of the Cholesky decomposition of the covariance as given by the actor’s implementation. In order to find the divergence specifically between either the means or covariances, the following is applied respectively.

$$\mathrm{T}_{c,\mu} = KL_C(p(x\,|\mu_1, A_1)||\,p(x|\mu_2, A_1)\,)$$
$$\mathrm{T}_{c,A} = KL_C(p(x\,|\mu_1, A_1)||\,p(x|\mu_1, A_2)\,)$$

3. Update Lagrangian multipliers by gradient descent according to

$$\eta_{C,\mu_{t+1}} = \eta_{C,\mu_t} - \alpha_{c,\mu} * \frac{\partial \mathrm{L}}{\partial \eta_{\mathrm{C,\mu}}} = \eta_{C,\mu_t} - \alpha_{c,\mu} * (\epsilon_{c,\mu} - \mathrm{T}_{c,\mu})$$
$$\eta_{C,A_{t+1}} = \eta_{C,A_t} - \alpha_{c,A} * \frac{\partial \mathrm{L}}{\partial \eta_{\mathrm{C,A}}} = \eta_{C,A_t} - \alpha_{c,A} * (\epsilon_{c,A} - \mathrm{T}_{c,A})$$
$$\eta_{d_{t+1}} = \eta_{d_t} - \alpha_d * \frac{\partial \mathrm{L}}{\partial \eta_{\mathrm{d}}} = \eta_{d_t} - \alpha_d * (\epsilon_d - \mathrm{T}_d)$$

Note that η_{C,μ_t}, η_{C,A_t}, and η_{d_t} are called α_μ, α_Σ, and α respectively. The corresponding α is given as α_*_scale in the code.

4. Build Lagrangian equation according to the function above.
5. Apply gradient decent to the Lagrangian equation. The optimization done within the M-Step is repeated a fixed number of times.

2.3.4 Actor description

The actor is represented by two parts: On one hand, it consists of a neural net which takes the states given by the environment and outputs the weights of options for each discrete action and the mean μ and covariance in form of its Cholesky decomposition A. This is shown as illustrated in Figure 20. On the other, the actor consists of simple transformation functions which take the values describing distributions from the neural net, create such distributions (see right side equations of Figure 20) as functions and sample the action from it.

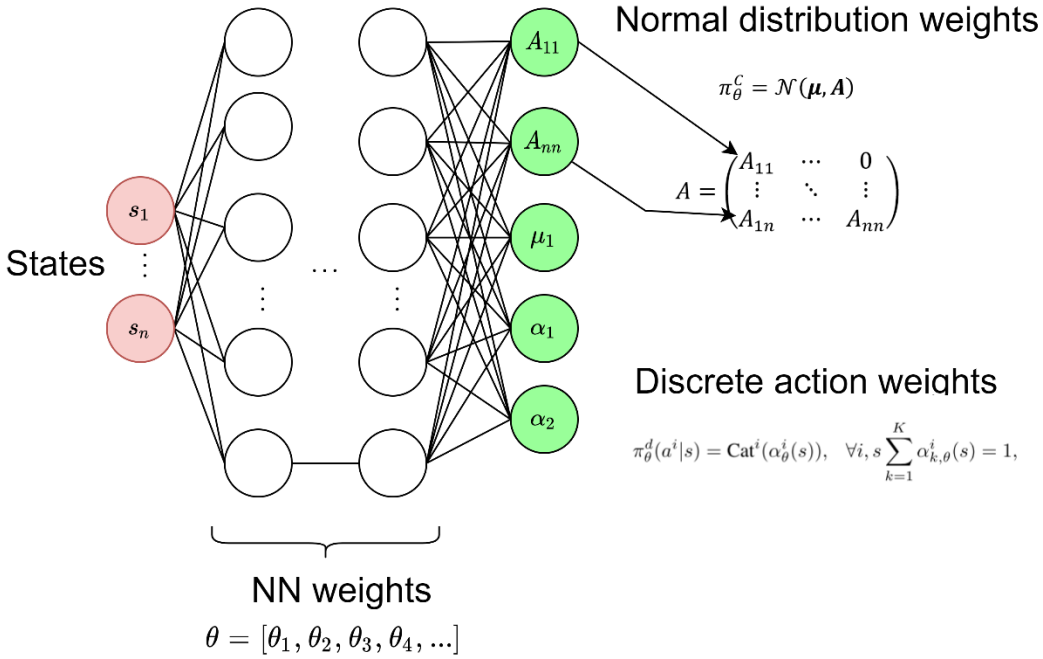


Figure 20 - Visualization of the actor's NN layout

2.3.5 Critic description

The critic is represented by a neural net similar to the actor. However, states and actions are used as inputs and result in a single output which represents the estimated Q-value (see Figure 21).

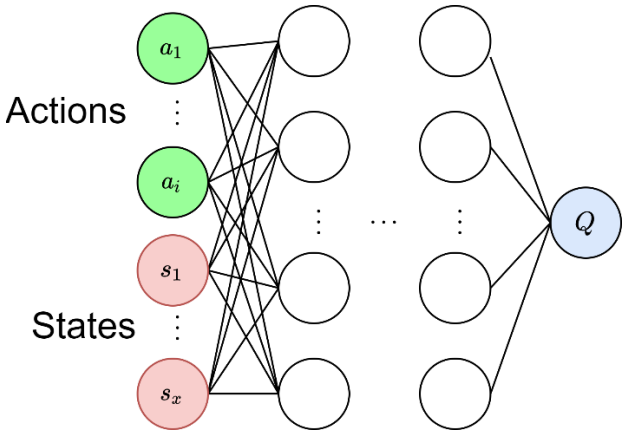


Figure 21 - Visualization of ciritc's NN layout

2.3.6 Performance verification

In order to test the correct functioning of the algorithm, a controlled and known environment needs to be used to isolate errors of the environment from a poor performance of the MPO algorithm. Therefore, a standard OpenAI gym library [20] is used to verify the correct training. Since it does not provide an environment which takes both discrete and continuous actions, a superposition of a discrete and continuous environment is used. For this, the "LunarLander" is used in its discrete and continuous action version. The observations are concatenated together while the resulting action are split again according to their nature.

As a base-line default values have been used (blue line in Figure 22) but also alterations in the training as well as of the actor's neural net were investigated. The tests have been run for 20000 training iterations but showed that already after 1000 runs (see figure Figure 22) the algorithm has converged to the maximum reward possible. This was repeated with different variation of training strategy as well as changes to the actor. The training results indicate that an increased re-use of the sampled data is highly efficient while an increased size of the actor's neural net is in this case unnecessary and has a negative impact on the training.

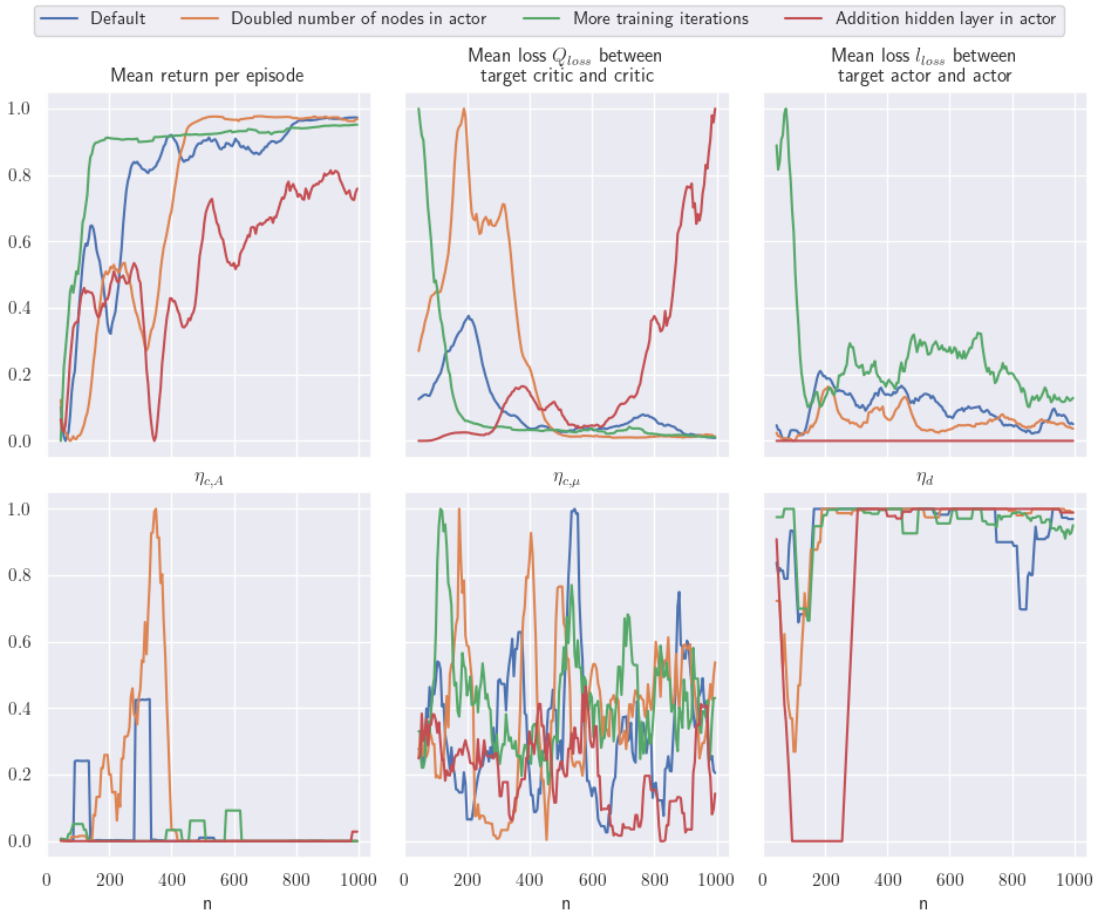


Figure 22 - Key performance indicators for training over the training iteration steps with different configurations of the training or actor

Mean return is the average of the sum of rewards over an episode, Q_loss is mean difference of Q-values from the critic and target critic (see 2.3.3.3). l_loss is the resulting average difference between the policy of the actor and the target actor (corresponding to the result of the Lagrangian equation of 2.3.3.5. The three Lagrangian multipliers for the covariance and average of the discrete actions and discrete actions (lower row, left to right) (see 2.3.3.5) indicate where points of improvement can still be found.

