Methods and Tools for Training AI with Digital Twin

[WP2; T2.2; Deliverable: D2.2 Version 2.3]

Non-Confidential

AI training using Simulated Instruments for Machine Optimization and Verification

Version	Status	Date	Page
Version 2.3	Public	2024.05.31	1/52

Document Information

Project	ASIMOV
Grant Agreement No.	20216 ASIMOV - ITEA
Deliverable No.	D2.2
Deliverable No. in WP	WP2; T2.2
Deliverable Title	Methods and Tools for Training AI with Digital Twin
Dissemination Level	Public
Document Version	Version 2.3
Date	2024.05.31
Contact	Roy van Zuijlen
Organization	Eindhoven University of Technology
E-Mail	r.a.c.zuijlen@tue.nl

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
Jan Willem Bikker	CQM	janwillem.bikker@cqm.nl
Michael Wild	DLR	michael.wild@dlr.de
Narges Javaheri	TFS	narges.javaheri@thermofisher.com
Maurits Diephuis	TFS	maurits.diephuis@thermofisher.com
Christos Kitsanelis	TNO	christos.kitsanelis@tno.nl
Sebastian Moritz	TrianGraphics	sebastian.moritz@triangraphics.de
Maurice Heemels	TU/e	m.heemels@tue.nl
Duarte Antunes	TU/e	d.antunes@tue.nl
Jilles van Hulst	TU/e	j.s.v.hulst@tue.nl
Roy van Zuijlen	TU/e	r.a.c.zuijlen@tue.nl

Formal Reviewers

Version	Date	Reviewer
1.0	2022.10.04	Ezra Tampubolon (Norcom), Pieter Goosen (TNO)
2.2	2024.05.15	Andreas Eich (Liangdao), Philipp Borchers (DLR)

Change History

Version	Date	Reason for Change
1.0	2022.09.16	Ready for formal review
1.1	2022.10.20	Updated after formal review. Ready for publication.
2.0	2022.11.03	Working document for final version
2.1	2024.01.17	Deliverable combined with separate documents.
2.2	2024.04.17	Ready for formal review
2.3	2024.05.31	Updated after formal review. Ready for publication.

Abstract

One of the main premises of the ASIMOV project is that optimal settings and behavioural policies of cyber-physical systems can be found through Artificial intelligence (AI) techniques that leverage digital twins (DTs). DTs generate artificial data to feed the training of AI data-hungry algorithms, dramatically reducing the needed data from system trials. In this setting, it is crucial to define the interface between the DT/real-system and the AI agent or algorithm. This interface establishes how the DT generates training data for the AI algorithms so that the behaviour and optimization possibilities of the system can be understood by the AI algorithms. The present document defines this interface with generality and shows that it is broad enough to capture the two main use cases, namely the Electron microscope and the Unmanned Utility Vehicle. The proposed interface relies on three main components: pre-processing, post-processing, and variations. Pre-processing matches high-level actions/decisions to be taken by the AI agent to proper low-level inputs of the system. Post-processing translates real observations generated by the system and/or simulated observations generated by the DT into suitable information for the AI agent to respond to, e.g., the State and the Reward in Reinforcement Learning (RL). Variations pertain to informative data generation, namely by considering sufficiently rich variations or scenarios to provide a suitable training set. Related literature associated with these three main components of the interface is surveyed. For each of the two main use cases of the project, the Electron Microscope, and the Unmanned Utility Vehicle, the three main components are instantiated, and use-case-specific training features are discussed. The last chapter of the document is dedicated to the most important lessons learned during the project regarding pre- and postprocessing, and variations.

Contents

Terms, Abbreviations and Definitions

Table 1 - Terms, Abbreviations and Definitions.

Abbreviation	**Meaning**
AI	Artificial Intelligence
ALE	Arcade Learning Environment
CPS	Cyber-physical System
DQN	Deep Q-Learning
DT	Digital Twin
EM	Electron Microscope
FL	Federated Learning
GAN	General Adversarial Network
GPS	Global Positioning System
HMM	Hidden Markov Model
ICA	Independent Component Analysis
KPI	Key Performance Indicator
LDA	Linear Discriminant Analysis
MDP	Markov Decision Process
MSE	Mean Squared Error
NN	Neural Network
ODD	Operational Design Domain
PBRS	Potential-Based Reward Shaping
PCA	Principal Component Analysis
PID	Proportional Integral Derivative
POMDP	Partially Observable Markov Decision Process
PT	Physical Twin
RL	Reinforcement Learning
RLA	Reinforcement Learning Agent
RM	Reward Machine
RS	Real System
SEM	Scanning Electron Microscope
STEM	Scanning Transmission Electron Microscope
TEM	Transmission Electron Microscope
TG	TrianGraphics
UC	Use Case
UUV	Unmanned Utility Vehicle
VAE	Variational Autoencoder

1 Introduction

Cyber-physical systems (CPSs) are increasing in complexity and number, which emphasises the need for more advanced methods to control how they behave. The goal of the ASIMOV-project is to optimise the process of determining optimal settings or behavioural policies for these systems. Due to the complexity of such systems, Artificial Intelligence (AI), especially reinforcement learning (RL), is considered a promising direction for controlling CPSs. In RL, an AI agent learns to perform desired behaviour by interacting with an environment (Sutton & Barto, 2018). By exploring the complete domain of the environment, namely from all-encompassing experiences, an optimal policy can be discovered.

A challenge within RL is the need for a huge amount of experience to eventually discover the optimal policy. It is often impractical to gather the necessary amount of experience data from the actual system/environment. Consider for example the two CPSs under consideration in the ASIMOV-project: the electron microscope (EM) and the unmanned utility vehicle (UUV). These systems have a highly complex structure with many inputs. Using these systems for collecting the needed experience is unfeasible. Furthermore, training time is very costly on these real systems (RS's). To overcome these two drawbacks, digital twins (DTs) are used in the ASIMOV-project. In fact, synthetic data generated by the DT can replace or complement the actual environment data. Moreover, DTs allow faster than real-time training, in a cost-efficient digital environment.

A crucial aspect in a DT-supported training environment, is the interface between the AI agent and the DT. This interface will be created in this task. In particular, the central question addressed in T2.2 is:

How can a DT create corresponding training data for an AI, so that behavior and optimization characteristics of the modelled product can be understood by the AI?

The structure of the interface between DT/RS and AI agent can be seen in Figure 1. It should be applicable to the RS as well and consists of three blocks: pre-processing, post-processing, and variations. Pre-processing matches (high-level) actions/decisions to be taken by the AI agent to proper (low-level) inputs of the system. Post-processing translates real observations generated by the RS and/or simulated observations generated by the DT into suitable information for the AI agent to respond to, e.g., the State and the Reward in RL. The block "variations" pertains to informative data generation. It considers sufficiently rich variations or scenarios to provide a good training set. In particular, the DT-based training setting in ASIMOV allows to introduce more "exceptional/rare" cases in the variation of the training set, to properly prepare the AI agent for these scenarios. This contrasts with the generation of data based on the RS, in which such rare cases are often not (sufficiently) encountered. Hence, the trained AI agent is typically not trained for choosing proper actions in these cases. This is a potentially strong benefit of DT-based training (certainly for safety-critical applications or when calibrating fragile expensive equipment). In this task, these three blocks are elaborated in detail and eventually mapped to the use-cases.

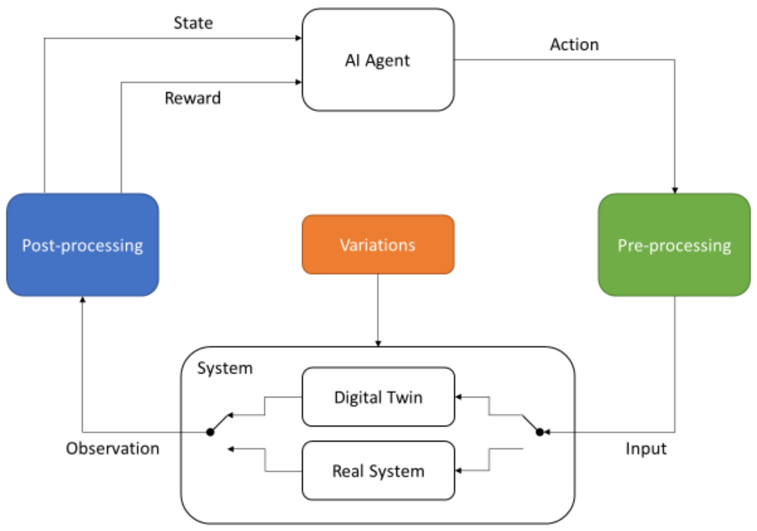


Figure 1 Structure of Task 2.2 contents.

The remainder of the chapter is organized as follows. Chapter 2 presents definitions of terms that will be used throughout the deliverable. Chapter 3 gives the current state of the art of the interface; in particular, it provides an overview on the methods corresponding to pre-processing, post-processing and variations that can be found in literature. Chapters 4 and 5 give the view on the interface from the perspective of the EM and UUV use-case, respectively. Chapter 6 discusses the generic solution of the interface by comparing the use-cases of Chapters 4 and 5. In particular, it finds commonalities and differences between the interface of the EM and UUV. In Chapter 7, the conclusions and a summary of the most important results and achievements are given.

2 Definitions

This chapter provides an overview of the definitions of terms that will be used throughout the remainder of this document. Since two fields are combined in this chapter (the control and the machine learning fields), there are discrepancies between the terms, see, e.g., (Busoniu, de Bruin, Tolic, Kober, & Palunko, 2018). These discrepancies motivate the present chapter. A total of nine main terms are listed in this chapter: State, Reward, Observation, Action, Input, Internal state, Pre-processing, Post-processing, and Variations. The underlying relation between these terms can be seen in Figure 1. The first five terms are briefly mentioned since they are discussed in detail in D2.1. Internal state is an important term as will turn out later in the document, related to the Markov property. The last three terms are the main focus of this deliverable. The definitions are as follows:

Table 2 Definitions in deliverable 2.2.

State	*State* (s) is the signal that is used by the AI agent and forms a representation of the current environment. This choice of *State* is user-determined. The state is obtained or estimated from the *Observation.*
Reward	*Reward* (r) is the signal that represents the goal. The *Reward* is also user-determined from the *Observation*. It is used by the AI agent to update its policy.
Observation	*Observation* (y) is the signal that the system (DT or CPS) outputs from receiving an input. This signal is then *post-processed* to generate the *State* and the *Reward*.
Action	*Action* (a) is the signal that the AI agent generates, based on the *State* and *Reward.*
Input	*Input* (u) is the signal that is entered into the system after *pre-processing* of the *Action.*
Internal state	The *internal state* (x) characterizes the current environment completely. This means that history is implicitly embedded in the internal state. In a Markov Decision Process (Ghavamzadeh, Mannor, Pineau, & Tamar, 2015), we can give the probability of ending in a new internal state from the current internal state and the current input using a state transition function. The internal state satisfies the Markov Property by definition. The *Observation* is usually a function of the Internal state and often has reduced dimensionality (partial observability).
Pre-processing	In this document and in the ASIMOV project in general, pre-processing refers to the manipulation of the signals that the AI agent produces before they are put into the DT or RS. Note that this definition is different from the definition of pre-processing in most literature on data analysis. There, pre-processing refers to the process of manipulating data before it is used for analysis.
Post-processing	*Post-Processing* refers to the manipulation of the *Observations* produced by the system in order to create the *State* and the *Reward* to be used for the AI agent. Note, this is also different from the definition of post-processing in data analysis literature. In data analysis, post-processing concerns the manipulation of data after the main analysis has been performed. In this document, the 'post' refers to the fact that the signals have been produced by the system.
Variations	The system on which the training of the AI agent is being done is one instance of the digital prototype, which corresponds to one intended purpose. Variation can be used: • To improve the AI agent's robustness by changing the DTs properties slightly and therefore imitating artefacts of the RS (e.g., manufacturing anomalies). This also can account for uncertainty in the internal state (partial observability). • To make the trained AI agent applicable in other intended purposes, in which the CPS behaves substantially differently. Training the AI agent with such variations can be seen as a way to enable a zero-shot approach for calibration of the RS (Degrave, et al., 2022).

3 State of the art

In this section, the state of the art regarding pre-processing, post-processing and variations as defined in Chapter 2 is presented.

3.1 Pre-processing

The goal of pre-processing for interactions with the system is to reduce the complexity of the RL problem. While the *Inputs* (u) might be required to be very complex in order to maximize the obtained *Rewards* (r), it might be possible to obtain almost the same *Rewards* with simple *Actions* (a) using smart pre-processing. Additionally, the pre-processing block can be used to change the *Inputs* (u) in order to increase safety or satisfaction of constraints.

3.1.1 Input Shaping

In this section, different methods for shaping the input are detailed. In general, the dimensionality of the input space for typical CPSs can be very high. For example, consider the problem of generating a trajectory for a pen in order to replicate handwriting. This problem could be formulated as finding a sequence of points that, when connected, produce written text. Finding an appropriate sequence of positions on a 2D plane which together constitute handwritten text presents an extremely complex task. Using what we call input shaping, we can reduce the complexity and learn to solve the problem more quickly. There exist many ways of doing this, such as discretization of the action space into a finite set of positions, parametrizing the inputs into a reduced set of sub-trajectories, learning an encoding of possible moves, and smart interpolating between points or actions. An important concept to consider is whether the shaped input satisfies controllability of the system.

3.1.1.1 Discretization

The first question that arises is why the action space should be discretized in the first place. Firstly, the theoretical analysis behind discrete action space problems is more developed than that of continuous action spaces. However, many CPSs have a continuous input space by design, which results in an infinite set of possible inputs at every timestep (Tang & Agrawal, 2020).

There are multiple ways in which the action space can be discretized. The simplest is to consider a continuous input space and to simply divide the space into many discrete parts. This method causes a problem however, as for a high dimensional input space, the number of discrete actions quickly rises. An alternative approach is to factorize the joint distribution over discrete actions, which achieves the same discretization but reduces the amount combinations which are possible (Tang & Agrawal, 2020). These considerations reveal a trade-off between lowering the dimensionality of the actions and maintaining the accuracy of the inputs, as the methods that result in lower dimensionality also compromise on the possible inputs that can be given.

Another consideration to make when discretizing actions is whether the actions should be relative or absolute. This concept is best illustrated with an example. Consider UC1, the electron microscope. In this use case, the objective is to calibrate the instrument by tuning knob values which improve the resulting image quality. For instance, if the image is under-focused the focus knob should be turned until focus is achieved. The value of the signal that is being sent to the focus lens, which is a voltage, is not necessarily relevant to the calibration solution and might have an offset in between different attempts. Additionally, using relative actions can improve numerical conditions, and can turn static problems into dynamic problems by creating trajectory through the input space. Note that when there are input constraints, the relative actions should be canceled when they would result in violation of the constraint.
The concept of discretization can be generalized into parametrized inputs, which are introduced in the next section.

3.1.1.2 Parametrized inputs

In order to simplify the AI training problem, we can reduce the complexity of the Action space compared to the Input space that is used in the environment through parametrization. In general, a parametrized input looks as follows:

$$u_k = \phi(a_k)$$

where ϕ is the parametrization function. Note that for multi-dimensional actions and inputs, the parametrization becomes a matrix of functions. Note that discretization is a special case of parametrization where we restrict the AI Agent to choose from a finite set of (indexed) actions a_k, after which a corresponding u_k from the input set is fed into the system.

Additionally, the input can be parametrized using basis functions which makes use of aspects of the actions generated by the AI agent. A general basis function parametrization looks as follows:

$$u_k = \sum_{i=0}^{n} \theta_k[i] \cdot \phi_i(a_k),$$

in which $\theta_k[i], \forall i \in \{1, \dots, n\}$ are the parameters and $\phi_i, \forall i \in \{1, \dots, n\}$ are the basis functions (Yamaguchi, Takamatsu, & Ogasawara, 2009). Examples of basis functions include radial basis functions, derivative approximations, etc. It is important to note that appropriate selection of the basis functions is inherently dependent on the system. Also note that a_k can be discrete or continuous. In Reinforcement learning, the AI Agent selects the values for both the parameters and the actions simultaneously. This can be viewed as making a high-level choice between different actions, and then supplying this choice with parameter values.

Examples of RL with parametrized inputs can be found in literature. For instance, in (Hausknecht & Stone, 2015), Deep RL is applied to a soccer robot. In the paper, the actions that the AI agent can perform operate on a higher level than the soccer robot's input signals. The AI agent can choose between different discrete actions such as moving on the field or shooting the soccer ball and has to assign continuous parameters such as the movement distance or shooting direction to complete the command. (Masson, Ranchod, & Konidaris, 2016) analyzes RL with an input parametrization similar to (Hausknecht & Stone, 2015). The paper focuses on analyzing the learning problem under this parametrized structure. As an extension, the set of discrete high-level commands can be chosen by the AI agent itself. This method is called encoding/decoding.

3.1.1.3 Encoding/Decoding

Often, the inputs required to be performed by the AI agent in order to achieve the objectives for the system are very complex in nature. However, it is also true that these same complex signals can often be grouped into families with similar characteristics. These groups operate on a higher level and are usually of lower dimension and less complex. This helps the overall AI training problem by reducing the complexity of the action space.

An encoder can use AI techniques in order to learn a mapping from complex input signals to a set of higher-level commands to choose from. For instance, in an EM, a sample manipulation platform can be controlled using commands such as 'higher focus' or 'lower focus'. These commands can then be decoded back into lower-level signals, i.e., trajectories of currents to a motor driving the stage to desired point using low-level feedback controllers.

As an example, consider openings in chess. Rather than interpreting every move as an action, an AI agent might instead be able to choose from a catalogue of known valid openings which are sequences of multiple moves. This way, there are fewer options to choose from which enables faster training.

The state of the art regarding action encoding learns these higher-level commands using supervised learning. In (Chandak, Theocharous, Kostas, Jordan, & Thomas, 2019), an "embedding" is inserted as an intermediate step of the RL policy and represents the high-level commands. The high-level policy (state-to-embedding) can be learned by the AI agent simultaneously with the set of high-level commands (embedding-to-input mapping). This can be seen in Figure 2.