2.3.7 Action conversion

While the algorithm provides primarily continuous values between 0 and 1 or discrete options in terms of integers (0,1,2,3…), the scenario generator requires the actions to be in a specific format, range, and data types. Specifically, it requires the objects to be passed as a JSON5-file containing a hierarchy of information. Without going into much detail, the format and content of the file, which represents the action as the optimization of the scenario, is given by a template.
The hierarchy is built as follows. Recursively, every item given as a string (e.g. “StaticModel”) has content either as another dictionary containing items (curly brackets) or a list of items (squared brackets). An item can also have a value associated with it. Items with a discrete value (e.g. “ruleID”) have their options listed as a string. Item containing a continuous value (e.g. “s”) are represented by a list which indicated the lower and upper bound.
An algorithm recursively walks through the file, identifying the hierarchy and continuous and discrete actions (including the constraints). From there, the action space is determined. In order to provide the action and therefore the file, this algorithm is reversed to create the hierarchy and substitute the constraints with concrete values.

2.3.8 State and reward conversion

The state and reward are currently supplied via a file which contains a simple list of values. These are read by the algorithm after a new state is requested.

2.3.9 Future improvements

Since this is the first prototype, some aspect is simplified but can be improved in the future. After closing the training loop, integrating the Scenario generator and the Feature Engineering and testing the working of and successful training in a small case study, big improvements are expected for the following issues.

On one hand, with the goal of being able to have the majority of parameters optimized by the RLA during the optimization, it is easy to say that the dimensionality of the action space gets out of control fast resulting in inefficiencies. This is especially true when dealing with discrete actions with possibly hundreds of options. Therefore, new techniques need to implemented. An example is the work of [21] which deals with large discrete action spaces which motivates their work with the large systems like Youtube or Amazon where new items are added constantly resulting in a huge number of options for recommendation. The approach “leverages prior information about the actions to embed them in a continuous space upon which it can generalize. Additionally, approximate nearest-neighbor methods allow for logarithmic-time lookup complexity relative to the number of actions.” [21]

The second issue is the serialized sampling of trajectories. Due to the resources the simulation of scenario requires, parallelization is key. Thanks to the already implemented replay buffer, the sampling of trajectories with multiple copies of the actor can be offloaded to clusters with the results be fed into it. Then, the training can be done on demand resulting in a new version of the actor and critic.

2.4 Workgroup: Storage and Compute

2.4.1 Introduction

The Storage and Compute workgroup is concerned with the development of an RL architecture utilizing DT. The emphasize of this work group is on specifying the interplay between all the components, i.e., reinforcement learners, feature engineering module, and also the scenario generator in the DT.
In the following subsections, we define the prototypes for the components and the interfaces between them in the platform. In doing this, we address certain crucial soft requirements such the scalability and maintainability of the prototype. We also address in the following some possible extensions for the basic models such as queueing of the learning process, tracking and storing the RL experiments, and also management of data.

2.4.2 Architecture

The system follows a microservice architecture and consists of four principal components with the same separation of concerns as the workgroups that were described previously.

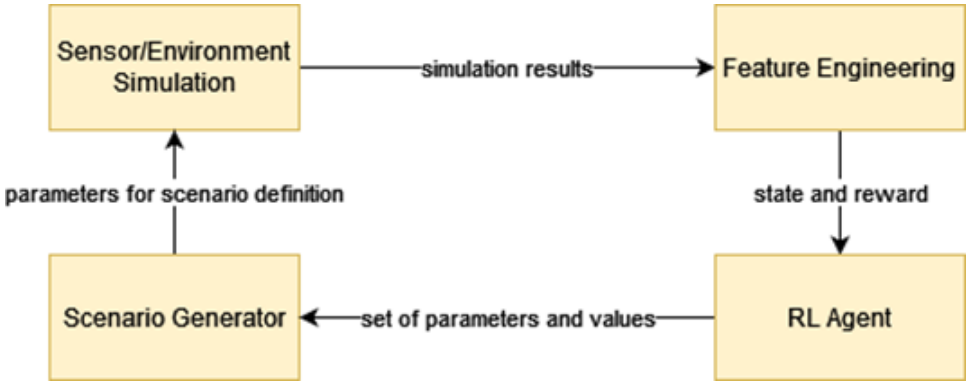


Figure 23 - UUV Use Case Process Diagram of the principal components during the training loop

Each component can consist of multiple sub-components where one component is an interface component that interacts with other units.
Apart from the interface component, the sub-components are not supposed to communicate with external units as indicated in Figure 24. All external communication is handled via the interface component.
It follows the idea of the Dependency Inversion Principle. In case an external unit changes, ideally only the interface component has to be changed in order to adapt to the new environment.
In addition, it allows development on the sub-components without affecting other units as long as the contract made on the interface is maintained.

The interface component provides at least one endpoint to fulfill its functionality. Additional endpoints may be provided, for example to inform of processing status, health, metrics, etc.

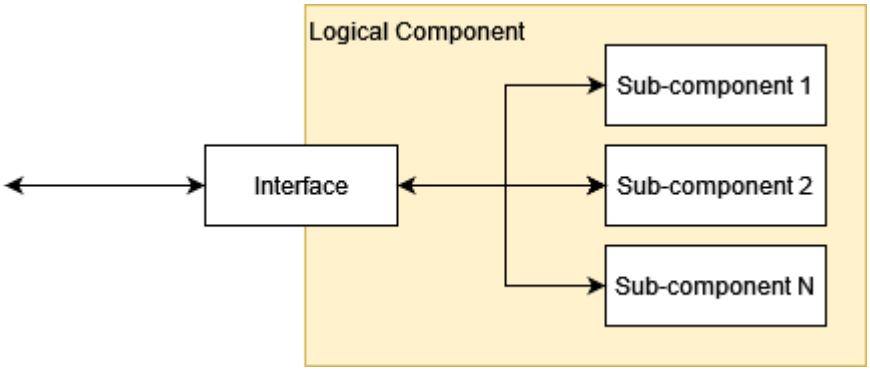


Figure 24 - Structure of a component with its interface and sub-components

The data exchange between the interfaces of the components is done via network. This ensures that the developed solution will be able to run in a distributed environment. In case units are running on the same physical host with access to the same hardware, the interfaces can provide an option for accepting and sending links to data on the local file system. In this case, only the path is transferred, and the actual data transfer happens in the interface-sub-component.

The exact mechanisms to exchange data can be chosen freely as long as the overall communication can take place with the chosen option.

Each sub-component / the interface component shall be realized as a containerized application. This makes sure that it can be deployed with low effort in different environments.

Containerization constitutes a state-of-the-art way for abstractions at the application layer and is based on Linux container technology. As codes, dependencies, and environments are packed together in containers, there is no need for substantial preparation for deploying them. In this hindsight, it is easy to distribute the RL architecture realized with this technology to the project partners, e.g., for testing purposes. Moreover, multiple containers can run on the same machine as isolated processes, so that the separation between the RL components can be implemented by means of this technology.

Another reason for using containers for realizing the RL architecture is that the possibility to additionally use the Kubernetes framework [24] for management purposes. Kubernetes constitutes a state-of-the-art technology for orchestration of containerized applications. It supports the automatic deployment, scaling, and management of containers which suits perfectly with ASIMOV's goal of creating a scalable and automated RL methodology with DT.