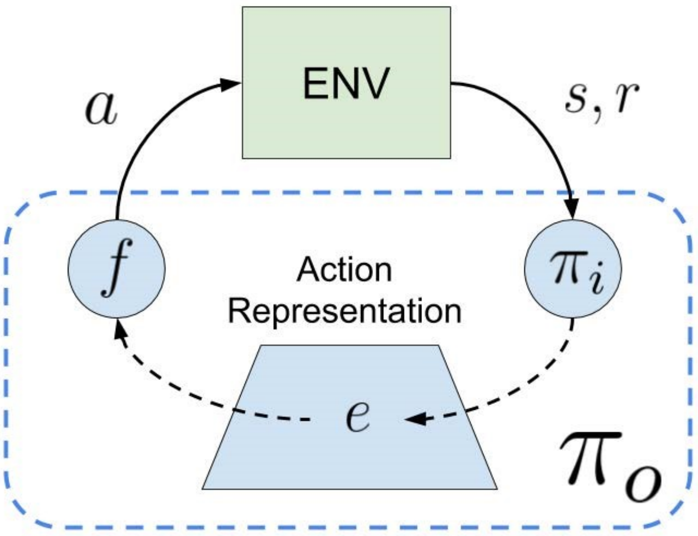


Figure 2 structure of "action representation" in (Chandak, Theocharous, Kostas, Jordan, & Thomas, 2019). The AI agent learns to map states (s) and rewards (r) to inputs (a) by separating into a state-to-embedding mapping and an embedding-to-action mapping. The embeddings are a reduced dimensionality abstraction of the possibly high-dimensional input space.

In (Kim, Yamada, Miyoshi, Iwata, & Yamakawa, 2020), the same type of idea is employed except using an autoencoder. This autoencoder learns different high-level (called "latent space") commands based on a set of expert demonstrations. Afterwards, the AI agent learns a policy network that outputs the encoded actions in the latent space rather than the system inputs.

3.1.1.4 Interpolating input signals

Most real-world applications operate in continuous time, while their digital controllers operate in discrete time. This poses the question: what input should the physical system use in between discrete timesteps of the controller? Answers to this question come in the form of interpolation. In control applications, there are many common interpolation strategies such as zero-order-hold (Aström, Hagander, & Sternby, 1984), generalized hold (Chou, Bruell, Jones, & Zhang, 756-761; Bai & Dasgupta, 1990). The theory behind these methods, as well as analysis of benefits and shortcomings, are quite mature (Aström, Hagander, & Sternby, 1984). Special attention should be paid to ensure that the continuous-time system still behaves as desired, for instance with stability or without inter-sample behavior.

3.1.1.5 Reachability

Reachability is a classical concept in the field of control theory, which becomes relevant for RL when pre-processing the system inputs. The reachability of a system is the ability to bring the system from a specific initial state to any other state. By shaping the input, we might lose reachability of the system, potentially making it impossible to reach high-rewarding states. This is illustrated in a simple example. Consider the following nonlinear state-space which describes a dynamic system, which we want to control to $x_k = 0$:

$$x_{k+1} = x_k^2 + u_k.$$

If we discretize the action space as follows: $u_k \in \{-1,1\}$, then, for certain initial values of x_k, e.g.,$x_k = 10$, it is impossible to control x_{k+1} to the origin given the allowable inputs. The idea of explicitly considering reachability for problems with input limitations can be found in recent literature on RL. Specifically, safe operation through recursive constraint satisfaction can be ensured by using reachability analysis (Yu, Ma, Li, & Chen, 2022). Alternatively, reachability in combination with Lyapunov analysis to ensure stable operation within the constrained space (Huh & Yang, 2020).

3.1.2 Constraint Handling

Systems have physical limitations while operating that should be taken into account in the control architecture. These limitations may contain constraints on the feasible input space, or constraints on internal states/observations. Constraints on the inputs are for example valves that have physical constraints on their positions, or voltages that only have a limited domain. Constraints on internal states or observations can either be physical constraints that certain positions cannot be reached, or safety constraints to ensure safe operation of the system. The process of dealing with all these constraints is called constraint handling. This section elaborates on the current state of the art on constraint handling.

There are different approaches to deal with input and output constraints. (Glattfelder & Schaufelberger, 2003) gave an extensive overview of different techniques that can be applied on controllers, mainly focused on classical controllers. Controllers are designed to deal with input constraints, output constraints, or a combination of both. The controllers in the overview are viewed from three perspectives: structure, transient response, and stability. This overview is useful when guarantees about stability have to be given. Since the overview is mainly focused on classical controllers, the proposed methods do not take optimality into account.

A popular method that can deal with input and output constraints, as well as optimality is model-predictive control (MPC), see e.g. (Camacho & Alba, 2013; Kouvaritakis & Cannon, 2016; Grüne & Pannek, 2017) In MPC a model is used for prediction of future outputs. Based on a given cost function and constraints, an optimal sequence of control inputs is calculated. Only the first control input is applied, and afterwards the procedure is repeated. Recent advances in MPC are focusing on combining data-driven techniques with MPC (Hewing, Wabersich, Menner, & Zeilinger, 2020; Berberich, Köhler, Müller, & Allgöwer, Data-driven model predictive control with stability and robustness guarantees, 2020; Berberich, Köhler, Müller, & Allgöwer, Data-driven model predictive control: closed-loop guarantees and experimental results, 2021).

In RL, constraint handling is covered in the field of safe RL. A survey on this topic can be found in (Garcia & Fernández, 2015). In safe RL the goal is still to maximize the expected rewards (similar to ordinary RL) but also to respect safety constraints on either states or actions. (Garcia & Fernández, 2015) considers two types of safe RL:

- Optimization criterion, where a safety factor is included in the reward calculation.
- Exploration process, where the goal is to only select safe actions.

Both types have their benefits and disadvantages. Depending on the type of problem, one type may be preferred over the other.

One of the recent advances within safe RL is the use of a predictive safety filter (PSF), see Figure 3 (Wabersich & Zeilinger, 2021). The idea behind the PSF is that the RL controller still has all the freedom to give actions, while the PSF verifies whether the actions are allowed with respect to the pre-determined constraints. (Wabersich & Zeilinger, 2021) uses MPC techniques within the PSF, to ensure a safe system.

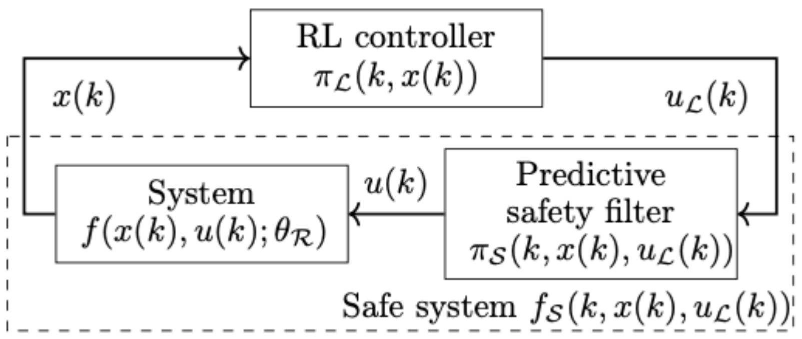

RL controller
$\pi_{\mathcal{L}}(k, x(k))$
$x(k)$
$u_{\mathcal{L}}(k)$
System
$f(x(k), u(k); \theta_{\mathcal{R}})$
$u(k)$
Predictive safety filter
$\pi_{\mathcal{S}}(k, x(k), u_{\mathcal{L}}(k))$
Safe system $f_{\mathcal{S}}(k, x(k), u_{\mathcal{L}}(k))$

Figure 3 Structure of a system with an RL controller attached, which uses a predictive safety filter to ensure safe operation. Figure adopted from (Wabersich & Zeilinger, 2021).

3.1.3 Tuning controller parameters

To further simplify the RL problem, one can make use of an existing standard feedback controller design with some freedom in the form of tunable parameters. Then, the parameters of this controller can be tuned using the AI agent, possibly dynamically. Notions like stability and robustness can be enforced due to the known character of the controller. In (Qin, Zhang, Shi, & Liu, 2018), RL is used to dynamically tune three parameters of a proportional-integral-derivative (PID) controller. A PID controller is a widely used feedback controller because of its effectiveness and simplicity. The user only needs to tune three parameters in the form of the proportional (P), integral (I), and derivative (D) gains. By allowing the AI agent to tune these parameters, the controller can become adaptive to changes in the system. Furthermore, the properties of the feedback system can be analyzed using mature theoretical tools available to PID feedback structures. The structure of this concept is shown in Figure 4.

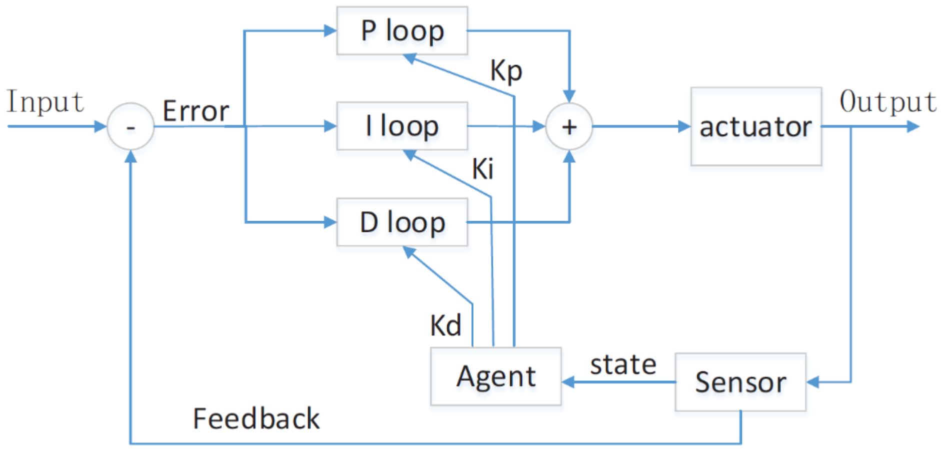


Figure 4 Structure of RL-tuned PID controller.

3.2 Post-Processing

The system (either a RS or its DT) and the AI agent are connected to each other by the post-processing step. Post-processing converts the information available from the system in such a way that it can be used by the AI agent. The post-processing can be divided into several categories:

1. Information preparation
2. State shaping
3. Reward function formulation

In Figure 5, a general structure of the post-processing is visualized, containing the above mentioned three categories. Within the post-processing, first all the available data is prepared in the *information preparation* part. Information preparation includes collecting all the available data and then making it a suitable, cleaned up, dataset for the next steps, without affecting the dimensionality. For preparation one could think of restoring corrupt data or normalization. The second part of post-processing is *state shaping*. State shaping can either add dimensions to the state or reduce it. Adding dimensions is useful when there is not enough information available in one observation, which can be done by adding history of previous states to the current state or make use of observers. State reduction can be used when the dimension of the observation is very high (e.g., an image), while the usable information in it can be captured in a low-dimensional vector. The purpose of state reduction is to decrease the load on the AI agent by reducing the dimensionality of the state information. The last category is the *reward function formulation*. The goal of that part is to formulate a reward function which the AI agent should maximize. The reward may depend on the observations, the estimated states or the control inputs.

Information preparation and state reduction are two well-known concepts in Big Data Analytics (see e.g., (García, Ramírez-Gallego, Luengo, Benítez, & Herrera, 2016)). Adding information by using observers is a concept from Control Theory to restore the Markov property, which is important in the learning process of an AI agent (Sutton & Barto, 2018). The remainder of this section is structured as follows: Firstly, an overview of methods of information preparation is given. Secondly, state shaping is explained and concepts are given that can help restoring the Markov property or reduce the load on the AI agent. Furthermore, the formal definition of the Markov property, Markov Decision Process (MDP) and Partially

Observable Markov Decision Process (POMDP) are given. At the end the reward function is elaborated in more detail.

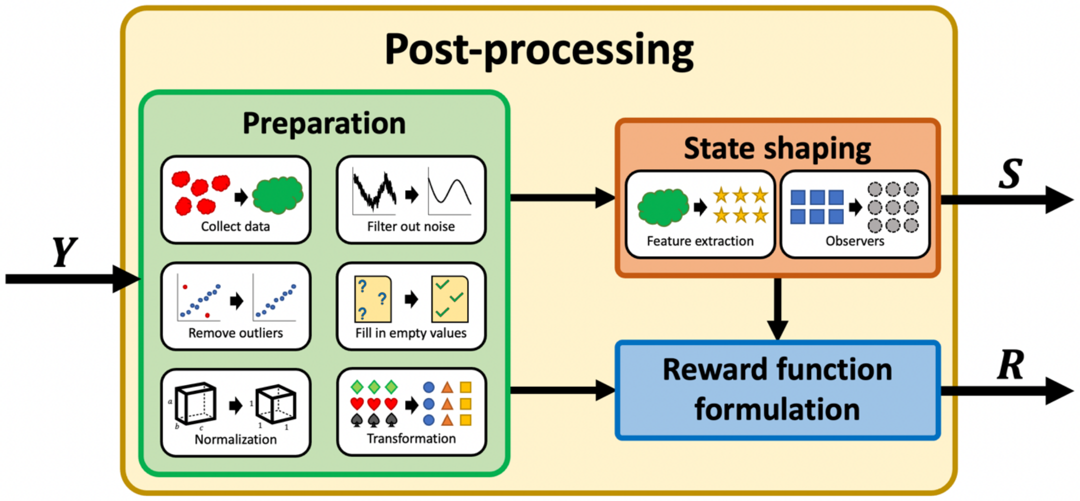


Figure 5 Overview of post-processing process.

3.2.1 Information preparation

The first step within post-processing is information preparation, where all available data is collected and made suitable for the subsequent steps. In the literature there are several authors who wrote methods on this topic, see e.g., (García, Ramírez-Gallego, Luengo, Benítez, & Herrera, 2016; Saleem, Asif, Ali, Awan, & Alghamdi, 2014; Famili, Shen, Weber, & Simoudis, 1997). Among all the methods mentioned, there are six main methods covered in almost every overview:

1. *Integration:* gather all available data and merge it. This should be done carefully to prevent, for example, duplicates.
2. *Noise handling:* measurements that contain irrelevant high frequent noise should be treated to suppress that noise.
3. *Cleaning:* removing and correcting bad data. Data which does not make sense, should be removed from the dataset. An example of cleaning is the removing of outliers.
4. *Missing data imputation:* within control systems this includes the handling of different sample times of sensors. The AI agent operates at a particular frequency, so the state information should be available at the same frequency. More information regarding missing data imputation can be found in (Luengo, 2012).
5. *Normalization:* can be used to scale the data.
6. *Transformation:* convert the data such that it becomes easier for processing in the subsequent steps. Transformation includes changing the units of the data or the rounding of numbers.

These six methods are visualized in Figure 5.

No major recent advantages in information preparation have been made, since the mentioned methods are highly dependent on the specific use case and the expertise of the researcher (Mansingh, Osei-Bryson, Rao, & McNaughton, 2016). General methods are given in (García, Ramírez-Gallego, Luengo, Benítez, & Herrera, 2016; Saleem, Asif, Ali, Awan, & Alghamdi, 2014; Mansingh, Osei-Bryson, Rao, & McNaughton, 2016), however, it is not possible to determine beforehand which methods should be used.

3.2.2 State shaping

A crucial part within RL is the state. The state contains all the information available to the AI agent at a particular time. The state can either be directly the output of either the RS or DT (solely with some preparation steps), or the result of some intermediate state-shaping steps. Depending on the type of process, different approaches have to be taken in the state-shaping part of the post-processing. An important process property that relates to this is the Markov property. The first part of this section is therefore dedicated to the Markov property with its various types.

When the type of process is known, the next steps within state shaping can be taken. This includes restoring the Markov property if that is required, extracting features from the output of the process, and/or building observers to estimate hidden information. These steps are elaborated in the remainder of this section from the second part onwards.

3.2.2.1 Markov Property

The goal of the ASIMOV project is to optimize the process of determining optimal control settings for complex high-tech systems by use of Artificial Intelligent (AI) technologies. In WP3, where the AI-agent is developed, RL is selected as a promising direction to achieve the goal of the ASIMOV project. The performance that the AI agent can achieve depends on the information (the state) that the complex high-tech system (the process) gives to the AI agent. A property that can be assigned to a process is the so-called Markov property. When a process is said to have the *Markov property*, it means that the new state X_{t+1} only depends on the current state X_t, and not on the states before.

In total there are four different Markov types, which are given in Table 2. These four types can be distinguished based on observability and whether actions are involved or not. If the complete internal state of the process is available at each time instance, it is said to be fully observable. In a lot of processes, only a part of the internal state is available at each time instance since not every relevant property can be measured/observed at each time step. Therefore, these processes are said to be partially observable. If there are no actions involved in the process, the process is autonomous and acts independently of the AI agent/user. On the other hand, the process is controlled if the AI agent/user can interact with the process. In the remainder of this section, these four types are elaborated in more detail.

Table 3 Overview of Markov types.

	Fully Observable	**Partially Observable**
Autonomous	Markov Chain	Hidden Markov Model
Controlled	Markov Decision Process	Partially Observable Markov Decision Process

The first type is a *Markov Chain*, which is autonomous and fully observable. Formally, the Markov Chain can be written down as:

$$Prob(X_{t+1}|X_t, \dots, X_0) = Prob(X_{t+1}|X_t)$$

The equation states that the transition from current state X_t to new state X_{t+1} is independent of the previous states.

The second type is a *Markov Decision Process* (MDP), which is controlled and fully observable. If a process is an MDP, the new state X_{t+1} only depends on current state X_t and current action U_t. MDP is an extension of the Markov chain and satisfies:

$$Prob(X_{t+1}|X_t, U_t, \dots, X_0, U_0) = Prob(X_{t+1}|X_t, U_t)$$

Since actions U_t are involved, this Markov type is noted as a decision process rather than a chain.

The third type is a *Partially Observable Markov Decision Process* (POMDP), which is controlled and partially observable. In a POMDP, observations (Y) are typically not the same as (do not represent the same information as) the internal state (X). The observations represent only a part of the information collected in the internal state. The underlying process however, with internal state X, does satisfy the properties of an MDP. Processes which are considered from the perspective of the observations do not necessarily satisfy the Markov property as an MDP:

$$Prob(Y_{t+1}|Y_t, U_t, \dots, Y_0, U_0) \neq Prob(Y_{t+1}|Y_t, U_t)$$

Since only limited information of the process internal state is considered, adding previous states could change the transition probabilities for the next state.

The fourth type is a *Hidden Markov Model* (HMM), which is autonomous and partially observable. The difference between the HMM and POMDP is that HMM is autonomous (as the Markov Chain). Similar to POMDP, HMM does have observations which give only partial information rather than full state information. The HMM can formally be written down as:

$$Prob(Y_{t+1}|Y_t, U_t, \dots, Y_0, U_0) \neq Prob(Y_{t+1}|Y_t, U_t)$$

In the remainder of this report, we will say that the state is Markov if state X satisfies the conditions of an MDP.

3.2.2.2 Restoring Markov property

In many processes that have to be controlled, the output only partially represents the internal state. In these cases, the process is a POMDP (assuming that the internal state does satisfy the properties of an MDP). Therefore, using the output of a process directly as state, results in the state not being Markov. In RL however, all algorithms in (Sutton & Barto, 2018) assume that the state available to the AI agent is Markov and satisfies the MDP properties. This assumption emphasizes the need for restoring the Markov property to make the state for the AI agent Markov again. There also exists methods that use Long Short Term Memory (LSTM), which are applicable to POMDPs, see e.g. (Hausknecht & Stone, Deep Recurrent Q-Learning for Partially Observable MDPs, 2015).

Two solutions are briefly discussed in order to restore the Markov property: (i) add a history of the process to the state; and (ii) use (state) observers. These two solutions can be understood from the following traditional and simple example. Suppose that we want to make decisions for the pedal of an autonomous car (brake or accelerate) based on positioning data (e.g., obtained from GPS). Using only the current position measurement to this effect is clearly not enough as it neglects velocity. In fact, braking or accelerating decisions will be very different if the car is still (zero velocity) or moving at high speed, while having the exact same position. One solution is to keep track of the history of positions. Another solution is to estimate the velocity based on previous position measurements. Both of these related solutions allow for informative pedal decisions to be taken.

Adding history is one way to transform a POMDP to an MDP in certain simple cases. However, measurements are often noisy. In such a case, the larger the history window, the better the noise in the measurements can be mitigated. For such problems (which encompass most cases of interest) the complete process history should be stored. However, this is often infeasible.

It is well-known (see (Bertsekas, 2012)) that the state probability distribution given the measurement history is a sufficient statistic for decision making. This means that all the information contained in the process history can be summarized in the state probability distribution given the previous measurements.

An ideal state observer is then one that provides this state probability distribution given the measurements. This is known as the Bayes' filter. A special case, under strong assumptions such as Gaussianity and linearity, is the Kalman filter.

In general, this ideal Bayes filter is hard to synthesize and run due to the curse of dimensionality and therefore other forms of observers are used. Such observers aim at simply recovering the internal state (and not the full state probability distribution) from previous measurements. Typically, an observer estimates the hidden internal states using the available observations in a recurrent manner. Such an observer turns a POMDP into an MDP when the state is replaced by the state estimation. This is not an optimal procedure as the information in the state probability distribution is lost, but an often used one in practice. Since the literature on observers is extensive, the complete next section is dedicated to them.

3.2.2.3 Observers

Knowledge of a system's internal state (X) is useful for the purposes of controlling the system to exhibit desired behavior. In many cases, the full internal state is not directly measured, as detailed in Section 3.2.2.1. However, indirect effects of the internal state can be visible through observations. A *state* *observer* or *state estimator* can estimate the internal state (X) from measurements of these observations (Y) along with the past system inputs (U).

As seen from the example, observers typically require knowledge of the underlying prescribing equations. In this case, the relationship between position and velocity.

The simplest version of an observer is a Luenberger Observer (Luenberger, 1964). Consider a general linear dynamical system in state space formulation:

$$\dot{x} = Ax + Bu + w$$
$$y = Cx + v$$

Where $x \in \mathbb{R}^n$ are the internal states, $y \in \mathbb{R}^o$ are the output measurements or observations, $u \in \mathbb{R}^m$ are the system inputs, $w \in \mathbb{R}^n$ are disturbances acting on the system, $v \in \mathbb{R}^o$ is measurement noise, and $A \in \mathbb{R}_{n\times n}$, $B \in \mathbb{R}_{n\times m}$, and $C \in \mathbb{R}_{o\times n}$ describe the behaviour of the system. Now, if $o < n$, then at least one of the internal states is hidden (note that this condition is not sufficient or necessary). In order to estimate this state, we use the observer described by:

$$\dot{\hat{x}} = A\hat{x} + Bu + L(y - \hat{y})$$
$$\hat{y} = C\hat{x}$$

where $\hat{x} \in \mathbb{R}^n$ is our state estimate, $\hat{y} = \mathbb{R}^o$ is our output estimate and $L \in \mathbb{R}_{n\times o}$ is the observer gain. We update our estimate based on the difference between our output estimate and the real measured outputs. Under certain conditions, the state estimate $\hat{x}$ will eventually converge to the real internal state x.
We can design the observer gain L using an optimization problem which minimizes the least squares estimation error, in which case the observer becomes a Kalman filter ((Bertsekas, 2012), Appendix E).

A Bayes' filter generalizes the concept of a Kalman filter to application for general POMDPs, see Section 3.2.2.1. Similarly, the Bayes' filter uses recursive state estimate updates based on measured outputs and inputs. The core idea behind these updates lies in Bayesian statistics, namely Bayes' rule:

$$Prob(X|Y) = \frac{Prob(Y|X)Prob(X)}{Prob(Y)}$$

in which we update our belief of the internal state X based on the new observations Y. There exist observers for nonlinear dynamical systems which also use a model of the system and observations to update estimated beliefs, such as the extended Kalman filter or high-gain observers (Khalil, 2015).

Additionally, observers can be used for simultaneous state and parameter (system property) estimation, in which the internal state of the model is appended with a state for the parameter to be estimated. This concept might be of special interest to ASIMOV Task 2.1, in which the relevant parameters of the DT are identified.

In ASIMOV, the idea of observers can be used to restore the Markov property for POMDPs. Additionally, Use Case 1 can potentially be reformulated as an estimation problem, in which estimation of the internal state of the electron microscope can result in easy calibration. Observer-like logic is key in estimation problems which underlines its relevance.

3.2.2.4 Feature extraction

Depending on the application, the output of the system in the form of the observations can take on different forms and different dimensionality. Simple examples of the output are discrete with few possible values, or a single continuous number. A bit more complex is a vector of continuous values. However, in real applications, the output could be a curve (e.g., quantity-over-time) or even an image, such as in the EM use case. The curve is typically described by a set of points; an image consists of gray values for each pixel. Here we give some examples and ideas on how to extract features of the complex shape that are still sufficiently informative as a Markov state. Here, domain knowledge and problem experience play a large role in the choice.

Principal Component Analysis (PCA) is a well-known and established technique in multivariate statistics. In machine learning, it is considered as belonging to the class of unsupervised learning. The starting point is a dataset with many variables that is summarized using only a few scores belonging to the first, second, third, … principal components. As an example, if a dataset consists of people with many variables describing their height, length of arms, legs, belly size, etc., the first component might pick up on the general size of the person. Mathematically, the first component is a linear function of the original variables. In PCA, the first component is the linear combination with the most variance in the dataset. The second component also finds the most variance but is “orthogonal” to the first; here it might be a score on the axis along fat vs thin. This way, the original possibly large number of output variables is approximated by a summary using a short vector of scores, all using linear expressions.

There are other variants of this theme using linear expressions; Factor analysis from statistics addresses similar problems to PCA but is more model-based. Linear Discriminant Analysis (LDA) can be applied if there are known groups in the data; it finds components that are most suitable for separating the groups, i.e., a multivariate dataset where each datapoint belongs to a pre-specified group. For more details on Factor analysis and LDA, see (StataCorp, 2007).

Standard examples often use tens or hundreds of variables as a starting point. Curves and images may be treated as a long vector of values and could be analyzed using the same tools. However, note that positional information is lost here. A somewhat dated example is recognition of faces (Turk, Pentland, Belhumeur, & Hespanha, 1991).

There are more elaborate variants of the discussed techniques as well, described in section 14.5 of (Hastie, Tibshirani, & Friedman, 2009). There, non-linear variants of PCA are discussed, called Kernel Principal Components. Section 14.6 discusses non-negative matrix factorization which is a variant for non-negative scores and strictly positive data, such as grey values in an image. Section 14.7 explains Independent Component Analysis (ICA). ICA tries to find scores in the data such that the projection scatterplots of the scores are as different as possible from multivariate normal. An example where this is beneficial is the classical cocktail party problem where various microphones in the room pick up sound from several independent sources such as talking people, music, etc. The sound time series from each of the microphones form the input; but there is no spectral analysis or FFT applied. PCA would find the most common part of the sound between the microphones, and ICA tends to find the separate sources of sound.

Autoencoders are another technique to handle complex inputs, see (Efron & Hastie, 2021), section 18.3. Here, a complex input is fed into a neural network with several layers. The target output is taken to be a copy of the input. The middle layer has relatively few nodes. As the neural net tries to translate input to (identical) output, the information has to pass through the middle layer. The model therefore has to “code” the information using only the information corresponding to the few nodes in the middle layer. Such a model is considered in the UUV use case, not for defining a state but to generate a reward. The idea is to learn an autoencoder based on a dataset of previous simulation runs. Then, a new simulation run is passed as input to the trained network. The goal is to assess whether this new point is “surprising” in light of the earlier simulations. If the new point is more difficult to predict, i.e., the autoencoder’s error is large, then this new point contains new information and therefore gives a higher reward.

In the EM use case, the output consists of an image on which the organization has much fundamental knowledge from physics and years of experience. Here, a spatial Fourier transform of the image gives a new image but with easier to recognize patterns. From here, it is also easier to apply *filters* to block certain frequency bands (e.g., block higher frequencies if the information is in the low frequencies). This example uses 2D images; a similar 1-dimensional variant is the domain of time series, signal processing, which finds many applications in audio signals.
Curves as output of some process may be parameterized or summarized by making use of the understanding of the problem. Here we think of an output $y = f(t)$ for sampled points, typically equidistant. Examples of the predictor t could be time, angle designating a point on a round object, or location. A simple approach could be to work with simple summaries of $f(t)$, such as mean, standard deviation, or percentile-based such as median, IQR, the 95-percentile. Also, these could be made of horizontal intervals. More elaborate summaries could be the (t, y) coordinate of where the curve is steepest.

Alternatively, understanding of the problem could propose a parametric description of $f(t)$, such as $y = f(t) + b \cdot e^{-ct}$ with parameters a, b, c. One observation gives many points on the curve $(y, f(t) + \mathrm{error})$ which can be fit using e.g., least squares. The resulting parameter vector (a, b, c) is then the state summary. Other common fits are polynomials, straight lines, and the Fourier transform for time series mentioned above.

3.2.3 Reward function formulation

Part of the post-processing is the reward function formulation, where a reward signal is created based on the systems' outputs. The reward signal should express the performance on a predetermined task in a single value, $R \in \mathbb{R}^1$. The AI agent then maximizes the total cumulative rewards, such that the performance of the AI agent is as good as possible in the long term. The reward function is not the place where the trajectory to the goal should be shaped in terms of a-priori knowledge. The reward function should include the overall objective and not reward intermediate steps deemed reasonable by the user. By doing so, we let the AI agent figure out the way to the objective itself with as much freedom as possible. In general, the reward function contains not how you want to achieve an objective, but only what you want to achieve (Sutton & Barto, 2018). In the remainder of this section several publications regarding reward function formulation are highlighted and discussed.

In (Eschmann, 2021), an overview with the history, links to behaviour sciences and evolution, and surveys on reward function design is given. A distinction is made between sparse and dense reward functions:

- *Sparse reward functions*: Rewards are only given to the AI agent when the final objective is achieved.
- *Dense reward functions*: Informative intermediate rewards are given to the AI agent while approaching the final objective, although the final objective has not been achieved yet.

Preference is given to dense reward functions to guide the AI agent to the final objective. Sparse rewards may be infeasible in terms of training time, since it could take a long time before rewards are found at all. If sparse reward functions are used, the training time can be decreased when curiosity-based exploration methods are used. Other popular methods to improve exploration when only sparse rewards are available are reward shaping and intrinsic motivation. The first theoretical publication on reward shaping is (Ng, Harada, & Russell, 1999), where potential-based reward shaping (PBRS) is formalized. In PBRS a term is added to the original (sparse) reward function to already achieve rewards in the vicinity of the sparse rewards. The disadvantage of PBRS is that a-priori knowledge is included in the reward function, something undesirable according to (Sutton & Barto, 2018). At intrinsic motivation rewards are given when new states are explored. This method encourages the exploration of new states in order to speed up the search for sparse rewards. The foundation for intrinsic motivation in RL is laid in (Singh, Lewis, Barto, & Sorg, 2010).

(De Moor, Gijsbrechts, & Boute, 2022) applied PBRS in combination with deep-RL in inventory management. The goal of using PBRS was to improve learning behavior by using an existing policy. The same shaped reward function as in (Ng, Harada, & Russell, 1999) has been used:

$$R' = R + F$$

Where F is the shaped reward term and R' is the new reward function. In (De Moor, Gijsbrechts, & Boute, 2022), F is a feedback term from the existing policy. By using reward shaping, an important condition is *policy invariance*. Policy invariance means that reward shaping should not change the original desired goal. Interesting publications related to PBRS are (Wiewiora, Cottrell, & Elkan, 2003) and (Harutyunyan, Devlin, Vrancx, & Nowé, 2015), where different types of shaped reward functions are discussed.

A completely new direction in reward function formulation is reward machines (RMs). The background of RMs comes from the question whether the reward function formulation should be a black-box for the AI agent or not (Icarte R. T., Klassen, Valenzano, & McIlraith, 2018). As mentioned before, the reward is expressed as a single value in $\mathbb{R}^1$. If the AI agent could have access to the structure of the reward function, it could help to create subproblems to increase the learning rate. RMs consist of pre-determined high-level states of the environment. The language used in RMs is Linear Temporal Logic (Icarte R. T., Klassen, Valenzano, & McIlraith, 2018). An example of an RM is given in Figure 6. The goal is to collect a cup of coffee (c), and reach the office (o), while avoiding obstacles ($*$). The corresponding reward machine can be seen on the right. It consists of four high-level states: u_0 no coffee, not on an object; u_1 Coffee, not on an object; u_2 on an object; u_3 in office with coffee. This level of abstraction can be useful for the AI agent during training. In (Icarte R. T., Klassen, Valenzano, & McIlraith, 2018) also the algorithm Q-Learning for RMs is provided which explicitly uses the structure of the RM. (Camacho, Icarte, Klassen, Valenzano, & McIlraith, 2019) gave a formal language to the RMs, and extended RMs with shaped rewards. Follow up work and extensions are provided in (Icarte R. T., Klassen, Valenzano, & McIlraith, 2022).

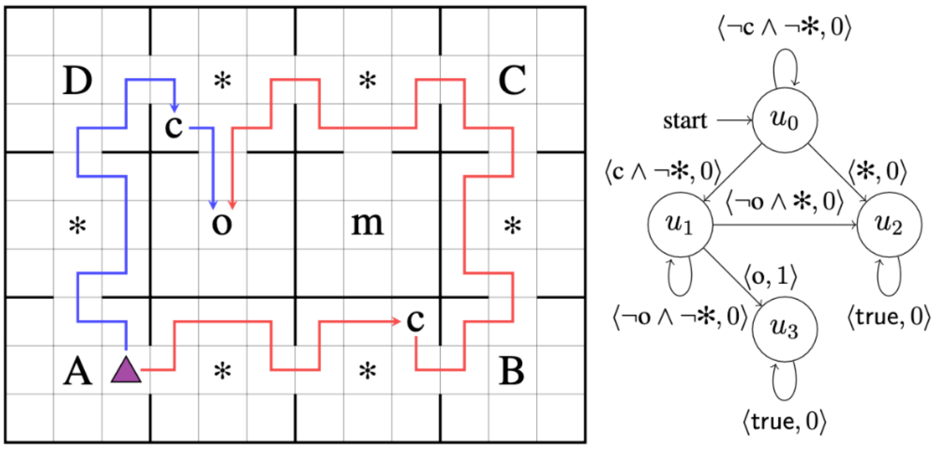

D
C
A
B
c
o
m
c
*
start
u_0
u_1
u_2
u_3
$\langle \neg c \wedge \neg *, 0 \rangle$
$\langle c \wedge \neg *, 0 \rangle$
$\langle *, 0 \rangle$
$\langle \neg o \wedge *, 0 \rangle$
$\langle o, 1 \rangle$
$\langle \neg o \wedge \neg *, 0 \rangle$
$\langle \text{true}, 0 \rangle$
$\langle \text{true}, 0 \rangle$

Figure 6 Example of a reward machine, adopted from (Icarte R. T., Klassen, Valenzano, & McIlraith, 2018).

3.2.4 Data compression in RL-pipeline

As data is key for RL, it makes sense to take a deeper look into the path data takes. Figure 7 tries to describe the interconnections between post-processing, the neural net of the RLA and variations during training, which will be discussed in the next section.

Postprocessing typically describes hand-crafted features that transform the Raw Data, from DT or PT to lower-dimensional data. Postprocessing hereby is typically explainable, as the data transformation happens through pre-defined measures. As it is developed as an outside component of the RLA itself, it also does not change its behaviour during training.

In contrast to that, the neural net inside of the RLA is influenced by training. It transforms the outputs of the post-processing to an action. This transformation effectively does something similar to conventional post-processing, with major differences in explainability. A neural net could, given sufficient neural net capacity, recreate the same post-processing, if data drives it to this decision.

Depending on the ensemble of conventional post-processing the entire data pipeline can be influenced by the variation in different ways. As variation aids data-driven approaches to better generalization, it affects the neural net inside the RLA. The three different data-pipelines show differing amounts of post-processing and neural net sizes, which in turn provide varied levels of explainability and generalization.