In the following, an overview of the components and their basic functionality is given:

RL-Agent:

Endpoint Start Training Loop:

Input: Training Parameters

```
{
"id": 232334,
```

```
"training-parameters":"file://path/to/<id>/training-parameters.json"
}
```

Output: Status of finished training

```
{
"id": 2323334,
"status": success/failure/aborted
}
```

Configuration:

- Model parameter
- Queue/Endpoint for Output
- Queue/Endpoint for Performing Actions

Endpoint Generating Action:

Input: State, Reward

```
{
"id": 232334,
"state-reward": "file://path/to/<id>/state-reward.json"
}
```

Output: New set of parameters and values (compare Figure 4)

```
{
"id": 232334,
"parameter": "file://path/to/<id>/parameter.json"
}
```

Configuration:

- Model parameters
- Queue/Endpoint for Output

Scenario Generator:

Input: JSON string or path to JSON file with basic parameters and values (see Figure 4)

```
{
"id": 232334,
"parameter": "file://path/to/<id>/parameter.json"
}
```

Output: Path to zip-file with Carla Project, modified OpenDRIVE file and action parameters

```
{
"id": 232334,
"carla-zip": "file://path/to/<id>/carla.zip",
"opendrive-file": "file://path/to/<id>/opendrive-file",
"parameter": "file://path/to/<id>/parameter.json"
}
```

Configuration:

- Basis OpenDRIVE File
- Basis Project Data
- Queue/Endpoint for Output

Additional notes:

- Basis Project data must not change during runtime
- Relays the JSON-file that it receives as input, uses the parameter it needs and leaves other parameters unchanged

Sensor/Environment Simulation:

Input: Project Data for Carla and modified OpenDRIVE file

```
{
"id": 232334,
"carla-zip": "file://path/to/<id>/carla.zip",
"opendrive-file": "file://path/to/<id>/opendrive-file",
"parameter": "file://path/to/<id>/parameter.json"
}
```

Output: Simulation results

```
{
"id": 232334,
"osi-file": "file://path/to/<id>/osi-file.osi"
}
```

Configuration:

- Simulation configuration
- OpenSCENARIO file: file://path/to/openscenario-file

- OpenDRIVE file: file://path//to/opendrive-file
- Queue/Endpoint for Output

Feature Engineering:

Input: Simulation results

```
{
"id": 232334,
"osi-file": "file://path/to/<id>/osi-file.osi"
}
```

Output: State and reward

```
{
"id": 232334,
"state-reward": "file://path/to/<id>/state-reward.json"
}
```

Configuration:

- Algorithm configuration
- Queue/Endpoint for Output

In addition to the four principal components there are additional components that are not required to provide the main functionality of the prototype but improve user and developer experience, ease of deployment and reproducibility:

UI:

Input: Parameters for the Training Run via UI
Output: Run Success / Failure
Configuration:

- Link to all units

Additional Notes:

- Must be able to stop/pause all components

Model Management and Tracking:
Exceeds a single functionality.
Multiple endpoints for tracking experiments, storing, and retrieving models.
Configuration: Data backend

2.4.3 Implementation of the prototype

In our prototype of the RL+DT architecture, we realize the sub-components as docker containers [22] having executable python files containing the designed algorithms. The applications within the containers are supported by the flask framework [23] so that they can mutually communicate via HTTP request.
The initial choice for an http REST interface based on flask [23] was made due to its simplicity and its low required effort for local testing. For the later development we plan to switch to an architecture based on AMQP (Advanced Message Queuing Protocol) as this simplifies a possible parallelization extension of the training and operation process.

The payload of the messages between the containers is currently empty as the parameters that would usually send as payload are fixed and therefore part of the current interfaces contract. For future versions JSON formatted configuration will be sent and expected that contains various information. For example, paths to input and output files, an identifier and status information.
The configuration of the docker containers is done with a docker-compose file which describes the setup and mounts common directories. Running the prototype is currently done on a single machine, even though the possibility of running them across multiple machines via docker swarm would be already given due to their communication via network.

As all components are running on the same physical host with access to the same hardware, the interfaces currently expect the data to be on the local file system. Later versions of the prototype will be able to send links to remote locations in the payload of their messages, so that the data transfer happens in the interface.

An additional component that is embedded in our RL architecture is a logging and tracking component for the machine learning process. With logging and tracking of the RL experiments, one has a better control of the learning process by being able to e.g.:

- observe the parameter used for the experiments,
- observe the model used in the previous experiment,
- find out the best hyperparameters in the overall experiment,
- and compare the result of the learning experiments over time.

This functionality opens for instance the possibilities of transferring learning results between different times and different learning agents. Moreover, by utilizing this component one can involve experts to additionally analyze the learning outcome of the experiments and reproduce the learning experiments if further analysis is needed.

The API that is used for this purpose is the API of ML-Flow [26] whose structure is depicted in the following figure:

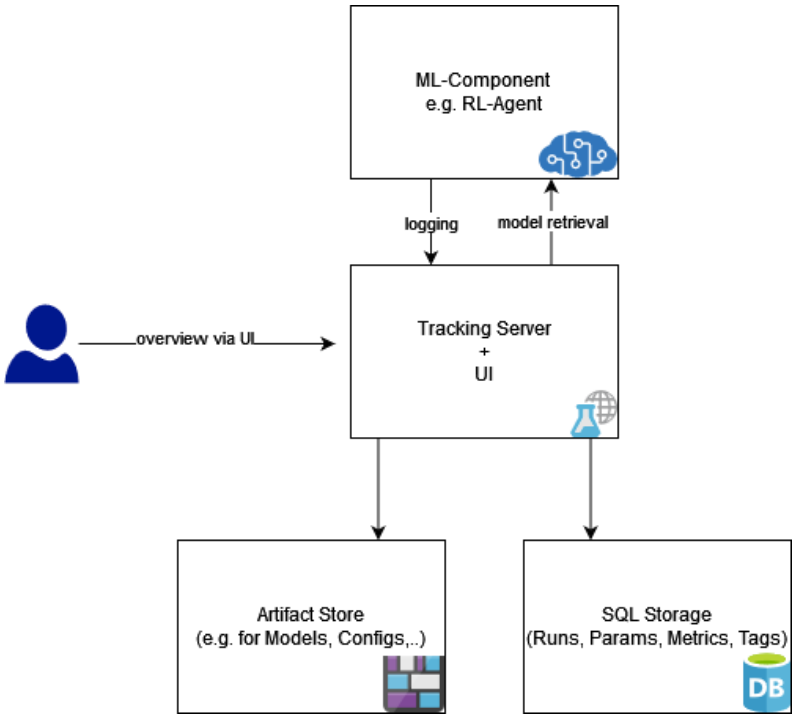


Figure 25 - MLflow structure

To enable all partners to run the prototype on their hardware and to ease development, there is a shared Gitlab from DLR to collaborate on the code basis. For the exchange of container images, especially for closed-source components, NorCom provides a private container registry accessible by all partners.

NorCom also hosts the model management and tracking solution, based on the ML-Flow API [24].

Currently the prototype is expected to run in learning mode which means it is the responsibility of the RL-Agent to run experiments as long as the condition for continuing is fulfilled. It was noted that for the operational phase, the RL-Agent will only provide actions based on the state and reward it was given. Running the simulation is then not the responsibility of the RL-Agent anymore.

A user interface (UI) is provided within the DaSense Analytics platform of NorCom. Via the UI the parameters for each component can be adjusted and selected. The UI will be used to start and stop experiments, to monitor the status and to deploy the solution on a cluster. A screenshot of it can be seen in Figure 26.

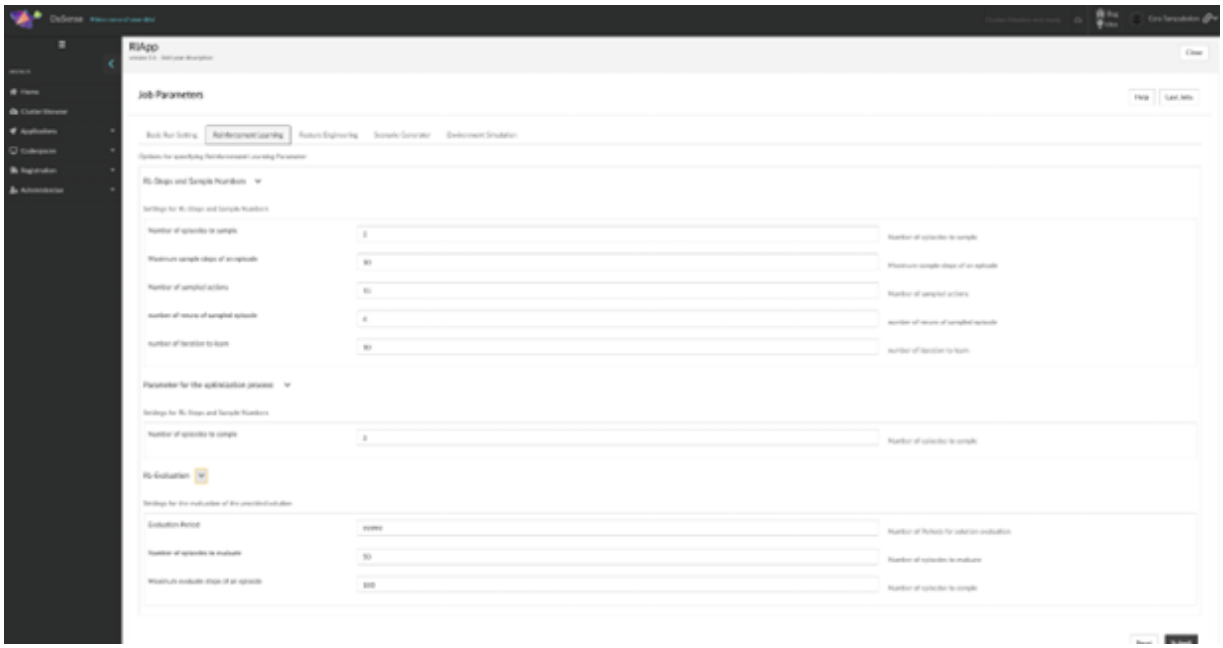


Figure 26 - Screenshot of the UI for the prototype within DaSense Analytics

2.4.4 Future improvements

The REST interfaces will be replaced by a queuing architecture. Communication via HTTP messages was helpful for setting up a simple and functional prototype but requires additional effort if asynchronous work has to be performed. As potential scalability is important, the interfaces of the components will be changed to use AMQP for communication. This allows easier scaling of components. The envisaged structure can be seen in Figure 27.