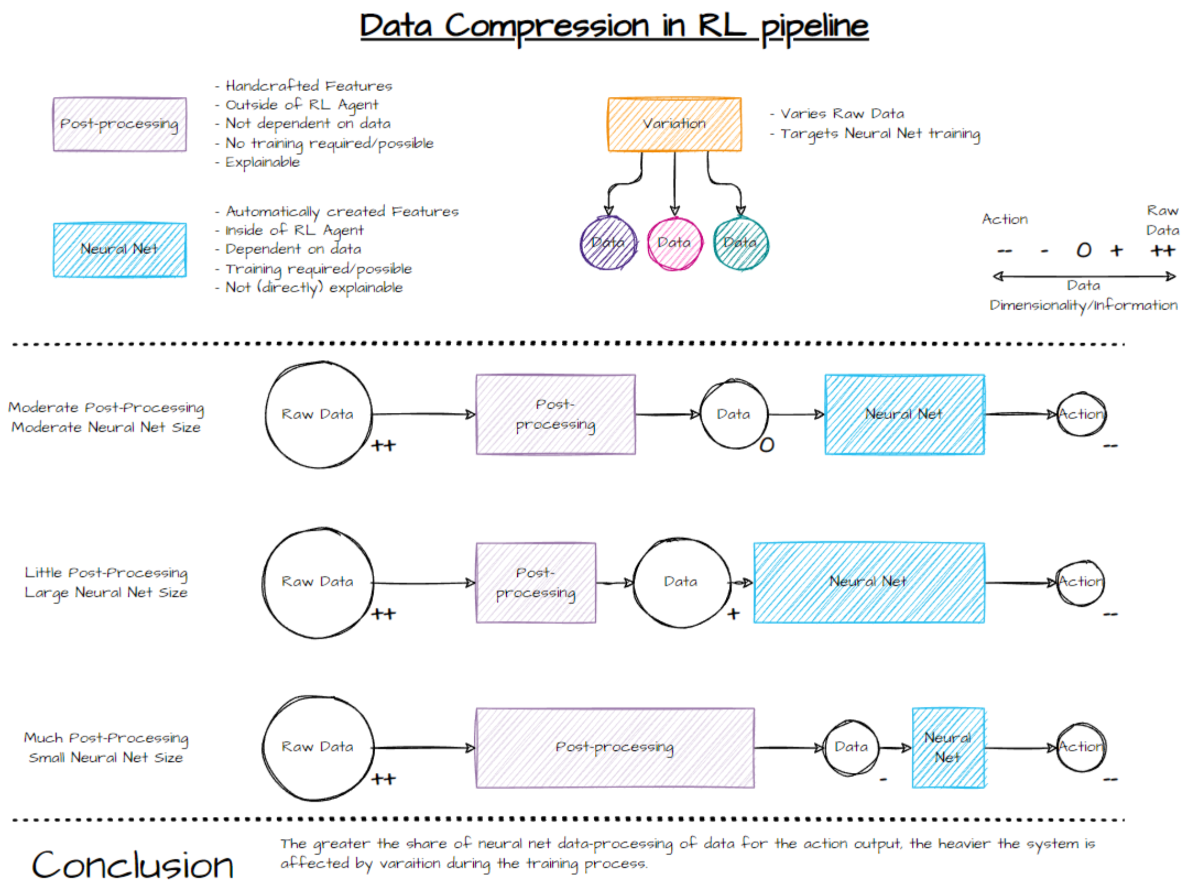


Figure 7 Data compression in RL pipeline.

3.3 Variations

It is of great importance to generate data with the DT, that is relevant for the AI agent's training process. As the AI agent will be paired with a RS in the operational phase, it is therefore not only important for the training data to be accurate, but also to represent imperfections in the behavior of the RS and differences in the configuration of multiple systems.

When producing and operating CPSs, there will always be machine-to-machine differences due to, for instance, manufacturing inconsistencies or wear over time. Therefore, even if a perfect model of one instance of a machine is available, an AI agent trained on this model is not guaranteed to have good performance on other instances of that machine. Engineers at NASA state that in DTs "manufacturing anomalies that may affect the vehicle are also explicitly considered, evaluated, and monitored" (Glaessgen & Stargel, 2012). Furthermore, it is often impossible to have a perfect model of an RS. This can lead to the AI agent relying on features that are only present in the simulation and not in the RS. Hence, there is almost always a model mismatch. Training an AI agent on a model which does not fully match the RS has detrimental effects on the AI agent's performance on the RS. For these reasons, we seek to train the AI agent in a varying environment that switches between multiple machine instances, multiple models or even multiple domains. Good performance on the varying environment is then much more likely to result in good performance on the RS.

A lot of research on ensuring AI agents perform well in multiple domains is done on arcade games. This concept is called generalization. Arcade games are useful to this end because they are well known, require similar strategies (in other words small variations of high-dimensional control tasks) to win, and offer widely available frameworks (e.g., Arcade Learning Environment (ALE)) and benchmarks. A method to evaluate the generalization of a policy is to "inject extra stochasticity to the environments during the evaluation process" (Zhang, Vinyals, Munos, & Bengio, 2018). This and other techniques can be used as a regularizer to prevent overfitting or as an evaluation to detect overfitting. Generalization of the AI agent to perform well in other environments can be seen as the AI agents being invariant to changes in the observation space (Farebrother, Machado, & Bowling, 2018). The authors in (Cobbe, Klimov, Hesse, Kim, & Schulman, 1282-1289) address the issue of an AI agent overfitting to a specific environment (which is the DT in the case of ASIMOV) and find, that “deeper convolutional architectures improve generalization, as do methods traditionally found in supervised learning, including L2 regularization, dropout, data augmentation and batch normalization.” More general-purpose features are learned, which can be adapted to similar problems via fine-tuning. On a similar note, the authors in (Farebrother, Machado, & Bowling, 2018) show that Deep Q-Networks (DQN) overspecialize to the training environment. They offer the solution to reuse learned representations to improve the generalization capabilities of DQN. Further, the approach to fine-tune the weights of a learned neural network is discussed, where the authors found that “reusing a regularized representation in deep RL might allow [them] to learn more general features which can be more successfully fine-tuned.” In our use cases, the trained policy can ideally also be applied, or at least be the training basis for the application, on slightly different physical systems, to cover the product family aspects.

Independent of which of the above-mentioned problems, variation needs to be applied at some point during the training process. In (Moos, et al., 2022), possible ways of handling this variation are discussed. Depending on how the variation is included, the authors distinguish between different types of robustness that can be achieved.

A system is *Transition Robust* when it can cope well with every - even with the least likely - state transition.

It is *Disturbance Robust* when it can cope with a system that is influenced by parameter changes or modeling errors not in control of the AI agent.

Action Robust describes a system that can be controlled, even though the AI agents' actions are manipulated by an adversary.

Lastly, an *Observation robust* system is characterized by the fact that the system is robust against adversarial attacks that try to generate observation data, that is indistinguishable from real data, but influences the RL agents’ output to a great extent.

We can further distinguish which type of parameters are changed during variation. For a common understanding of the different parameter types, the nomenclature used in the ASIMOV T2.1 document shall be used. In this scheme, u is introduced as the input into the system, which is controlled by the AI agent. Typically, these inputs can be set independently and are the means for optimizing the system in ASIMOV.

Disturbances d are another input type that influences the system. Disturbances can be seen as external signals acting on the system, that cannot be controlled by the AI agent. There are however ways to include them in the simulation.

The system parameters c are not inputs, but rather a set of parameters that define the behavior of the system itself. They are, besides the internal model structure, the main influence on how a system translates its inputs to outputs.

The output of each system is described as y.

3.3.1 Three Kinds of Variations

Variation can be introduced in different ways in the DT. Depending on where the variation is introduced and which parameters are varied, we distinguish between different variation types in ASIMOV, that are explained below. For a wider view on the consequences for the use cases, the Use Case Mapping sections provide further details.

3.3.1.1 Disturbance variation

The disturbance variation targets the disturbances d. As they are varied, small changes of the system can be introduced for improved robustness of the AI agent. It forces the AI agent to cope with varying responses of the CPS.

Disturbance Variation can vary after each action of the AI agent.

In the STEM use case, such disturbance variation could include effects that cannot be influenced by the configuration or control of the microscope. This could be room temperature, camera noise, electron source noise, vibrations, etc.

In the UUV use case, such effects could be noise in camera or Radar images, as well as gusts of wind in the virtual environment.

3.3.1.2 System Configuration Variation

Digital Models used in traditional simulations represent a typical instance of the physical counterpart. Due to manufacturing inaccuracies, wear and slightly different operating configurations, multiple systems of the same type can behave differently. These effects are captured by system parameters c, which generates different CPS instances when varied. Ideally, the AI agent should be able to perform well for all common CPS instances, be it new or worn-out, for instance. Furthermore, this can help to overcome the inadequacies of the simulation due to insufficient parameter choices. In other words, we want to avoid that the model overfits one some specific parameters choices, that do not reflect the actual behavior that shall be learned.

This type of variation can also include a switch to a different model of the same system, as proposed in (Khairy & Balaprakash, 2022), which could be first-principles-based or purely data-driven. If the RL loop switches between these models in between episodes, the optimal policy is one that performs well for both models. Hence, bias to one type of modeling error is reduced.

As it is unnatural for a system to change its behavior in terms of system configuration during an optimization process, this variation can only be applied in-between episodes.

System Configuration Variation can also be realized in two different ways, both of which are compatible with the concept explained in Figure 7. One way is to create arbitrary system parameter combinations to build a virtual fleet of fictional systems without a link to RSs. The other way is to create a real fleet of multiple RSs, linked to their DTs. The diversity in the fleet and the resulting differences in the system configuration parameters of their DTs defines the system configuration variation.

Examples of the System Configuration Variation in the STEM use case could be the high-tension voltage, the magnification, targeted resolution, sample type, sample thickness, aperture dimension, position, etc.

The UUV use case offers such possibilities in the form of varying the properties of Vehicle, Sensor and Driving Function.

3.3.1.3 Domain Variation

This is the type of variation which typically results in the largest differences in the environment. It is represented by a major change in the interaction of the system with the AI agent. This can include different outputs y and actions u.
The idea is to switch to a completely different system which is required to perform the same type of task. This way, the optimal policy of the AI agent is more generalized in the sense that it can solve the whole task type, rather than only the specific system. This idea is employed in the RL benchmark AlphaZero.

AlphaZero by Google DeepMind (Silver, et al., 2017) is able to play multiple different games (chess, go, shogi) at an extremely high level using a single AI agent. It is able to do so by exploiting commonalities in high-performing playing strategies in these games. To understand this, consider the games it is able to play. The games are all examples of combinatorial games, where both players have perfect information and make moves in sequence. These games can typically be represented as decision trees, in which the different choices that players can make are different branching paths of the tree. The difficulty in solving such games typically stems from the fact that these trees can grow very large, resulting in many possible ways a game can play out, too many to compare explicitly.

AlphaZero makes use of a neural network to perform tree searches more efficiently, drastically reducing the time to evaluate the tree. The key lies in the fact that certain tree branches should be given more attention than others. This is also how human players approach these games. Rather than evaluating and comparing every possible option, often only a few promising candidate moves are considered. For example, in chess, if one of the move choices leads to a forced checkmate by the opponent, the entire branch following this move choice can be disregarded. This idea generalizes to alpha-beta pruning (Knuth & Moore, 1975), in which any move that can lead to a worse position than the current best candidate is disregarded. Using neural networks, AlphaZero is able to learn an advanced tree-search algorithm with many tricks similar to alpha-beta pruning, making the search even faster. But because tree-search generalizes, AlphaZero is able to perform well for not just chess, but any combinatorial game.

AlphaZero learns the tree-search algorithm through self-play on increasingly complex games. By starting with a simple combinatorial game, the solution to the game can be found by the AI agent in relatively little time. By switching to a new game which is more complicated, the lessons from the simple game can still apply so long as both are combinatorial games. This concept is called *curriculum learning* (Soviany, Ionescu, Rota, & Sebe, 2022). Since the structure of the inputs and the outputs of the system may change with the variation of the domain, a wider view of the consequences is necessary. It also typically introduces manual intervention and happens after successful complete training of the AI agent for one type of system.

Examples in the STEM use case could be the application of the AI agent on a new type of microscope with other controls or estimating aberrations from a different type of output image, e.g., a STEM image rather than a diffraction pattern.

In the UUV use case, the change of Operational Design Domain (ODD) would be an example for such a variation.

3.3.2 How to introduce variations

There are several techniques to introduce variations. In this section, a selection of these techniques is summarized.

3.3.2.1 Random

The first, and easiest approach is to have variations at random. To be able to select variations at random with a uniform distribution, bounds on the variation domain should be provided. If the random values are selected by means of a normal distribution, the mean and standard deviation should be provided. Normal distributions can be very useful when the parameter is approximately known with uncertainties if these uncertainties are normally distributed. If the parameter can have a value within a large domain and shows a uniform distribution on that domain, random values with a uniform distribution are useful. The advantage of random variations is that no bias is introduced. The disadvantage, however, is that this method is extremely time-consuming, as covering every possible combination of variations often results in a huge variation space. Furthermore, there is no preference for specific variations. That results in “common” variations having the same probability as “uncommon” variations.

3.3.2.2 Multivariate structure of randomness

In DTs, it is common to use many random variables, e.g., a random variable for multiple locations and time points. In this situation, the multivariate probability distribution should be considered as well. For instance, if temperature of some component, for each location on the component and time point the overall distribution may be known. However, temperatures close in time and space are likely to be similar. It can also be that there is a structural gradient, e.g., locations close to a heat source are hotter – then the argument applies to temperature deviations from the long term mean of its particular location. We continue the example just with temperature. Independently generated random temperatures would be very different for nearby locations and time points. A plausible physical model is needed with noise terms that can safely be regarded as random. In this example considering just location, where is the overall component's temperature with some distribution (e.g., Gaussian) and with random zero-mean location-specific Gaussian deviations. This equation may be extended with time trends, heating up/cooling down with a speed with random fluctuations that may be PS dependent, e.g. for system, time, location, and where cooling down varies between systems by the expression with zero-mean system level contributions on top of an overall mean. Another example of a very complex multivariate structure is the random generation of worlds in games such as Minecraft (https://www.minecraft.net/) which generates a random landscape with hills, caves, water, trees.

3.3.2.3 Space-Filling

The second approach, that deals with the disadvantage of the variations at random that there is no equal distribution, is to use a space-filling Latin-hypercube design, see e.g. (Gramacy, 2020). In (Eriksson, Johansson, Kettaneh-Wold, Wikström, & Wold, 2000), more techniques are given when experiments must be chosen from fixed ranges. All the aforementioned techniques still have no preference for specific variations; all these techniques determine the variations beforehand, without any interaction with the system.

3.3.2.4 Neural Networks

A technique that introduces variations that became of great interest in the last couple of years within the field of neural network, is the variational auto-encoder (Doersch, 2016). Because of a low-dimensional hidden layer in the neural network, which forces the reproduction of the input to be summarized in a low-dimensional projection. The idea is to remove the encoder part, and only sample from the low-dimensional layer, to produce a high-dimensional layer. The advantage is that a lot of feasible variations can be created, based on real data. Also, anomaly detection can be executed to find rare cases, see e.g., (An & Cho, 2015). Furthermore, techniques such as Generative Adversarial Networks (GANs) may be of interest to introduce variations (Creswell, et al., 2018). GAN's also aim to create "fake" situations that are useful to train a network.

3.3.2.5 Curriculum Learning

Within curriculum learning, variations are introduced by task generation. In task generation, sub-targets are determined that the AI agent should solve. Ideally, these tasks have an increased difficulty. The tasks are created manually. Automated task-generation is still an open-research question (Narvekar, et al., 2020).

3.3.3 How to convert DT performance to RS performance?

The goal when training an AI agent with a DT should be that the learned policy can be applied to the RS directly. In accordance with (Degrave, et al., 2022), this can be seen as a zero-shot approach. If the performance of the policy is not satisfactory, the AI agent can be fine-tuned on the DT of the RS. Here, DT is to be understood as an instance of the DT Prototype, that is linked to one physical twin (PT) and therefore contains all details about it (Glaessgen & Stargel, 2012). In (Nichol, Pfau, Hesse, Klimov, & Schulman, 2018) the authors mention a trend in RL to "train on the test set". They introduce a metalearning dataset, consisting of many similar tasks sampled from a single task distribution, to construct a suitable benchmark. This dataset can be used to evaluate few-shot RL algorithms.

By not having realistic variations of the digital environment, it is unlikely for the AI agent to perform well on the RS. Like detailed above, three kinds of variations were considered when training the AI agent.

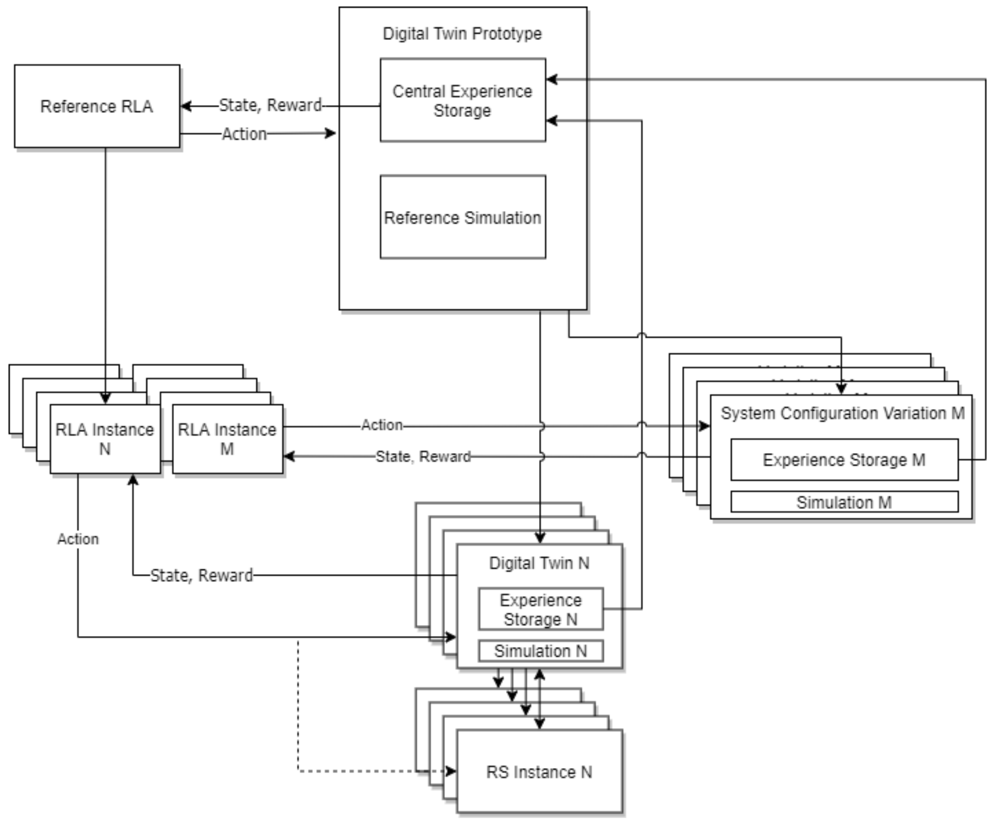


Figure 8 This figure introduces an interpretation, in which System Configuration Variations are seen as instances of a DT Prototype (further details about the differences to the DTs in the text). The boxes indicate the different entities, and the arrows indicate either dataflow, an instantiation from a generic entity, or (in the case of a double arrow) a twinning process. The dotted arrow indicates that the action can in principle be applied to the RS instance directly, but this is not the preferred option, at least not during training. The two stacks of RL Agent (RLA) instances are in theory the same, but hint at the difference between the DTs and the System Configuration Variations.

In the architecture described in Figure 7, a reference RL Agent is trained on a centralized experience storage that was fed by all the instances of the DT Prototype. Those have in turn been used to train the instances of the RL Agents. This option requires the experiences to be stored locally, which might not always be desired.

Constructing the policy of the Reference AI agent can be seen as an application of federated learning (FL), which "is an approach to machine learning, in which the training data is not managed centrally." (Ludwig & Baracaldo, 2022) While the biggest motivation to apply federated learning might be in applications in which data privacy and data ownership are paramount, there are also more practical reasons. Especially with big datasets in settings where the bandwidth is not great, it can be very practical not having to send the data to a central storage. In FL the parties performing the local training are called *clients*, and the instance that orchestrates the training is called *aggregator*. In our case the Generic AI agent can serve as aggregator and the instances are the clients. Each client is trained on an environment (which is the instantiated and twinned DT), which contains all the manufacturing anomalies of the RS (Glaessgen & Stargel, 2012). The aggregator receives the trained models from the clients and performs *model fusion.* In the case artificial neural nets, this could be realized in averaging the weights. This merged model will then be deployed either to all the existing instances, where the next iteration of this process can start, or to a completely new instance, where it serves as the starting point for the local instance. If the performance is not satisfactory, the fine tuning with the instantiated DT happens. Since

Version	Status	Date	Page
Version 2.3	Public	2024.05.31	28/52

the instance of the DT shall be connected to, and twinned with the RS, this translates into a good performance on the RS. Note, that there may be multiple aggregators, each commanding a party of clients. This is relevant for our domain variations, where the structure of the models may vary significantly.

4 Mapping UC1 (STEM)

The first ASIMOV use case (UC1) concerns transmission electron microscopy. To achieve state-of-the-art resolution, Transmission Electron Microscopes (TEM) require a tuning procedure, often operated by an expert user. Such a procedure is crucial in bringing TEM to its lowest aberration state. Aberrations cause deviations in electron trajectories, or, equivalently, in the electron wavefront, thus deteriorating the imaging resolution. Our aim in the ASIMOV project is to automate the aberration correction process using a new method which takes advantage of an AI agent, trained on the data simulated by a DT of a TEM.

To estimate and correct aberrations different output images can be used. Here we use so-called Ronchigram images, of an amorphous sample in Scanning Transmission Electron Microscope (STEM) mode. We initially aim at controlling 1st order aberrations, namely, 2-fold astigmatism, and defocus and assume that all higher-order aberrations are controlled to be near-zero by methods already available in the TEM software.

A Ronchigram is a convergent-beam electron diffraction pattern (CBED) that carries information both about the properties of the specimen and the properties of an electron beam, such as its aberrations. Depending on the amount and type of aberrations present in the electron beam, different patterns emerge in the Ronchigram image. For instance, if 2-fold astigmatism is present, the Ronchigram of an amorphous sample shows stretched patterns in the presence of defocus (see Figure 9).

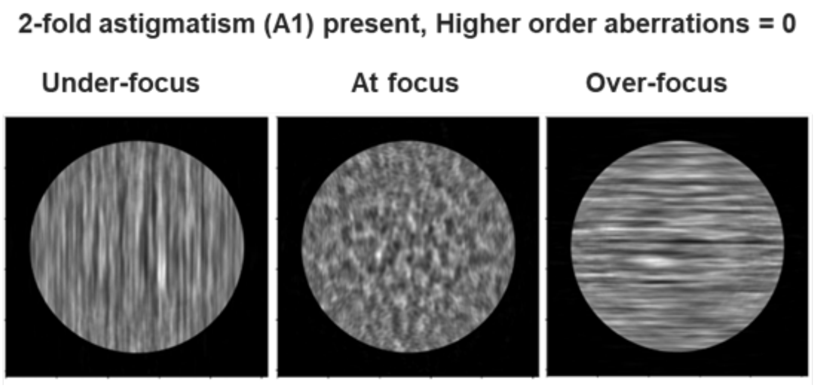

2-fold astigmatism (A1) present, Higher order aberrations = 0
Under-focus
At focus
Over-focus

Figure 9 Response of Ronchigram image to 1st order aberrations, defocus and 2-fold Astigmatism: In presence of 2-fold astigmatism, Ronchigram pattern rotates by 90 degrees while changing focus settings.

For the objective of ASIMOV, the AI agent should learn how to move the state of the TEM system, via control knobs for defocus and astigmatism, towards a state with zero aberrations (zero defocus and 2-fold astigmatism in this case). Table 3 shows how the concepts explained in this document map to the TEM use case (UC1). The remainder of this section is dedicated to pre-processing, post-processing, and variation methods used in the UC1.

Table 4 Mapping of concepts used in this document to the TEM use case (UC1).

Concept	Definition in UC1
Observation	Ronchigram images.
State	Aberration values accessible via Ronchigram images.
Internal state	Full state of the electron microscopy system, including, but not limited to, aberrations of the electron beam (due to imperfections in optical components, such as lenses), electron beam energy, the detection device, and the specimen. The system is only partially observable.
Reward	A function representing the lower or zero aberration values.
Action	The signal that the AI agent generates from which the aberration controller values (input) is inferred.
Input	The value of the controller knobs for defocus and 2-fold astigmatism.

4.1 Pre-processing

The AI agent employed in UC1 utilizes step size and direction to define actions for modifying aberration values, specifically defocus and 2-fold astigmatism in two directions (real and imaginary components of astigmatism). In a TEM system, practical changes in aberration are achieved through lower-level control mechanisms, specifically adjustments in the current of the magnetic lens coils. Consequently, the actions determined by the AI agent must be translated into the parameters of the lower-level control. This translation relies on methods established during the TEM setup. However, an alternative approach exists. We can utilize relative changes in aberrations as an indication for relative changes in controller knobs, which, in this case, correspond to the currents in the magnetic coils. As long as the AI agent successfully recognizes the concept of "better" or "worse" based on the microscope images, the controller knobs can be adjusted until the microscope reaches an optimal state with minimal aberrations. This assumes a linear relationship between lens currents and aberration values, which is generally not accurate. Although, the linear approximation holds when the change in current is small, it can be influenced by various factors, such as coil geometry and magnetic material properties. Furthermore, this assumption is valid only when the lens has not reached the saturation of its magnetic materials. In practice, even strict linearity is not necessary. This is because the AI-based correction is an iterative process, and as long as the concept of "better" or "worse" remains valid, the system can be guided towards the optimal state.

Additionally, the TEM controller knobs impose limitations on the range of input values, to ensure the operator remains in the reasonable working regime of the TEM. Therefore, constraint handling is also relevant for UC1. Currently, the AI agent is aware of the boundaries of the possible actions via the environment. The environment is limited to aberration values that are within the boundaries of allowed values for the controller knobs.

4.2 Post-processing

4.2.1 Information preparation

Currently, we apply the following transformations to prepare the images for the RL algorithm:

1. Cropping the image, so that it only includes the diffraction pattern within the disk (the so-called bright field disk)
2. Normalizing the image
3. Applying a window function (e.g., Kaiser window)
4. Applying the 2d Fourier transform

The aforementioned steps yield an image that highlights the presence of aberrations in a more obvious way, particularly to the human eye. While it is possible to begin the RL algorithm with the Ronchigram image itself as the input, based on our experience, we have observed that incorporating these post-processing steps enhances the efficiency of AI training and the generalization of the trained model to real data.

In the post-processed image in the case of zero aberration, the Ronchigram image corresponds to a small circle. As defocus increases, the disk radius increases, and with large 2-fold astigmatism an ellipse-like shape appears. Figure 10 shows two examples of Ronchigram images and their transformed form after the 4 steps described above.

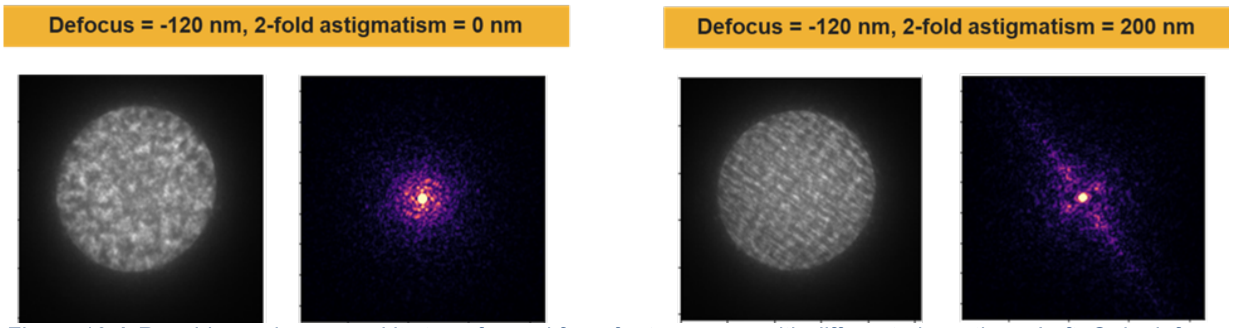

Defocus = -120 nm, 2-fold astigmatism = 0 nm
Defocus = -120 nm, 2-fold astigmatism = 200 nm

Figure 10 A Ronchigram image and its transformed form for two cases with different aberrations. Left: Only defocus is present, right: Both defocus and 2-fold astigmatism are present.

4.2.2 Reward function formulation

In UC1, the reward should represent the level of aberration, which should be as small as possible, ideally zero. Aberrations, i.e., the deviation of the electron wavefront from an ideal spherical shape, are caused by imperfections in an optical system. As mentioned before, the image we use for estimating the aberration level is the Ronchigram image of an amorphous specimen and from there we further define a reward function.

Using the domain knowledge, a reward function, representing the level of aberration, was carefully formulated. This was done, firstly, by feature reduction on the post-processed image (the Fourier transform image) and, secondly, by defining reward as a function of the reduced features. The process is as follows. The prepared (post-processed) image is smoothed with a Gaussian kernel. Subsequently, the shape appearing in the smoothed image is quantified. This is done by interpreting the spectra as an unknown realization of a 2-dimensional bivariate Gaussian distribution. We then determine the eigenvalues of its covariance matrix. For an ellipse this gives 2 non-equal values. A scaled ratio of these numbers was used as the reward function in the proof-of-concept case. In this proof of concept, the reward is now a direct function of the state, next to being near convex and an example of a so-called shaping reward. Finally, it is not sparse, which can make learning more efficient.

The above engineered reward function has clear shortcomings. For instance, it does not provide a convex map through changes of aberrations in presence of noise in simulated data. In addition, it is not easily generalizable to other aberrations which generate more complicated shapes. Therefore, a new method for defining a state & reward was developed. The goal is to achieve a reward that is more general in terms of extendibility to other types of aberrations and needs less human involvement in defining the reward function. In this method, the state is ascertained using a neural network (currently a resnet18 encoder (He, Zhang, Ren, & Sun, 2016)) which uses the entire prepared image, rather than the image with reduced features, as the input. The AI agent, in short, needs to take a decision based on an image alone, outputting new knob values every turn. The reward can be sparse, simply zero everywhere except for the goal state, or inverse shaping based on the distance the knob settings are from an ideal value.

4.2.3 State shaping

In general, the full internal state of the physical TEM system is unknown. The aberration level of the system is inferred via the observation, the Ronchigram image in our case. The full internal state of the system is only partially observable. This is also the case for the DT of the system as this is a complex system with only a few observable images and parameters. Although several parts of the physical system are abstracted by simple models in the DT, there are some parameters of the system which are hard to estimate using only one image.

A few cases for which the internal state cannot be inferred from one observation (one Ronchigram image) are as follows. One Ronchigram does not necessarily provide sufficient information to infer both first order aberration coefficients, namely, the 2-fold astigmatism and the defocus values. For instance, a large astigmatism and no defocus can result in an almost identical image as a large defocus and no astigmatism. Another case is when the accurate parameters describing the performance of the camera are not available and they cannot be estimated from one Ronchigram. In this case a pre-estimation process is most likely required. Another example is if a magnetic lens in an electron microscope shows significant hysteresis effect. This means that using the coils' current value as a control parameter will result in different magnetic fields and thus different beam quality, depending on whether the preceding change in current was in decreasing or increasing direction. The hysteresis effect is not currently covered in the DT, and due to its complexity, it should be investigated whether this effect is significant enough to merit a further modelling step.

To restore the Markov property for the system one approach is to use the system's history, i.e., use a sequence of images over time, or use a different parameter estimation process before running the DT simulations. This can provide the Markov property, which is important for the AI agent. This means that an AI agent bases its decision solely on the state that the system is currently in.

4.2.4 Offline vs. online RL

In the current setting the DT generates an image dataset with sampling over the control parameters and small variations over the rest of parameters, accompanied by a metadata file describing the parameter values for each image. The AI agent then starts the learning and interacts with the offline dataset, rather than with the DT itself. This is known as offline RL. This has been convenient for proof-of-concept. However, this approach has limitations. For instance, the AI agent is limited by the sampling resolution in the dataset. Moreover, it will become increasingly difficult to generate an offline dataset once the number of control parameters increases. Therefore, the preferred method for the future is for the AI agent to directly interact with the DT.

4.3 Variations

In addition to parameters controlled by the AI agent there are other parameters in the DT that are varied to provide a more realistic scenario for training the RL algorithm. Below is a list of variations considered in the system according to three types described in Section 3.3.

Disturbance Variation: Effects that cannot be influenced by the configuration or control of the microscope. For example, when controlling first order aberrations (defocus and 2-fold astigmatism) it is possible that higher order aberrations still have small, nonzero, values. Therefore, small random values of higher order aberrations are added to produce variations in the DT outputs. Moreover, sources of noise, such as camera noise, are added to the DT to create variations.

System Configuration Variation: Another source of variations can be the parameters, which are unknown in the physical measurements. For instance, when the thickness of the sample is unknown, the DT can produce outputs with several thickness values so that the AI agent can learn to become robust to it.

Domain Variation: An example of domain variations is in the STEM use case where the AI agent could be applied on a different microscope type with different inputs or outputs, e.g., a STEM image rather than a diffraction pattern (Ronchigram). However, this type of variation so far has not been used in the RL training.

5 Mapping UC2 (UUV)

The following chapter will explain how the concepts of the previous chapters can be mapped to the Unmanned Utility Vehicle (UUV) use case.

5.1 Pre-processing

Before the pre-processing pipeline can begin, the actions provided by the AI agent must be converted into a JSON file. This file contains information in the form of data structures that describe the variation of the environment, such as the density of trees in certain areas and the position of vehicles on the roadside. The JSON file is then used to create the varied Unreal environment, which can then be loaded together with the corresponding OpenDRIVE and OpenSCENARIO files to run an instance of the simulation. Figure 11 shows a high-level view of the pre-processing pipeline from the JSON file to the Unreal Engine. The JSON file is passed to a variant process that evaluates the variation information contained in the JSON file and creates a Triangraphics (TG) project file. The TG project file contains all the variants specified in the JSON file. Since the TG project file cannot be processed directly by Unreal, the next process generates data that can be imported by Unreal. After the import, the data is then available for simulation.

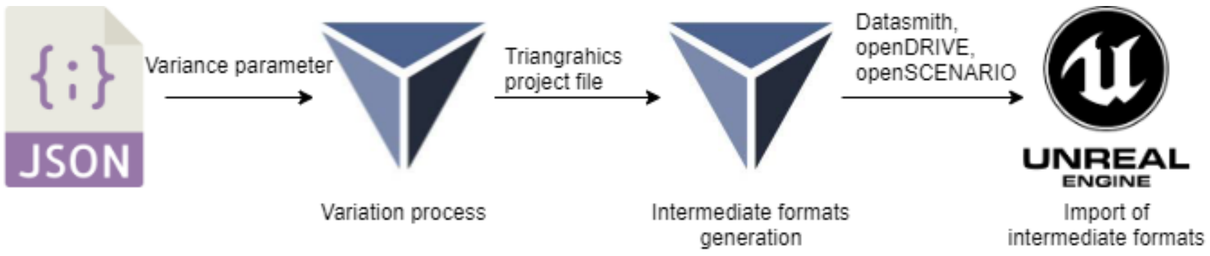

JSON
Variance parameter
Triangrahics project file
Datasmith, openDRIVE, openSCENARIO
UNREAL ENGINE
Variation process
Intermediate formats generation
Import of intermediate formats

Figure 11 High-level view of the pre-processing pipeline.

This section focuses on pre-processing, discussing the construction of the pipeline, which plays a central role in the creation of the 3D environment. An essential aspect of the pipeline is the interpretation and application of variation information provided by the RLA. They form the basis for the generation of the 3D environment, which is in the end executed in the simulation environment CARLA. In addition, the use of different complex data types, each of which is used to control different variations, is discussed. Furthermore, the choice of file formats is discussed, which are important in the pipeline for the processing of data. Finally, the significance of material descriptions, which are important for a realistic representation of 3D environments, is discussed.

5.1.1 Detailed view of the pre-processing pipeline

In this section, we discuss the individual steps of the pre-processing pipeline as well as the considerations behind the chosen structure. Before the pipeline begins its work, the actions generated by the AI agent must be converted into a JSON file containing the necessary variation data. Among other aspects, this file determines the distribution of trees in certain regions and the placement of vehicles along the roads. To rule out inconsistencies such as overlapping vehicles, additional validation is required to ensure the correctness of the data and to guarantee effective learning of the RLA. A 3D environment for CARLA is then generated based on the JSON file. This is also combined with OpenDRIVE and OpenSCENARIO files to be able to start an executable simulation. A schematic representation of the entire pre-processing pipeline, from the initial JSON file to the final CARLA 3D simulation, is presented in Figure 12.