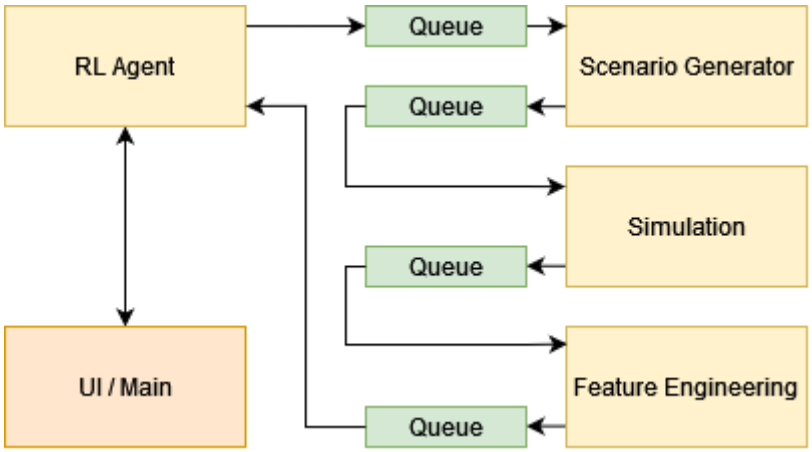


Figure 27 - Architecture with queues

An example where three instances of the scenario generator and three instances of the sensor / environment simulation are deployed can be seen in Figure 28. Each component will fetch the next job from the queue as soon as it finishes with its current job.

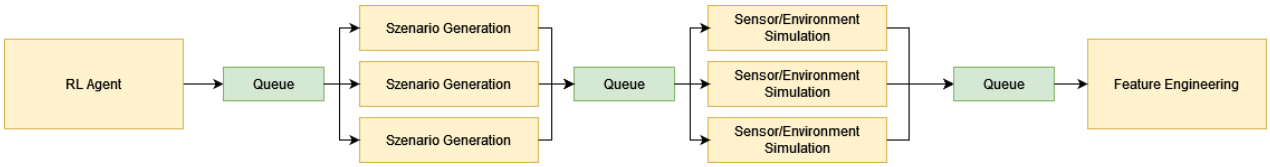


Figure 28 - Example where three Scenario Generator units and three Simulation units are used in parallel. As soon as a unit is finished with its current task, it fetches a new task from the queue

The main contrast between this extension and our first prototype is that the tasks for each component in the RL architecture are stored in a separate queue buffer. The advantage of this approach is that each component does not need to wait for the request from the main component in order to execute the task. Merely, the components execute the tasks stacked in their buffer. This may open the possibilities to parallelize the RL components workflow, which in turn increases the performance of the ASIMOV's solution.

As containers per se do not yet provide mechanisms for scaling the applications, porting the container structure to Kubernetes is envisaged.

3. Physical twin setup

To test and demonstrate the toolchain for the optimization of physical system, such a system needs to be developed as well as a DT. As already stated, the goal of the UC's optimization is the alternation of the virtual environment given properties of a vehicle and its testbed. This can be seen in Figure 29 where on the right the loop of RLA, Scenario Generator, Environment Simulator and Feature Engineering is shown and on the left vehicle in the loop is given. In the following, the testbed is explained, the DT development process documented and the twinning process and integration into the toolchain described.

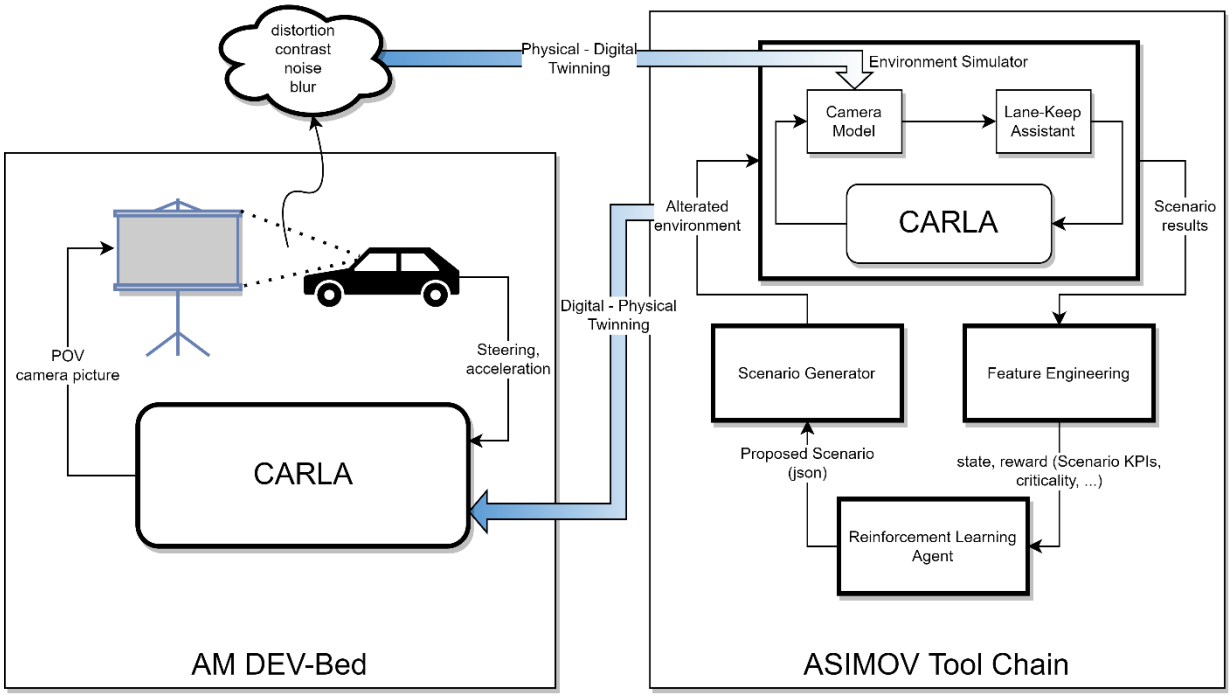


Figure 29 - Overview of the interaction between the ASIMOV Tool Chain and the physical system "AM DEV-Bed". The name of the latter is the sub-project name of the Testbed within DLR

3.1 DEV-Bed

AM DEV-Bed stands for "Automotive Model Demonstration Evaluation and Verification Bed" and is a multi-purpose demonstrator developed by DLR. It is a vehicle-in-the-loop setup where a miniature vehicle based on the f1tenth-challenge's RC-Car (https://f1tenth.org/) is adapted and set into a virtual

environment. The link between the virtual representation of the car within the virtual environment and the actual vehicle works as following:

- Physical-to-Virtual interaction: The vehicle controls itself and thus also the virtual counterpart within CARLA using steering and acceleration commands. As the communication of the system is ROS 2-based, see figure [DeV-bed architecture], the testbed receives the published outputs of the driving function and feeds them back via a ROS-CARLA bridge to the virtual vehicle.
- Virtual-to-physical interaction: The virtual vehicle has a POV camera, and its output is projected onto a monitor which is positioned in front of the physical vehicle, thus mimicking the RGB-camera's stimulation as if the vehicle would be driving in the environment itself. The virtual camera within CARLA has the possibility of adjustments (see https://carla.readthedocs.io/en/latest/ref_sensors/#rgb-camera) such as f-stop, fov, gamma, ISO, and positioning in 6 degrees of freedom with respect to the origin of the car. However, these settings are left untouched.

The architecture of the DEV-Bed is intended to be modular by the usage of ROS 2 as the communication form between components. The system consists of a Simulation-PC which handles the environment simulation in the form of CARLA and a CARLA-ROS-bridge. It is connected via LAN with the vehicle. The vehicle is based on a Nvidia Jetson which has a simple webcam connected to it via USB and runs the driving function. The prototypical, unfinished setup can be seen in Figure 30.