When designing a pre-processing pipeline, different challenges must be addressed. In simulation, it is common to differentiate between the visual representation of the 3D environment and the road network on which the cars drive or the scenarios run. This distinction results in different files, each of which meets specific requirements.
The visual implementation of a 3D environment is a complex task, as it requires many parameters and involves a significant amount of work. For example, creating a polygon grid representing a road requires detailed consideration of various elements such as the number and width of lanes, kerb dimensions, and specific road features such as markings and pavements. This complexity also extends to intersections

and junctions. Furthermore, the creation of a comprehensive 3D environment requires the generation of other elements such as buildings, forests, fences, and green spaces, which together form a realistic and complete 3D environment.

In addition to the visual representation, a detailed road network is essential, which includes the real connections of the visible roads. This includes not only turning possibilities and lane transitions at intersections, but also the integration of road signs that potentially influence driving behaviour, such as speed limits. These aspects together contribute to the creation of a comprehensive and functional transport infrastructure.

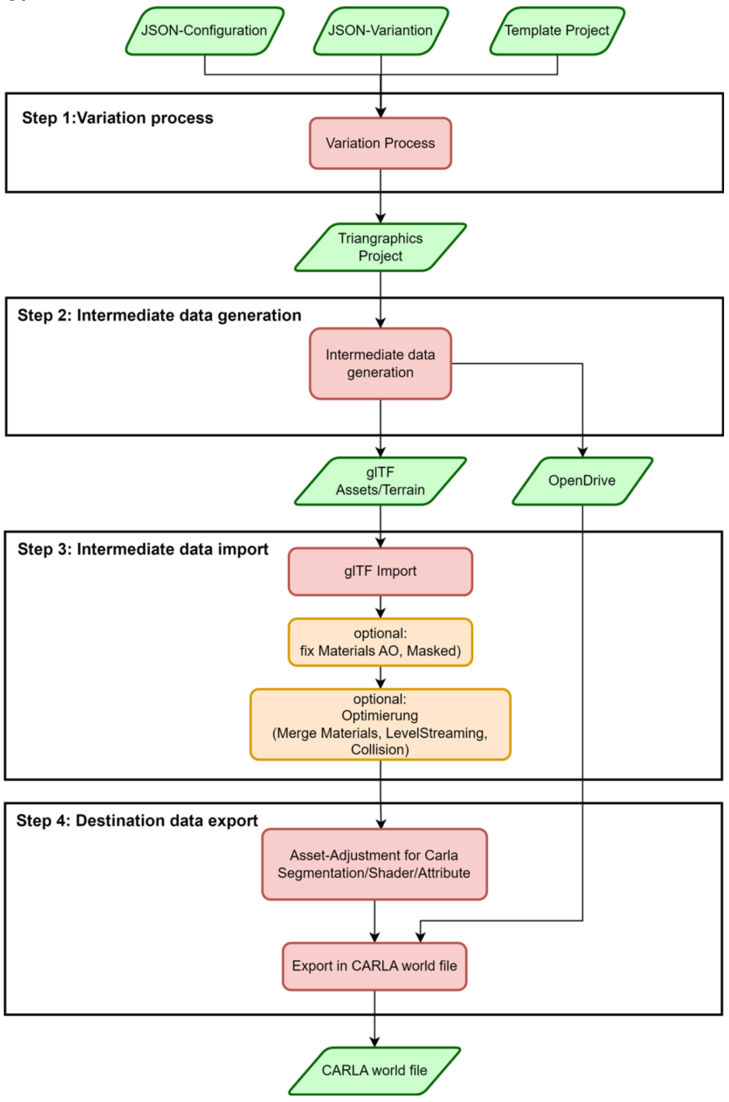

JSON-Configuration
JSON-Variantion
Template Project
Step 1:Variation process
Variation Process
Triangraphics Project
Step 2: Intermediate data generation
Intermediate data generation
glTF Assets/Terrain
OpenDrive
Step 3: Intermediate data import
glTF Import
optional: fix Materials AO, Masked)
optional: Optimierung (Merge Materials, LevelStreaming, Collision)
Step 4: Destination data export
Asset-Adjustment for Carla Segmentation/Shader/Attribute
Export in CARLA world file
CARLA world file

Figure 12 Detailed view of the pre-processing pipeline.

The following is a detailed look at the four steps of the pre-processing pipeline, focusing on the data types and file formats used.

Step 1
In step 1 of the variation pipeline, the focus is on the creation of an enriched Triangraphics (TG) project generated by a variation process. Three files are provided to the variation process: a configuration file, a variation file and a TG template project file. The configuration file contains information that remains unchanged during the entire learning process and across all pipeline runs. It can be used to control the level of detail of objects in the 3D environment.

The variation file created by the RLA contains information about which modifications are intended for a pipeline run. A variation file describes a specific 3D environment using complex data structures. A 3D environment can be reproduced from a variation JSON file at any time. This saves considerable storage space, since the JSON file is much smaller than the generated 3D environment for CARLA.
The TG template project file serves as the basis for the resulting project file. It defines unchangeable objects such as houses or the road network and determines the intermediate format that is generated in the second stage.

Step 2
In step 2, the TG project file created in the previous step is used to generate intermediate data. This step is necessary because the Unreal Editor, which is used in the following step, cannot import the TG files directly. A glTF file is generated for the visual part of the 3D environment and an OpenDrive for the road network.

Step 3
Step 3 involves importing the intermediate data resulting from the previous step into the Unreal Editor, a process automated by so-called blueprint script files. In this step, not only the integration of the data takes place, but also the possibility of making optional refinements. These adjustments are used to improve the visual quality as well as the performance of the 3D environment. These can be measures such as merging materials, or specific configurations that make level streaming more efficient and adjust collision settings.

Step 4
Step 4 uses blueprints to export the data imported and processed in the previous step. The export results in a CARLA world file. This special file, which is used for integration into the CARLA simulation environment, summarises all relevant information required for the visualisation of the 3D environment.

For a more in-depth discussion of the specific steps and methods, please refer to document D1.3.

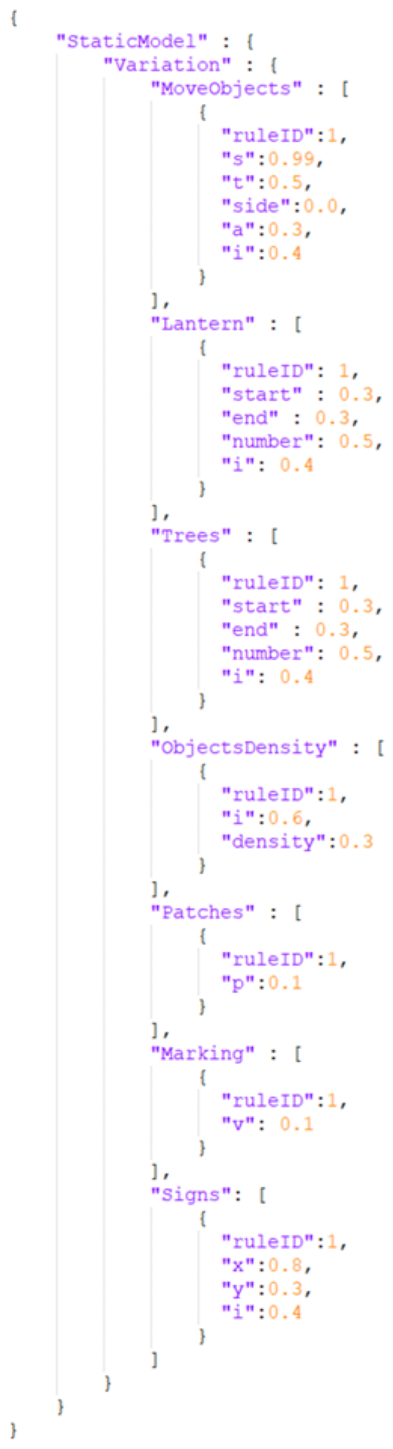
5.1.2 Data types

The pipeline is controlled by two types of JSON files. The first is a configuration file that defines the settings for the entire learning process. In addition, the RLA provides a JSON file that defines specific variations for each individual run of the pipeline. The content of such a file can be seen in Figure 13.
This file contains complex data structures that allow modifications of individual variation classes of the 3D environment. Complex data types exist within the JSON file to be able to vary the following variation classes:

- Objects on the road (e.g. cars)
- Trees in a row of trees
- Trees in an area
- Road signs
- Road patches (e.g. manhole covers, tyre tracks)
- Markings

For each of these classes, a separate data structure was designed, each containing parameters with simple data types which can be used to control the variation. Essentially, a distinction is made between discrete and continuous data types. An exact structure of the data types can be found in D1.3.

The specification of the data types and value ranges is a challenge, as they must be designed in such a way that the RLA has minimal knowledge of the 3D environment, but at the same time allows for a wide range of variations. This is achieved through value range mapping: continuous values in the range 0 to 1 are mapped to specific values in the 3D environment for the RLA. For example, such a value could represent a position on a road where a vehicle should be parked without the RLA knowing the exact position.



```
{
    "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 13 Content of an example JSON variation file.

5.1.3 File formats

Different file formats are used in the pre-processing pipeline, each of which is selected according to its specific application. This section provides an overview of the most important formats.

5.1.3.1 JSON

For the control of the variations and the configuration of the pipeline, JSON (JavaScript Object Notation) was chosen because it is easy to generate and read. This benefits the RLA in particular, which generates the variation files. In addition, JSON can be read and processed easily, which is performed in the first step of the pre-processing pipeline. Its wide availability in numerous programming languages simplifies integration. Compared to XML, JSON is characterized by a more compact structure that favours efficient and fast data exchange. Therefore, it has become one of the standard formats for data exchange and application configuration.

5.1.3.2 Triangraphics project

The TG project was developed by the company Triangraphcs. It consists of a collection of files that contain a wide range of information, including attributes, elevation maps and vectors.
A TG project includes at least one vector file, which can contain numerous vectors. These vectors can have various modifiers assigned to them that determine how they should be interpreted. For example, a vector can be interpreted as a road or a river depending on the modifier. In addition, it can be defined in the project which target formats should be generated. In the context of the UUV use case, such formats are OpenDRIVE and glTF.

5.1.3.3 glTF

The Graphics Language Transmission Format (glTF) [https://www.khronos.org/gltf/], developed and managed by the Khronos Consortium, is specialised to store, and transmit 3D data efficiently. It optimises data size and layout, which reduces processing time and improves performance for real-time applications. As a universal format, glTF supports the exchange between different systems and offers a wide range of features, including meshes, textures and materials, complemented by PBR for realistic renderings. Thanks to these features, glTF has wide industry acceptance and is used in a wide range of applications and game engines. In the context of ASIMOV, glTF serves as an intermediate format containing the 3D environment and necessary material descriptions and was chosen for its compactness and fast and efficient processing.

5.1.3.4 OpenDRIVE

OpenDRIVE is a file format developed by the ASAM (Association for Standardization of Automation and Measuring Systems, [https://www.asam.net/]) organization that enables the realistic simulation of virtual road networks. It includes aspects such as lanes, intersections, and pedestrian crossings. This format is essential for the automotive industry as it allows vehicles to be tested in simulated environments without real safety risks. Within ASIMOV, ego vehicles navigate on a road network provided by the OpenDRIVE file.

5.1.3.5 OpenSCENARIO

OpenSCENARIO, developed and maintained by the ASAM organisation, is an established standard in the automotive industry. It is used to describe driving scenarios and traffic situations that can then be used, for example, to test driver assistance systems in a simulation. In contrast to OpenDRIVE, which provides an accurate model of road networks, OpenSCENARIO focuses on the dynamics and interaction of road participants within these road networks. As part of the ASIMOV project, the traffic scenarios are defined using OpenSCENARIO files, which are then simulated in a varied 3D environment.

5.1.3.6 CARLA world file

CARLA (CAR Learning to Act) is an advanced simulation environment designed specifically for the development and testing of autonomous vehicles. As an open-source project, CARLA is based on the powerful Unreal Engine, which enables the use of realistically designed 3D environments in simulations. Users have the choice of testing their vehicles under a wide range of conditions, such as changing weather conditions or dynamic traffic scenarios. CARLA is characterized by several additional features that are particularly useful in the context of vehicle simulation, including highly developed sensor models such as lidar or radar. To visualize a 3D environment in CARLA, a CARLA World file is required. This contains all the assets required to describe the scenery in detail. Within the ASIMOV project, CARLA serves as a simulation platform for the Unmanned Utility Vehicle use case.

5.1.4 Material description

An essential part of the pre-processing pipeline is a realistic representation of the 3D environment. The materials play an important role in this. Among other aspects, they determine how objects absorb, reflect, and refract light rays, which has a great influence on realism. The change in lighting conditions, for example due to the change in time of day or weather conditions, can significantly influence scenarios. Different methods for material description can be used, including Physical Based Rendering (PBR), which is particularly common in video games. OpenMaterial [https://github.com/LudwigFriedmann/OpenMATERIAL] is an alternative method that is becoming increasingly important in traffic simulation. The two material descriptions are explained in more detail in the following sections.

5.1.4.1 PBR

PBR is a method of computer graphics that emulates materials and light sources in such a way that they appear physically correct or at least believable. The primary goal of PBR is to create realistic graphics that appear consistent under different lighting conditions and produce plausible results. PBR is built into many modern graphics engines and applications by default, from games to professional 3D software. For example, PBR is supported in unreal, which in turn is used by the simulation environment CARLA. PBR is also supported by a variety of 3D formats, such as glTF. PBR materials are often based on several textures, such as albedo, metallicity, roughness and normals, to name a few. Combined, these textures represent the behavior of a specific material.

5.1.4.2 OpenMATERIAL

For a long time, physical accuracy was not a priority in the visual representation of geometries, mainly because of limited memory and computing capacities. Therefore, resource-saving methods were preferably used. With advanced hardware, the focus is now shifting to approaches that require more resources but achieve physically correct results. One example of such a method is OpenMATERIAL [https://github.com/LudwigFriedmann/OpenMATERIAL]. The main goal of OpenMATERIAL is to develop a universal, standardized 3D model exchange format that integrates a physically correct representation of materials for rendering and sensor simulation (see [https://github.com/LudwigFriedmann/OpenMATERIAL]). OpenMATERIAL is mainly used in simulations where physical correctness is essential, especially in sensor simulations.

5.1.4.3 Comparison

PBR is an established material description that can be used to simulate realistic material behaviour. This is achieved by emulating physical laws and properties. Consistent and realistic visual results can be achieved.

OpenMATERIAL, on the other hand, is not as well established as PBR. However, it is much more precise in terms of physical accuracy. Inconsistencies between real and simulated behaviour should be minimised or eliminated. However, the accuracy comes at a price and it is likely that this will lead to slower execution efficiency.

In summary, PBR offers advantages in terms of acceptance, especially for applications that require high speed with high visual quality and realism. OpenMATERIAL is to be established as a standard in the future under ASAM. This should increase future acceptance. If high realism and consistency is required, OpenMATERIAL is the better choice. For ASIMOV, PBR was chosen because both execution efficiency and accuracy are important, as can be read in D2.1.

5.2 Post-processing

Post-processing takes all the measurement data collected during the simulation runs and calculates relevant states and rewards out of that. In the UUV use case, this will contain two aspects, representing the goals of the optimization process. On one hand, an anomaly detection, based on an autoencoder neural network, will be used to measure the information gain of every simulated scenario. On the other hand, criticality metrics will be used.

An Autoencoder is a neural network, which has the same number of neurons in the input and output layer. Its main purpose is to reconstruct the input data in the output layer, with as little error as possible. The easiest way to do this would be to learn the identity function, meaning that every input is passed through the network without any modifications to it. To suppress this behavior, the hidden layers in-between input and output layer, have less neurons. This ensures compression of the input data and therefore an automatic extraction of relevant features of the input vector. This also means that the autoencoder can cope with data of similar type better than when confronted with completely new data. This makes it applicable for use as anomaly detection, by measuring how good the autoencoder can reconstruct the data, it is confronted with.

The Anomaly Detection will be using the entire set of measurement data, gathered during one simulation run, and will calculate the reconstruction error of every signal by providing it as input to the autoencoder network. The overall reconstruction error will be used to measure the anomaly value and therefore the information density of the dataset. A high reconstruction error represents a high information value. This will be part of the reward function. After calculating the information value, the neural network will be retrained, also incorporating the just seen dataset as additional training data. That way, when confronted with similar data, a low reconstruction error and therefore low information value will be determined by the anomaly detection. The individual contributions of every signal to the overall anomaly score will represent part of the state.

The criticality metrics will be scenario specific and will be evaluated for every simulated scenario as well. It is planned to use an ensemble of different key performance indicators (KPIs) to measure the criticality of scenarios based on different aspects. The selection process for the KPIs can be found in (Westhofen, et al., 2022).

In Figure 14, the entire post-processing workflow for reward calculation can be seen.

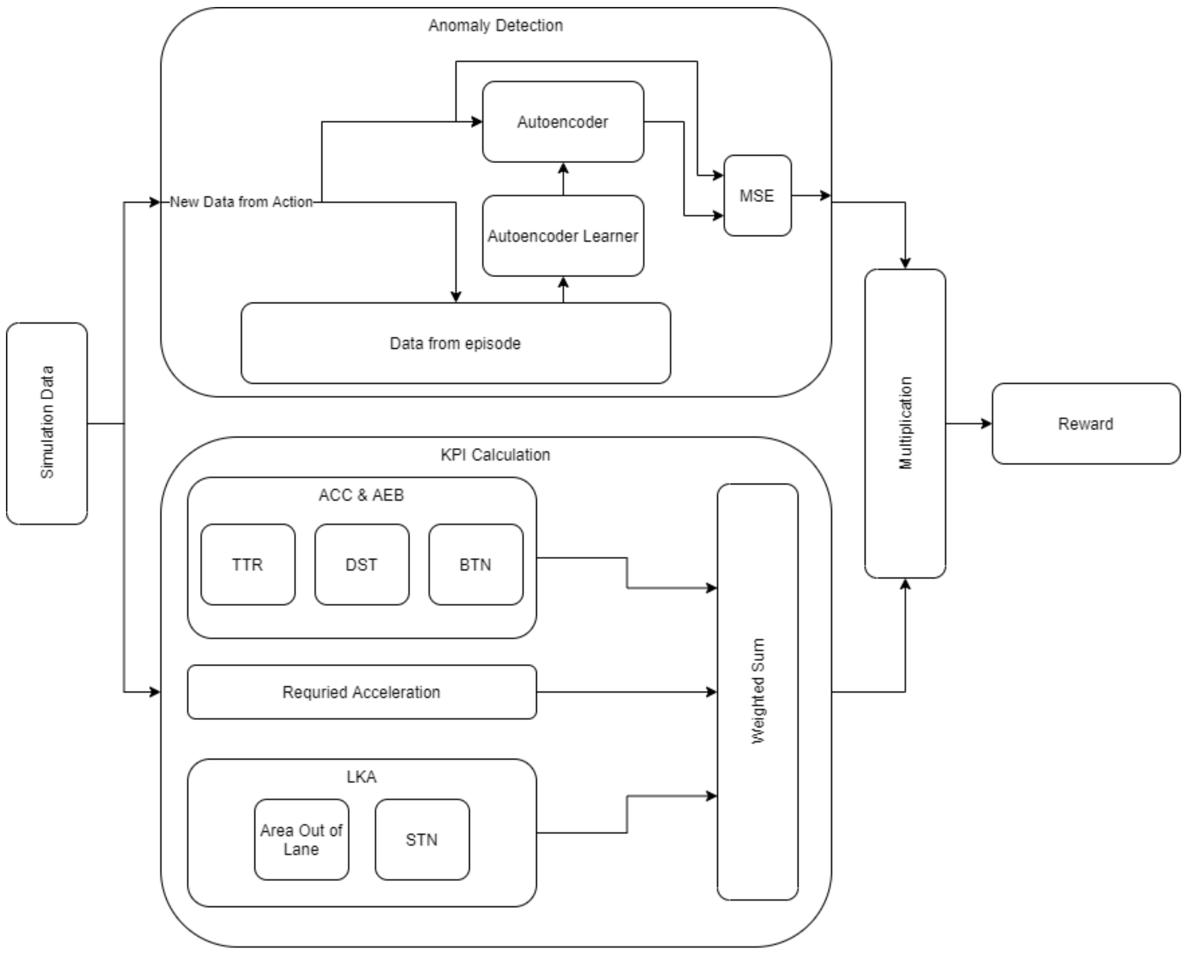


Figure 14 Reward shaping for UC2.

First tests have shown that the anomaly detection part of the post-processing would require a very large state vector to be able to save the history of already evaluated scenarios. As this is not feasible due to an already large state space, the Markov property would be violated without including this information. As this can lead to severe problems in learning with the RLA, the anomaly detection has been taken out of the RL loop for now and will only be used as a standalone tool to analyze data, but not directly use it as part of the reward.

As we have seen earlier, state shaping is a crucial part in RL. The state is all the information the Agent has in a given time step. We repeat that a process having the Markov property means that the new state Xt+1 only depends on the current state Xt and the action, not on the states before.
Considering the Scenario Generation Process in the UUV use-case this could become an issue, as we will explain in the following.

- The Agent proposes a new scenario for which a certain criticality will emerge when it is simulated.
- This criticality is multiplied with the novelty of the scenario to get the reward.
- The way that novelty is implemented is, that it uses information from previous scenarios (actions). Special caution is needed that the Markov property is not violated at this point.
- An easy solution is to add a history of the process to the state. That means return the previously suggested scenarios additionally to the state. This however bloats the state to an infeasible big size.
- As an alternative the novelty detection won't be part of the reward and state but be used in external posterior analysis. This may lead to the undesired output of similar critical scenarios that needs to be addressed in another way. Note, that these considerations stay abstract at the current stage of the use-case, since the reward is solely calculated using the criticality.

5.3 Variations

Variations are needed to make the AI agent more robust during training. Slight variations in the vehicle's response to actions lead to the AI agent proposing more general actions, which avoids overfitting to the DT. This allows for an easier transfer of the learned actions to the RS, which slightly differs from the DT. For the variation to work as intended and to yield an advantage, it must be ensured that the actual physical twin lies in-between the boundaries of the variation. The variation can therefore include experts' knowledge about typical parameter ranges of certain vehicle types.

Variation will happen in between different episodes of learning as part of resetting the environment from the AI agent's perspective. During an episode, i.e., a sequence of actions that include requests for test cases, the vehicle will not be varied, as it must be ensured, that each varied vehicle offers a consistent response across its test sequence.

Variation itself can happen either by randomly changing certain parameters of the vehicle inside some limiting boundaries, or by systematically changing parameters to ensure an evenly covered space of vehicle properties.

6 Lessons learned

Based on the described methods in the state of the art in chapter 3, combined with the use case mapping to these methods in chapter 4 and 5, an overview can be created of the most important lessons learned. In this chapter, the most important lessons learned are written down, divided into pre-processing, post-processing, and variations. The chapter concludes with abstracts of two papers related to this deliverable. These papers are published as part of the ASIMOV project.

6.1 Pre-processing

Pre-Processing can be used to manipulate the output of the RL agent to fit the purpose of the actual application. This can happen e.g. in the form of limits that shall not be exceeded to ensure system safety, or limits of parameters where the digital twin models are valid. It is however beneficial to limit the outputs not by filtering the output of the RL agent, but instead by systematically limiting what the agent itself can output as actions. This way it can be ensured that the agent can expect its actions to be transmitted correctly to the system and different actions do not end up as similar inputs after filtering. Safety limits can still be held by the systematic output limitation approach inside the RL agent. A safety net in form of hard limits after the RL agent is still a good idea in addition, but should be part of the physical system itself, instead of an add-on component.

Another way preprocessing can be used effectively is by converting actions into usable inputs for the CPS by changing datatypes.

Encapsulating the components of the pre-processing pipeline in Docker containers has proven to be particularly helpful, when extensive pr-processing is needed. This approach offers considerable advantages, especially for larger teams working on different areas, as it effectively avoids conflicts with regard to dependencies. Additionally, it eliminates the need for an in-depth understanding of parts that you are not working on. If such a complex pre-processing pipeline is used, it has to be ensured that it can be executed with sufficient speed, as it is directly integrated into the RL loop and is executed in every RL step.

6.2 Post-processing

Neural networks have the capability to identify a model that accurately represents the training dataset. However, ensuring that the learned model encompasses the relevant aspects of the experiment, and consequently enables accurate predictions on the real (inference) data is a significant challenge. By post-processing the training data, generated by the DT in this case, we ensure that the problem is well defined. Furthermore, reducing the dimensionality of the problem becomes helpful, and in some cases, essential, to focus solely on the relevant aspects of the experiment. Post-processing has a major effect on how a DT can be used for AI training. Naturally, an equivalent post-processing is also applied on the outputs of the real system during the inference. During the ASIMOV-project, some valuable lessons were learned related to the post-processing. This section summarizes the two most important ones. The first one is related to the data dimensionality, while the second one is related to the Markov property.

Data dimensionality

Data is being post-processed because the original raw data generally contains unnecessary information, that does not benefit the actual goal that shall be achieved by analyzing this data. During the training the neural networks can easily over-train, learning irrelevant aspects of the data, which may harm the generalization of the models on real data. Therefore, one way that post-processing can be useful is as a dimensionality reduction tool. While the DT itself is a tool to generically model a system's behavior, postprocessing is created with the application of the system in mind.

This observation becomes evident when considering the two primary use cases of ASIMOV. In the STEM use case, it was demonstrated that utilizing the amplitude of the Fourier transform (FT) of the original image (Ronchigram image) leads to improved generalization of the reinforcement learning (RL) approach to real data. The rationale behind this lies in the physics of Ronchigram image formation, where the amplitude of the FT encapsulates the most relevant information regarding the electron beam while disregarding irrelevant sample-related details. Figure 15 provides an example highlighting the disparities between simulated and real data. In this case, nine Ronchigram images and their corresponding FT amplitude images were used for different aberration values. The image in the middle represents the best state and the goal of the experiment is to guide the system to arrive to this state. The noticeable disparity between the simulated and real Ronchigram images can primarily be attributed to the partially unknown structure of the actual sample. Incorporating this complex real sample structure into the simulated models is not feasible or practical. However, the changes in the aberrations are more obvious in the FT images compared to the original Ronchigram images, particularly to the human eye. This phenomenon also helps the generalization of our neural network models to real data. It is important to note that a structural solution for improving generalization to real data involves reducing the gap between simulated and real data. Additional methods to achieve this are enhancing the digital twin models or employing machine learning-based methods designed to address this gap. By implementing such solutions, the significance of post-processing steps may diminish. However, in the initial stages, leveraging domain knowledge through post-processing steps has enabled the proof of concept and has set the stage for further advancements.

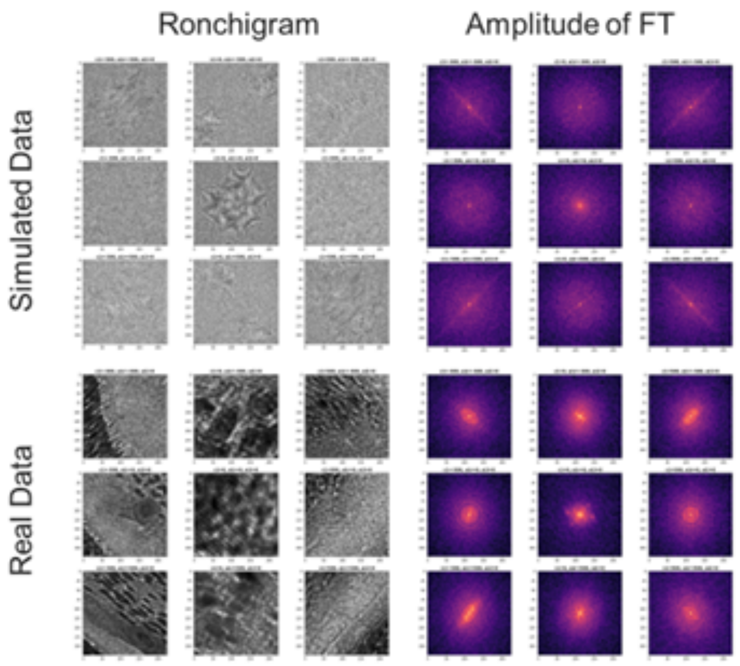


Figure 15 The Ronchigram and its Fourier transform's amplitude for nine values of aberration combinations for simulated and real data. The aberrations that are varied are defocus (along the rows) and 2-fold astigmatism (along the columns.

The UUV use case also uses this approach. The DT, which can be seen as a combination of virtual environment and vehicle, is producing data in the form of a time series, which describes the movement of all traffic participants inside the scenario. This data is then further post-processed to be compressed to the respective KPIs that quantify the criticality of the entire scenario. The agent itself therefore also does not see raw data, but rather some very compressed information.

These observed forms of post-processing are obviously beneficial for creating data that leads to more robust generalized RLA training, because it is less affected by raw data noise and inaccuracies. But it, of course, also limits the capabilities of the RLA, as it eliminates raw data details. The selection of the most suitable method depends on the specific conditions of the application.