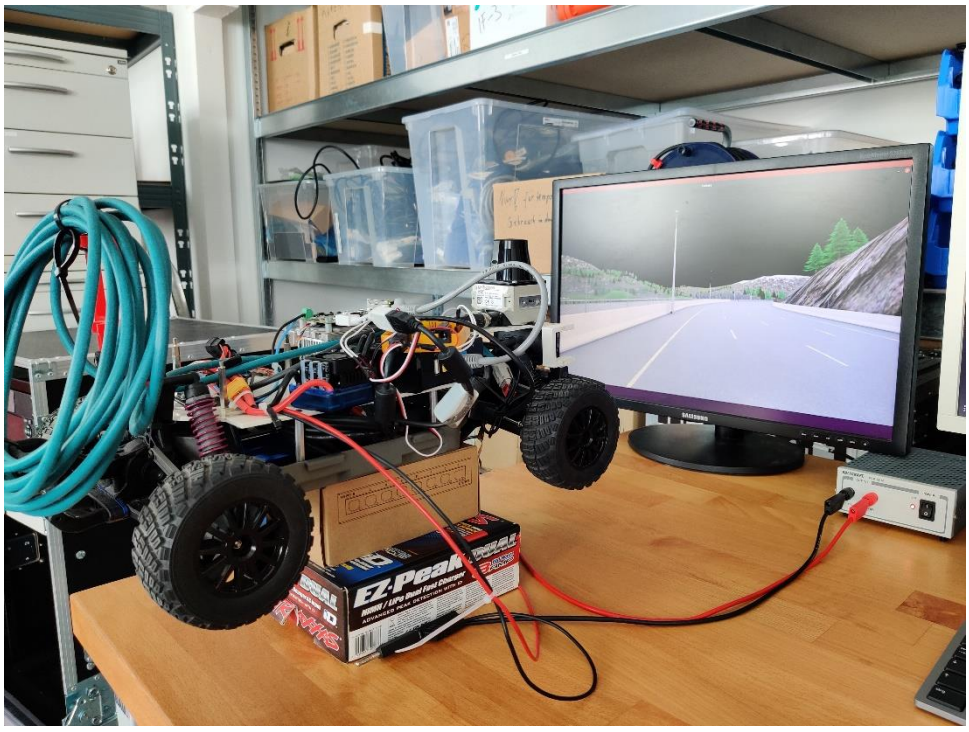


Figure 30 - Picture of the first, unfinished prototypical setup

3.2 DT development

We acknowledge that the DEV-bed's sensor stimulation and vehicle dynamics do not accurately replicate the real RC-car on a street or any real environment. However, we consider it as the “ground-truth”. The information flow as shown on the right side of Figure 31 shows that the DEV-bed's Digital Shadow reuses the Virtual Environment and driving function but has two instances where information is not processed virtually but actual real-world effects are induced. On one hand, this happens during the screen-camera interaction as well as the actuation measurement. For the sake of simplicity, a DT of the previous is prototypically developed to integrate it later into the toolchain as shown in Figure 31. The DT would be a camera model which mimics the conversion of the RGB-picture given by CARLA from the sensor of the virtual vehicle to the RGB picture the camera from the real RC-car. Thus, the Digital Shadow consists of the Virtual Environment, the camera model which can automatically be updated as well as the driving function.

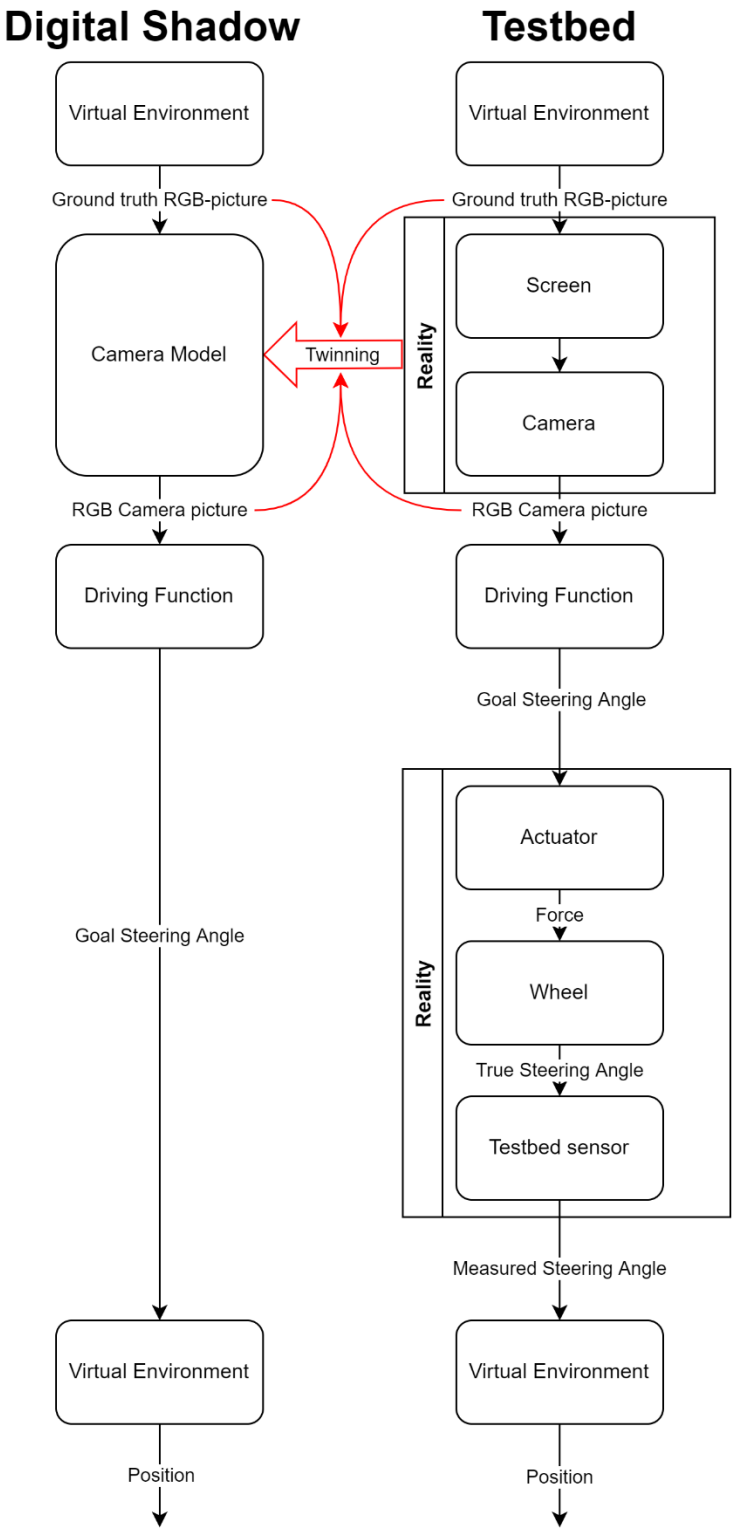


Figure 31 - Information flow through the DEV-bed highlighting the two instances where components are not virtual and which part shall be twinned

The camera model was developed in a “grey-box” approach. While there are physical phenomena that can be observed and are identified in the process, they are not tried to be explained and integrated as first-principle models, but algorithms are put into place which replicate the resulting behavior.

To identify static phenomena, a test picture (see Figure 32) is displayed on the monitor and a frame of the camera feed is taken for comparison (see Figure 33). In the following the identified phenomena and their quantization is explained.

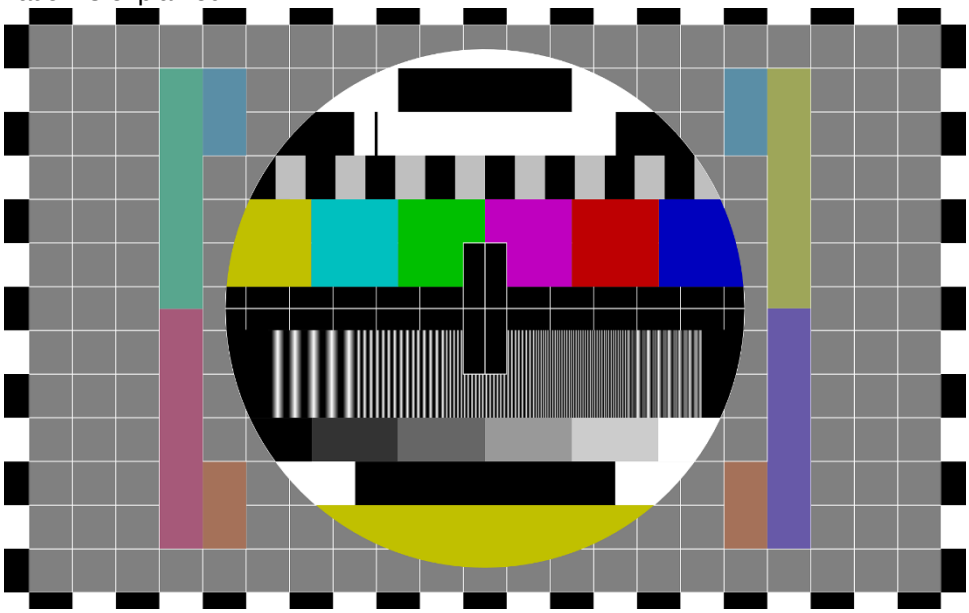


Figure 32 - Original test picture

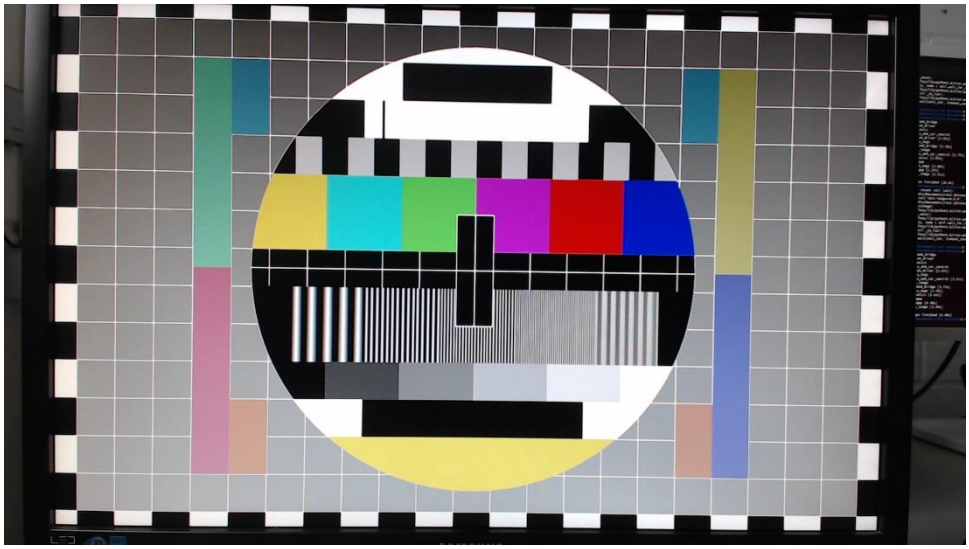


Figure 33 - Sample frame taken from the camera feed

Perspective: Due to the imperfect alignment of the camera with the monitor, a shift, scaling and warping effect can be seen. The monitor or camera can be both off-axis and at an angle to the camera or monitor, respectively. In addition, the monitor only occupies part of the camera's frame (scaling). These phenomena are addressed by first identifying (or closest pixel to) the four corners of the monitor by searching for the closest pixel to the corner of the test frame from the camera which is above a certain brightness threshold. Then a perspective warp function can be applied on the test picture (see warpPerspective() in OpenCV: Geometric Image Transformations). This method is able to replicate the found perspective phenomena. The limitations are that it is sensitive to finding the right pixel representing the corners of the screen which needs to be visible. It also assumes that they are linear e.g., no barrel or pincushion distortion, which hasn't been observed in the test frame.

Brightness: When comparing the grey grid on the test picture in Figure 32 with the one from Figure 33 an obvious shift in brightness of from top to bottom (increasing brightness) can be seen. This is due to the polarization of the monitor where with increasing viewing angle higher brightness distortion is induced. In addition, the shift can be seen from left to right due to the same reason but not as strong. To adjust the original test picture in such a way that the brightness mimics the one from the test frame, the test picture is sliced vertically along the grid of grey boxes. It is estimated that the brightness within one box does not change significantly. Thus, for each slice, the brightness of the topmost and undermost box is measured. From the test picture (Figure 32) the true brightness is known. The necessary brightness adjustment in each slice for each pixel row of the test picture is then linearly interpolated between the aforementioned topmost und undermost box.

Blur: Due to the imperfectness of the monitor as well as the camera, blur is induced which needs to be accounted for. However, the direct measurement of blur is not possible. However, there are methods of measurement for sharpness which is directly reverse related to blur like the calculation of the variance of the picture's Laplacian (as described https://pyimagesearch.com/2015/09/07/blur-detection-with-opencv/). In short, a high value of the Laplacian is given where sharp edges between pixel brightness is detected which can be understood as high sharpness and low blur. The overall variance gives a reliable metric. However, for reproducible results, the underlying image needs to be similar and for the no link to the variance needed for Gaussian blur is established. For this, a lookup table is created. The original test picture is iteratively blurred with increasing levels of Gaussian blur represented by sigma. The sharpness according to the above-mentioned process is calculated and the sigma-sharpness data points are stored. To reverse the calculation, the test frame from the camera is "reverse-warped" (the test frame looks now like the test picture) according to the already mentioned perspective change and sharpness is measured. Using the sigma-sharpness data as a lookup function, the respective sigma can be calculated and applied to the test picture.

Delay: In addition to the static phenomena, dynamic ones need to be addressed. During test recordings of the camera footage, motion blur could not be observed. However, the system causes a delay which seems to influence the driving function's ability. In short, control inputs for the virtual vehicle coming from the driving function processed on the RC-car are based on old visual information from the environment. The total delay experienced is a sum of the delay of monitor, webcam, driving function and communication via the network to transmit the control command. It is unclear how much each component contributes to the overall delay as the RC-car's and simulation workstation's clocks are not synchronized and therefore measurements across machines cannot be performed. Thus, the entire delay from change in the environment's state to an alternated control command will be included in the model. As the frequency of processing at the RC-car remains at 30 fps, the delay needs to be as a queue where the vehicle acts on old information and needs to be modelled accordingly. The measurement of the delay is straight forward: the RC-car is positioned in line to the lanes inside the simulation without any forward movement. The virtual vehicle is then instantly rotated by 5°. The time is measured between this event and a step change in steering command of the vehicle. This is repeated five times and averaged to account for fluctuations.

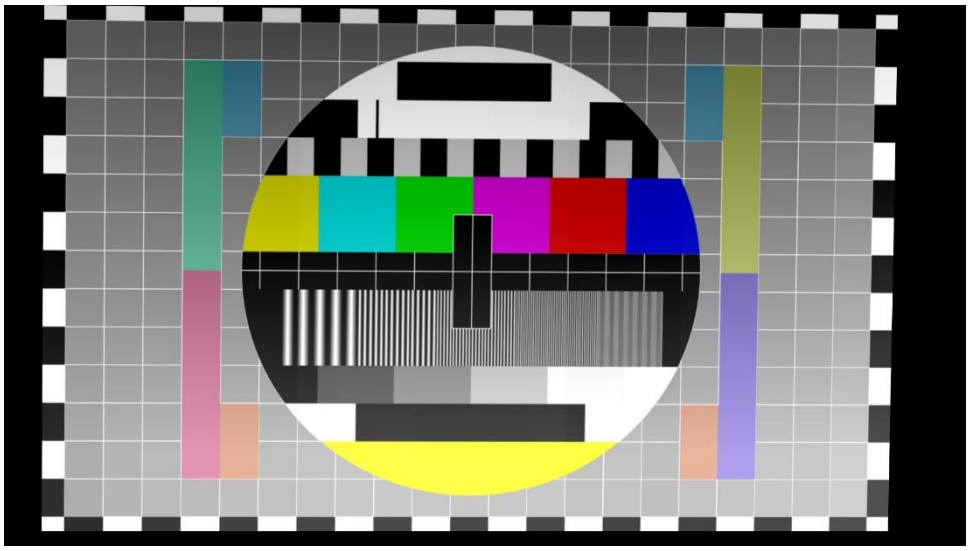


Figure 34 - Adjusted test picture

When blur, brightness and perspective adjustments are applied to the test picture, it looks like in Figure 34. It can be seen that the image looks subjectively similar to the Figure 33, thus it works as designed. However, slight differences can still be identified at first sight: colors do not match and effects from outside the screen are not accounted for. Validation of this model is pending and will be described in D2.5.1.

3.3 Toolchain integration

For the physical-to-digital twinning, parameters of the above-detailed model are stored in a JSON file. The model is then integrated into the Co-Simulation between the CARLA client script which receives the environment information (such as camera pictures) and the driving function. The parameter data is loaded in to configure the model which serves as a “filter” on the original CARLA footage.

Once the optimization is done, the Scenario and optimized world are loaded into the Dev-bed. This can be seen as the Digital-Physical Twinning see Figure 31.

4. Validation of Digital Twin Perception

For a sufficient digital twin of a UUV, the autonomous driving function must be provided with similar information as the physical system. Environmental information is perceived via sensors, relevant features are extracted via perception software, which is then written into object lists and handed over to the driving function. The interplay of environment, sensor and perception software within the digital twin is essential, to provide an object list that resembles the same information as its physical system counterpart in the same scenario.
The defining component in this complex is the sensor. How detailed the environment needs to be simulated (selection of parameters and fidelity) highly depends on what the sensor is capable to perceive. The perception software should be the same in the digital and the physical twin. But noise or in the data stream coming from the sensor need to be regarded.
How detailed each component needs to be reproduced to fulfil the designated task without wasting resources or overpowering the simulation framework is a crucial question if one wants to realize a useful UUV digital twin with autonomous driving function. In this chapter an experiment is described to answer the question for one of the sensors that are used for autonomous driving vehicles.

4.1 LiDAR-Sensors, basic properties and requirements for autonomous driving functions

One integral sensor for autonomous driving are LiDAR sensors (Light Detection And Range). Here laser beams of a specific wavelength are emitted by the LiDAR head. These are reflected by objects in the environment and finally propagate to the detector. Because the speed of light is constant for every given

medium and also the time of the emission is known, the system can calculate for every detected burst of light where it got reflected[1].
Therefore, each frame of information the sensor issues is a three-dimensional cloud of reflective spots of different intensities, called point cloud, see Figure 35. The result is not only influenced by the position of objects that caused the reflections but also by their reflectivity in the wavelength range of the laser, the local air density, weather (especially fog, but also rain), vibrations perturbing the measurement, e.g., smaller and larger road bumps that rock the vehicle together with the vehicle suspension, possible multi-reflections. A precise representation of the physical system would have to regard all these factors.

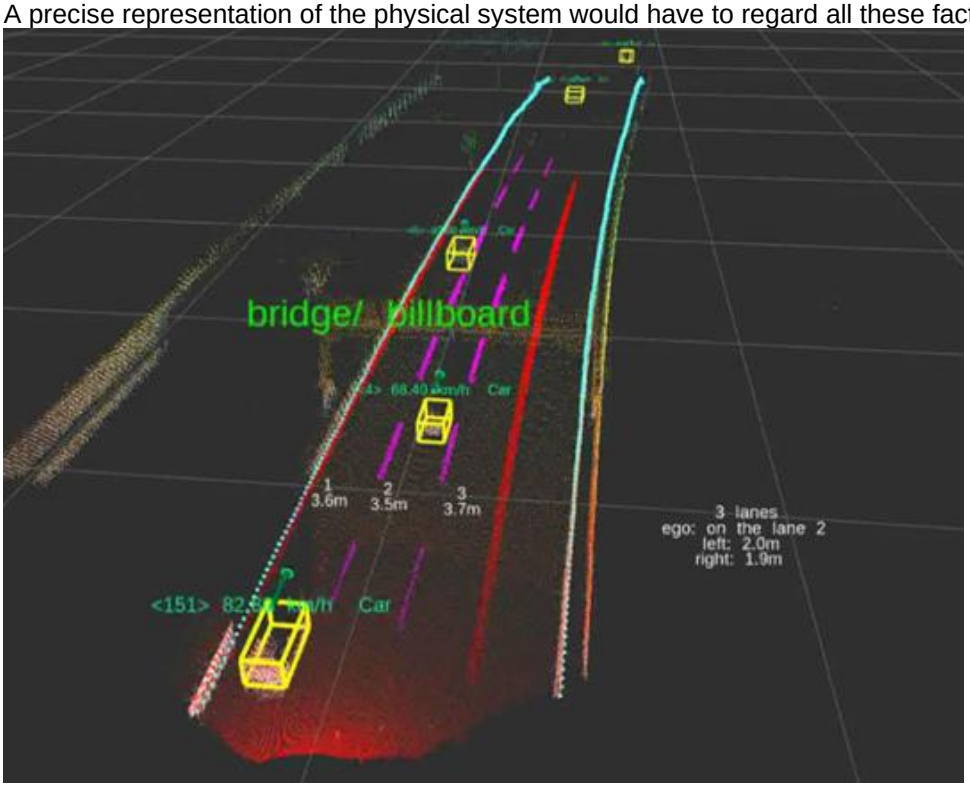


Figure 35 - LiDAR 3D Point cloud of an Autobahn scene. The scene was analyzed by perception software. Identified objects have been marked with bounding boxes, which have been annotated with additional information (class, e.g. car, and speed)

However common simulation tools, such as Carla, the program used in the described tool chain, do not regard all of these. E.g. Carla uses ray-casting to determine where the laser gets reflected, a technique that is not suited to consider multiple reflections (and therefore also the influence of fog on the propagation path of the laser). Implementing other factors, such as the exact roughness of the road to rock car and sensor as in the physical system, would consume a great amount of resources (collecting the information, building the model, run the model). Resulting in the question, if such a level of detail can be achieved economically. But also, if it is needed. Except for corner cases, most of the mentioned perturbations would only cause a minimal variation in the data stream. For an example see Figure 36.
Here a LiDAR image of a target plate (usually used for calibration) generated in Carla (left) and overlayed with data from the measurement of the physical system in a lab with a fixed LiDAR position is shown. The basic properties of the physical LiDAR have been reproduced, additionally 0.05 per cent of noise where added. Still the resulting points reflected from the target appear very ordered and with a continuous change in intensity from top to bottom (left). The points of the physical measurement do not show such a distinctive intensity change and especially in the top part of the target the pattern is significantly disturbed (right). However, the target as a whole is well recognizable in both instances. Also, the edge measurements of the target taken in the digital system fit the edges of the target of the physical system well. Compared to the absolute dimensions the deviations are large on point level and small on object level.

[1] The distance is calculated based on the time of flight, the detection of the angular orientation of the reflective spot depends on the LiDAR type.

Because AI-based object perception is based on point groups (shape etc), therefore on object level, and works despite different noise levels on a variation of typical roads and (mild) weather conditions, the deviations caused by a simplified framework as Carla uses it, should not alter the result, i.e. the object still gets recognized, classed and positioned as in the physical system.

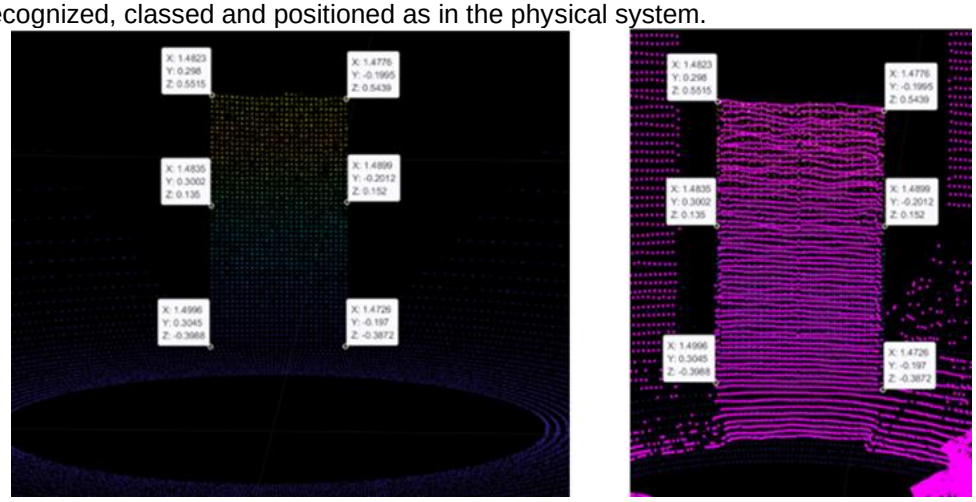


Figure 36 - LiDAR Measurement of a Target plate in Carla (left) and overlayed with the measurement in the physical system

4.2 Developing a sufficient cyber-physical testbed for LiDAR validation

To test this hypothesis, LiangDao designed a hunter/target experiment with a physical and digital counterpart. In the physical system, see Figure 37, the target is represented by an autonomous shuttle. Its course can be precisely programmed. Therefore, it can be easily reproduced in the digital system. The hunter is represented by a mobile roadside unit (RSU) consisting of two LiDAR and the hard- and software for object detection.[2] While the shuttle propagates on its course, the RSU records its environment. The raw data is analysed by the object detection, the shuttle gets identified, positioned and tracked. The object data is recorded and saved in an object list. For a validation of the physical system itself, the shuttle is also equipped with an RTK positioning system. Due to the mobility function of the RSU and the shuttle, both can be placed in a suitable environment with optimal relative positions.

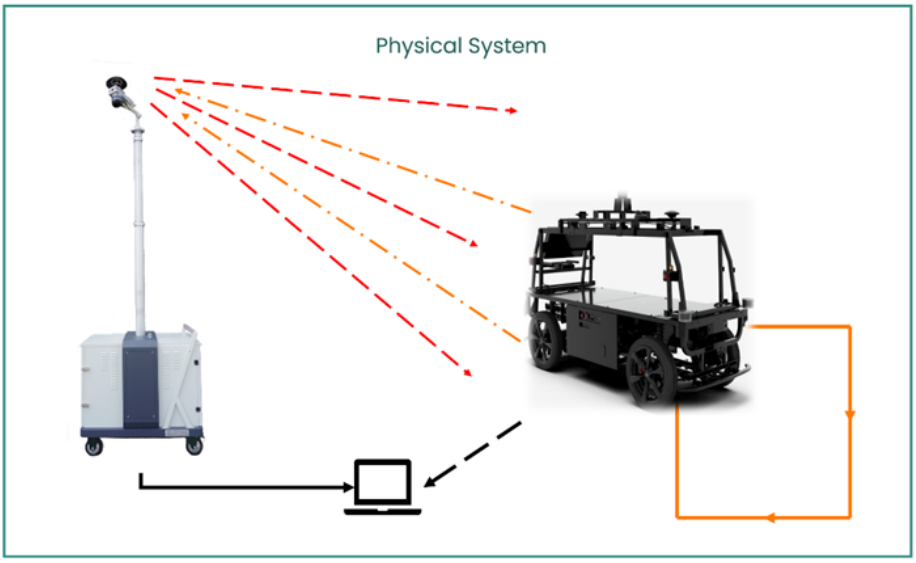


Figure 37 - Scheme of physical system part of the LiDAR validation experiment

[2] The object perception of the RSU is based on the autonomous driving function perception software. The results can be translated. However, using the mobile RSU allows a greater freedom of positioning.

The experiment is then rebuilt and repeated within Carla. The basic properties of the RSU, especially the used LiDAR, the shuttle and the environment are modelled to match the physical counterparts. However, weather properties, road details, multiple reflections are not considered. The general steps of the experiment are depicted in Figure 38. In both systems the shuttle is identified, positioned and tracked. The digital twin of the LiDAR is positively validated if the position of both shuttles (physical and digital) is matched within an error margin given by the intrinsic measurement error of the physical system.

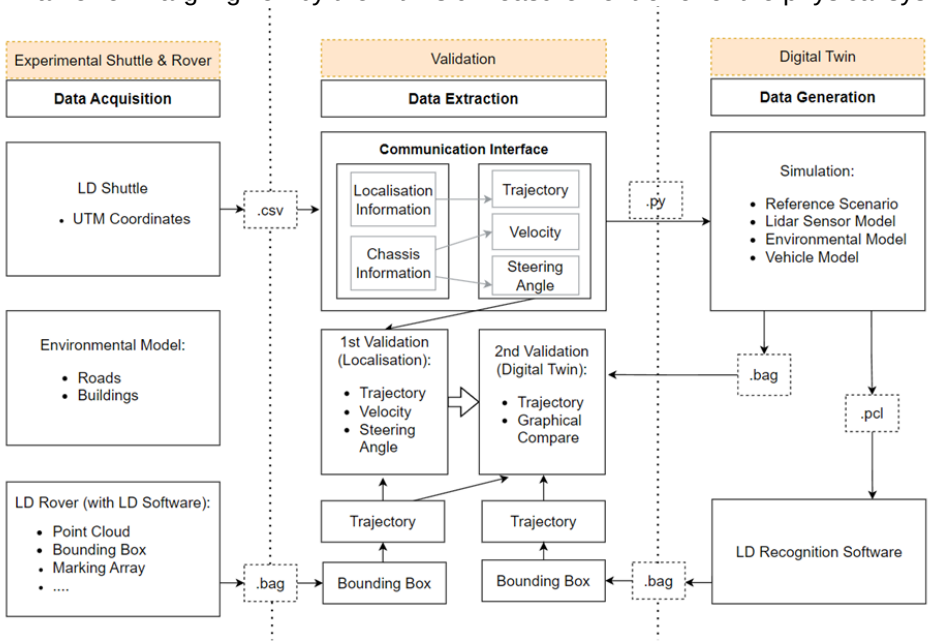


Figure 38 - Flow diagram of the LiDAR validation experiment

5. Summary and future steps

This report described the UUV UC development. It showcases the four main groups in which the task of creating the ASIMOV solution for optimizing scenario-based testing is divided: Storage and compute, Reinforcement learning, Environment Simulation, and Feature engineering. Their status of development is described in detail. It shows that all components are currently at an implementation stage with the main concepts and even details already developed.

On a top level, the next step is finalizing the implementation work and thorough testing of the individual components. In continuation, the next primary milestone is to test the implemented approach on the scale model demonstrator to gain insightful knowledge of the applicability of this testing approach. This will highlight potential issues and improvements that might arise which will be handled in the continuous iterations of the demonstrator.

6. Terms, Abbreviations and Definitions

Table 14 - Terms, Abbreviations and Definitions

ABBREVIATION	EXPLANATION
ACC	Adaptive Cruise Control
AD	Autonomous Driving
AEB	Emergency Braking
DT	Digital Twin
ES	Environment Simulation
FE	Feature Engineering
HTTP	HyperText Transfer Protocol
KL	Kullback-Leibler Divergence
LiDAR	Light Detection And Range
LKA	Lane Keep Assist
MDP	Markov Decision Process
ML	Machine Learning
MPO	Maximum a Posteriori Policy Optimization
NN	Neural Network
PT	Physical Twin
RL	Reinforcement Learning
RLA	Reinforcement Learning Agent
RSU	Roadside Unit
SC	Storage And Compute
SQL	Structured Query Language
TTB	Time To Brake
TTK	Time To Kickdown
TTM	Minimal Time To Maneuver
TTS	Time To Steer
UC	Use Case
UUV	Unmanned Utility Vehicle
VV&A	Verification, Validation and Accreditation

7. Bibliography

[1] “Continental,” [Online]. Available: https://www.continental.com/de/presse/pressemitteilungen/2019-12-12-hmi-cube-204622. [Accessed 22 11 2022].

[2] T. Brade and C. N. Birte Kramer, “Paradigms in Scenario-Based Testing for Automated Driving,” in *International Symposium on Electrical, Electronics and Information Engineering*, NewYork, 2021, pp. 108-114.

[3] “ASAM OpenDRIVE,” [Online]. Available: https://www.asam.net/standards/detail/opendrive/. [Accessed 25 11 2021].

[4] “ASAM OpenSCENARIO,” [Online]. Available: https://www.asam.net/standards/detail/openscenario/. [Accessed 25 11 2021].

[5] R. S. Sutton and B. Andrew, Reinforcement learning. An introduction, Cambridge, Massachusetts, London: The MIT Press (Adaptive computation and machine learning, 2018.

[6] R. Bellman, “A Markovian decision process,” in *Journal of mathematics and mechanics*, 1957, pp. 679-684.

[7] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare and J. Pineau, An introduction to deep reinforcement learning, arXiv preprint:1811.12560, 2018.

[8] R. G. A and N. Mahesan, On-line Q-learning using connectionist systems, Citeseer (37), 1994.

[9] M. P. Deisenroth, G. Neumann, J. Peters and others, “A survey on policy search for robotics,” in *Foundations and trends in Robotics 2*, 2013, pp. 388-403.

[10] esmini, “esmini - Github,” [Online]. Available: https://github.com/esmini/esmini. [Accessed 23 11 2022].

[11] “Unreal Engine,” [Online]. Available: https://www.unrealengine.com/en-US/. [Accessed 25 11 2021].

[12] “CARLA,” [Online]. Available: https://carla.org/. [Accessed 25 11 2021].

[13] L. N. C. K. T. e. a. Westhofen, “Criticality Metrics for Automated Driving: A Review and Suitability Analysis of the State of the Art,” *Arch Computat Methods Eng,* 2022.

[14] J. F. F. B. J. e. a. Degrave, “Magnetic control of tokamak plasmas through deep reinforcement learning,” *Nature,* vol. 602, pp. 414-419, 2022.

[15] A. Abdolmaleki, J. T. Springenberg, Y. Tassa, R. Munos, N. Heess and M. Riedmiller, Maximum a Posteriori Policy Optimisation, https://arxiv.org/pdf/1806.06920, 2018.

[16] A. Abdolmaleki, J. T. Springenberg, J. Degrave, S. Bohez, Y. Tassa and D. e. a. Belov, Relative Entropy Regularized Policy Iteration, http://arxiv.org/pdf/1812.02256v1., 2018.

[17] M. Neunert, A. Abdolmaleki, M. Wulfmeier, T. Lampe, T. Springenberger and R. e. a. Hafner, “Continuous-Discrete Reinforcement Learning for Hybrid Control in Robotics,” *Leslie Pack Kaelbling, Danica Kragic, Komei Suguira (Eds.): 3nd Conference on Robot Learning: PMLR (Proceedings of Machine Learning Research, 100),* pp. 735-751, 2020.

[18] N. P. Farazi, T. Ahamed, L. Barua and B. Zou, Deep Reinforcement Learning and Transportation Research: A Comprehensive Review, University of Illinois at Chicago, 2020.

[19] J. Duchi, Derivations of Linear Algebra and Optimization.

[20] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang and W. Zaremba, OpenAI Gym, 2016.

[21] G. Dulac-Arnold, R. Evans, H. van Hasselt, P. Sunehag, T. Lillicrap and J. e. a. Hunt, Deep Reinforcement Learning in Large Discrete Action Spaces, http://arxiv.org/pdf/1512.07679v2, 2015.

[22] Docker, “Docker Website,” [Online]. Available: https://docs.docker.com/get-started/#what-is-a-container. [Accessed 23 11 2022].

[23] Flask, “Flask Website,” [Online]. Available: https://palletsprojects.com/p/flask/. [Accessed 23 11 2022].

[24] MLflow, “MLflow Website,” [Online]. Available: https://mlflow.org/. [Accessed 23 11 2022].

[25] ASIMOV-consortium, *ASIMOV - Full Project Proposal,* 2020.

[26] Kubernetes, “Kubernetes Website,” [Online]. Available: https://kubernetes.io/. [Accessed 23 11 2022].

[27] RabbitMQ, “RabbitMQ Website,” [Online]. Available: https://www.rabbitmq.com/. [Accessed 23 11 2022].

[28] “MLflow,” 2023. [Online]. Available: https://mlflow.org/docs/1.28.0/rest-api.html. [Accessed 4 12 2023].