Knowledge of the postprocessing method that will be applied together with the RLA, can therefore play a substantial role in answering the question of how accurate a DT needs to be. The accuracy of the raw data of the DT is unimportant, as long as the post-processed data, that is directly relevant for RL training cannot be distinguished from the PT's postprocessed data.

A very simple example with extreme detail reduction in postprocessing can make that clearer. Let's assume a very simple real system that multiplies all its inputs by 2.2.

$$y_{PT} = 2.2 \cdot x$$

The system is now being modeled by a DT, which approximates the behavior with a multiplication of the inputs by 2:

$$y_{DT} = 2 \cdot x$$

The raw data results are obviously different, so one could argue that the DT is not accurate enough. But this decision is purely dependent on the combination of DT and Postprocessing. Let's assume the actual raw value of the data is not important at all, but after postprocessing it is only important, if the result is positive or negative:

$$f_{post-processed} = \text{sign}(y).$$

In this case, the postprocessed data is identical for DT and PT, which makes it perfectly reasonable to use the DT instead of PT for RLA training, without any negative effect on training results. The DT could be even simpler in this case, by eliminating the multiplication as a whole:

$$y_{DT-simplified} = x.$$

In this case, we can even state

$$f_{post-processed}(y_{PT}) = f_{post-processed}(y_{DT}) = f_{post-processed}(y_{DT-simplified}).$$

Although it is a very extreme example, it clearly shows that postprocessing is an essential step in making a DT compatible with RLA training and that the accuracy of the DT itself cannot be evaluated in isolation, but only together with the postprocessing.

Markov property

When applying postprocessing to compress (state) data, one needs to be careful not to violate the Markov property by losing critical information. If this happens, the RLA's perception of the environment is not entirely determined by its current state and action anymore and therefore the Markov property is violated.

The Markov property is also a crucial step in enabling the STEM use case. In STEM given only one Ronchigram image, it is not possible to deduce a singular set of aberration coefficients. However, by following a sequence of measured images while changing one or multiple aberration coefficients, a unique solution can be inferred. To address this issue, various approaches are employed.

- The problem was divided into two tasks. In the first task, a supervised learning network was utilized to predict a distance-to-goal metric. Subsequently, an optimization algorithm, such as Bayesian optimization, was employed to direct the measurements towards the minimum of the distance-to-goal metric.
- To train the neural network of the reinforcement learning (RL) system, we utilized a stack of images with different focus values as input. While this method proved effective, it can be sensitive to the defocus steps in the input data.
- We employed RL algorithms that explicitly incorporate sequence information.

Each of the methods possesses its own advantages and disadvantages, but they all aim at addressing the problem of the non-uniqueness of the solutions.

The importance of the Markov property also becomes clear using the same toy example as in the previous lesson learned in the post-processing section. Let's assume the postprocessing outputs only the absolute value of the actual output data. I.e. 10 and –10 will both be post-processed to 10. Assume that the agent has the task of getting close to the number 0 with possible actions being [-1, 1]. The starting state being 10, the agent will take the action 1. As information about the absolute value of the state is lost during post-processing, the state 10 could actually be 10 or –10 before post-processing. The Markov property would be lost in this case. But if the agent can take history into account, it can make up for that lost information. In this example, the agents next state, after action 1 would be either 11 or –9 resulting in a post-processed 11 or 9, which are different from one another. Combined with the information about the action that has been taken, the agent can reconstruct the lost information and therefore also restore the Markov property.

6.3 Variations

We found that the concept of variations is connected to post-processing in the following way. Imagine a universal function approximator, like a neural net being confronted with high fidelity outputs from a lot of different variations. Due to the diversity in this training set, it will learn to work reasonably well for a multitude of inputs. This requires “ignoring” specifics about the current variation but only focusing on robust features. This can also be called generalization. This is analogous to an explicit feature engineering, where high dimensional data are compressed to relevant features. This concept is also explained in Section 3.2.4.

The specific selection of the amount on handcrafted post-processing and neural-network-based learning can be influenced by the required explainability, generalization and available data. With enough representative data, the learned features are expected to provide better generalization, mostly at the cost of reduced explainability.

In the STEM use case, variations in the training data are introduced by adding disturbances and structural changes in the DT parameters. However, to use a reasonable range of parameters, it is sometimes necessary to tune the DT parameters. For this process we have used the measurements from the real microscope and subsequently, used an optimization algorithm, such as, Bayesian optimization, to search for the parameter of interest in DT. Ultimately, this process could be automated for more convenience.

6.4 Publications related to D2.2

During the project, two papers were published related to T2.2. Both of them are related to post-processing. Short abstracts of these two publications are given in this section.

6.4.1 Estimation of Dynamic Gaussian Processes

J. Van Hulst, R. Van Zuijlen, D. Antunes and W. P. M. H. M. Heemels, "Estimation of Dynamic Gaussian Processes," 2023 62nd IEEE Conference on Decision and Control (CDC), Singapore, Singapore, 2023, pp. 3206-3211, doi: 10.1109/CDC49753.2023.10383256.

Abstract: Gaussian processes provide a compact representation for modeling and estimating an unknown function, that can be updated as new measurements of the function are obtained. This paper extends this powerful framework to the case where the unknown function dynamically changes over time. Specifically, we assume that the function evolves according to an integro-difference equation and that the measurements are obtained locally in a spatial sense. In this setting, we will provide the expressions for the conditional mean and covariance of the process given the measurements, which results in a generalized estimation framework, for which we coined the term Dynamic Gaussian Process (DGP) estimation. This new framework generalizes both Gaussian process regression and Kalman filtering. For a broad class of kernels, described by a set of basis functions, fast implementations are provided. We illustrate the results on a numerical example, demonstrating that the method can accurately estimate an evolving continuous function, even in the presence of noisy measurements and disturbances. (van Hulst, van Zuijlen, Antunes, & Heemels, 2023)

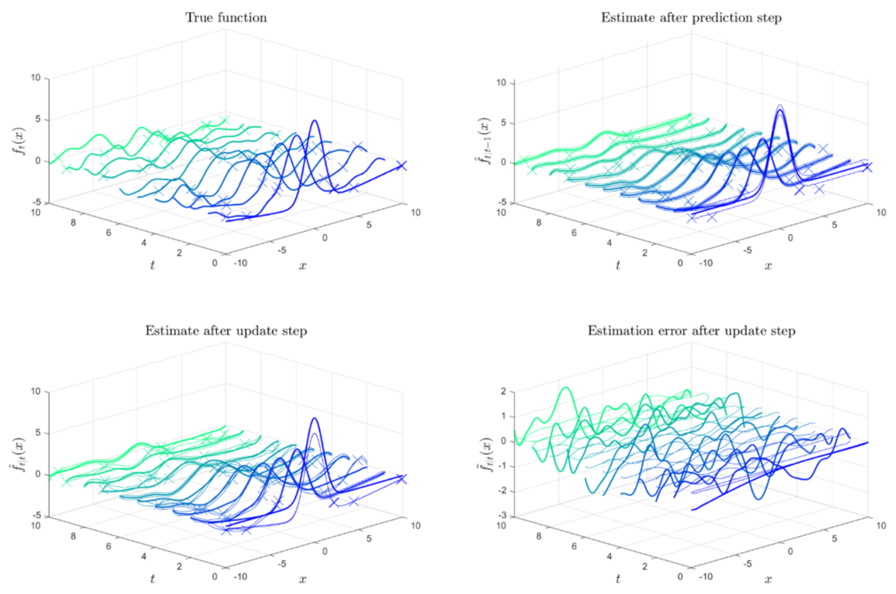


Figure 16 Estimation of dynamic Gaussian processes.

6.4.2 Bayesian Estimation for Linear Systems with Nonlinear Output and General Noise Distribution using Fourier Basis Functions

van Zuijlen, R. A. C., van Hulst, J., Heemels, W. P. M. H., & Guerreiro Tomé Antunes, D. J. (Accepted/In press). Bayesian Estimation for Linear Systems with Nonlinear Output and General Noise Distribution using Fourier Basis Functions. In 2023 IEEE 62nd Conference on Decision and Control, CDC 2023 (pp. 2166-2171). Institute of Electrical and Electronics Engineers.

Abstract: Bayesian estimation can be used to estimate the state of dynamical systems, but its applicability is hampered due to the curse of dimensionality. The published paper aims to mitigate this bottleneck for a relevant class of systems consisting of a linear plant with bounded input, driven by stochastic disturbances with non-linear noisy output; the distributions of the disturbances and noise have a bounded support but are otherwise general. Using a frequency-domain interpretation of the operations of the Bayes' filter, it is show that, under mild assumptions, exact Bayesian estimation can be pursued in a countable space of Fourier series coefficients, rather than in the usual functional space of probability densities. This fact leads to a natural approximate method, where the Fourier series coefficients corresponding to high frequencies are discarded. For this approximate method, the complexity of the conditioned state distribution, measured by the number of Fourier coefficients, remains constant at prediction steps and grows only linearly at each update step. The applicability of the results is illustrated in the context of electron microscopy, where a residual error analysis indicates that the approximation is accurate. In Figure 17, the evolution of the probability density function is shown, where update and prediction steps are alternating. In Figure 18, the proposed method with Fourier Basis Functions (FBF) is compared with the Unscented Kalman Filter (UKF), and faster convergence is shown to the true state. (van Zuijlen, van Hulst, Heemels, & Guerreiro Tomé Antunes, 2023)

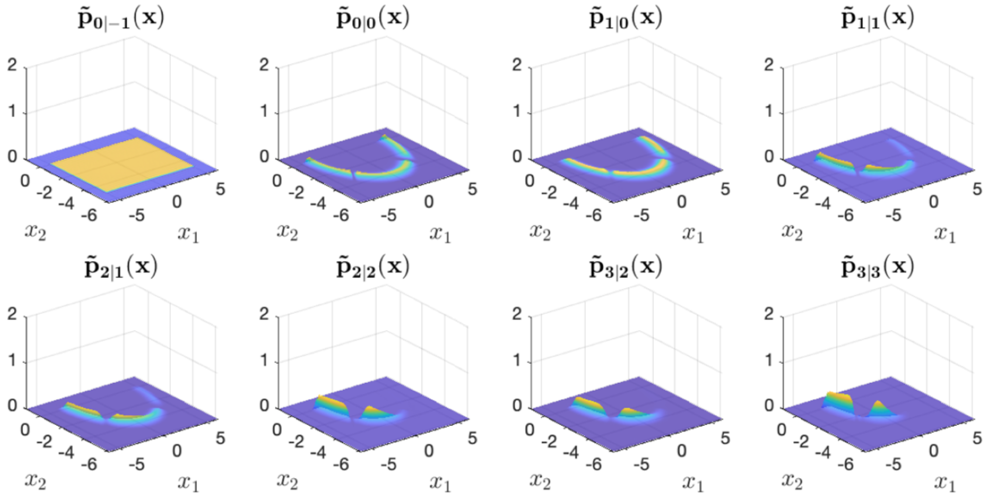


Figure 17 Development of probability density function.

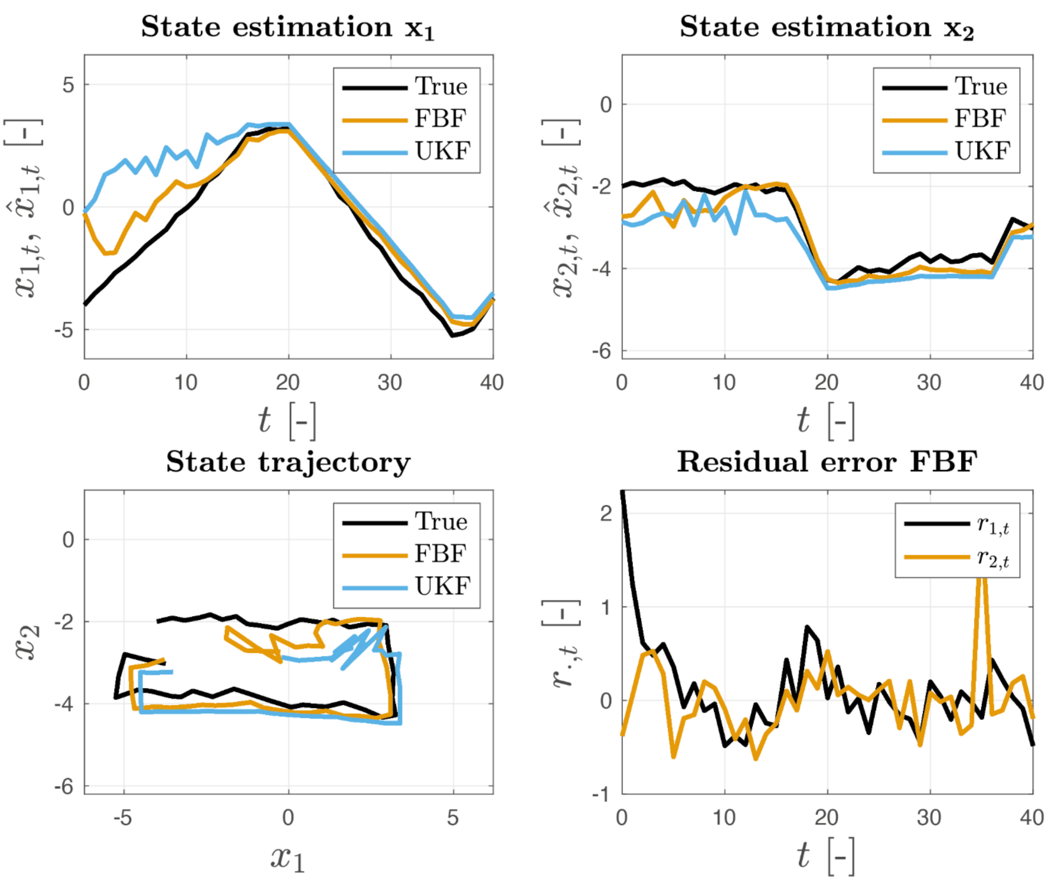


Figure 18 Comparison with unscented Kalman filter.

7 Conclusions

This document puts forward the proposed interface between the DT and the AI agent to be used in the ASIMOV project. The proposed interface relies on three main ingredients: pre-processing, post-processing, and variations. In the context of pre-processing, existing solutions include input shaping through discretization or, more generally, input parameterization, encoding and decoding, and input interpolation. Pre-processing solutions must ensure constraint handling. Post-processing can be divided into the following categories: information preparation, state shaping, and reward function formulation. Information preparation encompasses integration, noise handling, cleaning, missing data imputation, normalization, and transformation. State shaping techniques include restoring the Markov property, extracting features from the output of the process, and building observers to estimate hidden information. Solutions for formulating reward functions include reward shaping and reward machines. Variations can be introduced in different places in the model, including disturbance variations, system configuration variations and domain variations, and can be introduced randomly, via space-filling, or via neural networks. For each of the two main use cases of the project, the Electron microscope and the Unmanned Utility Vehicle pre-processing, post-processing and variations were instantiated, and use-case-specific training features were discussed. The last chapter of the document was dedicated to the most important lessons learned during the project regarding pre- and postprocessing, and variations.

8 Bibliography

An, J., & Cho, S. (2015). Variational autoencoder based anomaly detection using reconstruction probability. *Special Lecture on IE, 2*(1), 1-18.

ASIMOV-consortium. (2020). *ASIMOV - Full Project Proposal.*

Aström, K. J., Hagander, P., & Sternby, J. (1984). Zeros of sampled systems. *Automatica, 20*(1), 31-38.

Bai, E. W., & Dasgupta, S. (1990). A note on generalized hold functions. *Systems & control letters, 14*(4), 361-368.

Berberich, J., Köhler, J., Müller, M. A., & Allgöwer, F. (2020). Data-driven model predictive control with stability and robustness guarantees. *IEEE Transactions on Automatic Control, 66*(4), 1702-1717.

Berberich, J., Köhler, J., Müller, M. A., & Allgöwer, F. (2021). Data-driven model predictive control: closed-loop guarantees and experimental results. *at-Automatisierungstechnik, 69*(7), 608-618.

Bertsekas, D. (2012). *Dynamic programming and optimal control: Volume I.* Athena scientific.

Busoniu, L., de Bruin, T., Tolic, D., Kober, J., & Palunko, I. (2018). Reinforcement learning for control: Performance, stability, and deep approximators. *Annual Reviews in Control, 46*, 8-28.

Camacho, A., Icarte, R. T., Klassen, T. Q., Valenzano, R. A., & McIlraith, S. A. (2019). LTL and Beyond: Formal Languages for Reward Function Specification in Reinforcement Learning. *IJCAI*, (pp. 6065-6073).

Camacho, E. F., & Alba, C. B. (2013). *Model predictive control.* Springer.

Chandak, Y., Theocharous, G., Kostas, J., Jordan, S., & Thomas, P. (2019). Learning action representations for reinforcement learning. *International conference on machine learning*, (pp. 941-950).

Chou, C. C., Bruell, S. C., Jones, D. W., & Zhang, W. (756-761). A Generalized Hold Model. *Proceedings of the 25th conference on Winter simulation.*

Cobbe, K., Klimov, O., Hesse, C., Kim, T., & Schulman, J. (1282-1289). Quantifying generalization in reinforcement learning. *International Conference on Machine Learning.*

Creswell, A., White, T., Dumoulin, V., Arulkumaran, K., Sengupta, B., & Bharath, A. A. (2018). Generative adversarial networks: An overview. *IEEE signal processing magazine, 35*(1), 53-65.

De Moor, B. J., Gijsbrechts, J., & Boute, R. N. (2022). Reward shaping to improve the performance of deep reinforcement learning in perishable inventory management. *European Journal of Operational Research, 301*(2), 535-545.

Degrave, J., Felici, F., Buchli, J., Neunert, M., Tracey, B., & Carpanese et al, F. (2022). Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature, 606*(7897), 414-419.

Doersch, C. (2016). Tutorial on variational autoencoders. *arXiv preprint.*

Efron, B., & Hastie, T. (2021). *Computer Age Statistical Inference, Student Edition: Algorithms, Evidence, and Data Science.* Cambridge University Press.

Eriksson, L., Johansson, E., Kettaneh-Wold, N., Wikström, C., & Wold, S. (2000). *Design of experiments.* Stockholm: Principles and Applications, Learn ways AB.

Eschmann, J. (2021). Reward function design in reinforcement learning. In *Reinforcement Learning Algorithms: Analysis and Applications* (pp. 25-33).

Famili, A., Shen, W. M., Weber, R., & Simoudis, E. (1997). Data preprocessing and intelligent data analysis. *Intelligent data analysis, 1*(1), 3-23.

Farebrother, J., Machado, M. C., & Bowling, M. (2018). Generalization and regularization in DQN. *arXiv preprint.*

Garcia, J., & Fernández, F. (2015). A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research, 16*(1), 1437-1480.

García, S., Ramírez-Gallego, S., Luengo, J., Benítez, J. M., & Herrera, F. (2016). Big data preprocessing: methods and prospects. *Big Data Analytics, 1*(1), 1-22.

Ghavamzadeh, M., Mannor, S., Pineau, J., & Tamar, A. (2015). Bayesian reinforcement learning: A survey. *Foundations and Trends in Machine Learning, 8*(5-6), 359-483.

Glaessgen, E., & Stargel, D. (2012). The digital twin paradigm for future NASA and US Air Force vehicles. *53rd AIAA/ASME/ASCE/AHS/ASC structures, structural dynamics and materials conference 20th AIAA/ASME/AHS adaptive structures conference 14th AIAA.*

Glattfelder, A. H., & Schaufelberger, W. (2003). *Control systems with input and output constraints.* London: Springer.

Grüne, L., & Pannek, J. (2017). *Nonlinear model predictive control.* Springer.

Gramacy, R. B. (2020). *Surrogates: Gaussian process modeling, design, and optimization for the applied sciences.* Chapman and Hall/CRC.

Harutyunyan, A., Devlin, S., Vrancx, P., & Nowé, A. (2015). Expressing arbitrary reward functions as potential-based advice. *Proceedings of the AAAI Conference on Artificial Intelligence*, *29.*

Hastie, T., Tibshirani, R., & Friedman, J. H. (2009). *The elements of statistical learning: data mining, inference, and prediction.* New York: Springer.

Hausknecht, M., & Stone, P. (2015). Deep Recurrent Q-Learning for Partially Observable MDPs. *2015 AAAI Fall Symposium Series* (pp. 29-37). AAAI.

Hausknecht, M., & Stone, P. (2015). Deep reinforcement learning in parameterized action space. *arXiv preprint.*

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 770-778). IEEE.

Hewing, L., Wabersich, K. P., Menner, M., & Zeilinger, M. N. (2020). Learning-based model predictive control: Towards safe learning in control. *Annual Review of Control, Robotics, and Autonomous Systems, 3*, 269-296.

Huh, S., & Yang, I. (2020). Safe reinforcement learning for probabilistic reachability and safety specifications: A Lyapunov-based approach. *arXiv preprint.*

Icarte, R. T., Klassen, T. Q., Valenzano, R., & McIlraith, S. A. (2022). Reward machines: Exploiting reward function structure in reinforcement learning. *Journal of Artificial Intelligence Research, 73*, 173-208.

Icarte, R. T., Klassen, T., Valenzano, R., & McIlraith, S. (2018). Using reward machines for high-level task specification and decomposition in reinforcement learning. *International Conference on Machine Learning*, (pp. 2107-2116).

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of artificial intelligence, 4*, 237-285.

Khairy, S., & Balaprakash, P. (2022). Multifidelity reinforcement learning with control variates. *arXiv preprint.*

Khalil, H. K. (2015). *Nonlinear control.* New York: Pearson.

Kim, H., Yamada, M., Miyoshi, K., Iwata, T., & Yamakawa, H. (2020). Reinforcement Learning in Latent Action Sequence Space. *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, (pp. 5497-5503).

Knuth, D. E., & Moore, R. W. (1975). An analysis of alpha-beta pruning. *Artificial intelligence, 6*(4), 293-326.

Kouvaritakis, B., & Cannon, M. (2016). *Model predictive control.* Switzerland: Springer International Publishing.

Ludwig, H., & Baracaldo, N. (2022). *Federated Learning.* Springer.

Luenberger, D. G. (1964). Observing the state of a linear system. *IEEE transactions on military electronics, 8*(2), 74-80.

Mansingh, G., Osei-Bryson, K. M., Rao, L., & McNaughton, M. (2016). Data preparation: Art or science? *2016 International Conference on Data Science and Engineering (ICDSE)*, (pp. 1-6).

Masson, W., Ranchod, P., & Konidaris, G. (2016). Reinforcement learning with parameterized actions. *13th AAAI Conference on Artificial Intelligence.*

Moos, J., Hansel, K., Abdulsamad, H., Stark, S., Clever, D., & Peters, J. (2022). Robust Reinforcement Learning: A Review of Foundations and Recent Advances. *Machine Learning and Knowledge Extraction, 4*(1), 276-315.

Narvekar, S., Peng, B., Leonetti, M., Sinapov, J., Taylor, M. E., & Stone, P. (2020). Curriculum learning for reinforcement learning domains: A framework and survey. *arXiv preprint.*

Ng, A. Y., Harada, D., & Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. *Icml, 99*, 278-287.

Nichol, A., Pfau, V., Hesse, C., Klimov, O., & Schulman, J. (2018). Gotta learn fast: A new benchmark for generalization in RL. *arXiv preprint.*

Qin, Y., Zhang, W., Shi, J., & Liu, J. (2018). Improve PID controller through reinforcement learning. *2018 IEEE CSAA Guidance, Navigation and Control Conference (CGNCC)*, (pp. 1-6).

Saleem, A., Asif, K. H., Ali, A., Awan, S. M., & Alghamdi, M. A. (2014). Preprocessing methods of data mining. *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*, (pp. 451-456).

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., & Hassabis, D. (2017). Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint.*

Singh, S., Lewis, R. L., Barto, A. G., & Sorg, J. (2010). Intrinsically motivated reinforcement learning: An evolutionary perspective. *IEEE Transactions on Autonomous Mental Development, 2*(2), 70-82.

Soviany, P., Ionescu, R. T., Rota, P., & Sebe, N. (2022). Curriculum learning: A survey. *International Journal of Computer Vision*, 1-40.

StataCorp. (2007). *Stata multivariate statistics: reference manual.* Stata Press Publication.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction.* MIT press.

Tang, Y., & Agrawal, S. (2020). Discretizing continuous action space for on-policy optimization. *Proceedings of the AAAI conference on artificial intelligence*, *34*, pp. 2591-5988.

Turk, M., Pentland, A., Belhumeur, P., & Hespanha, J. (1991). Eigenfaces for recognition. *Journal of Cognitive Neuroscience, 3*(1), 71-86.

van Hulst, J., van Zuijlen, R., Antunes, D., & Heemels, M. (2023). Estimation of Dynamic Gaussian Processes. *62nd IEEE Conference on Decision and Control (CDC)*, (pp. 3206-3211). Singapore.

van Zuijlen, R., van Hulst, J., Heemels, W., & Guerreiro Tomé Antunes, D. (2023). Bayesian Estimation for Linear Systems with Nonlinear Output and General Noise Distribution using Fourier Basis Functions. *62nd IEEE Conference on Decision and Control (CDC)*, (pp. 2166-2171). Singapore.

Wabersich, K. P., & Zeilinger, M. N. (2021). A predictive safety filter for learning-based control of constrained nonlinear dynamical systems. *Automatica, 129*.

Westhofen, L., Neurohr, C., Koopmann, T., Butz, M., Schütt, B., Utesch, F., & Böde, E. (2022). Criticality metrics for automated driving: A review and suitability analysis of the state of the art. *Archives of Computational Methods in Engineering*, 1-35.

Wiewiora, E., Cottrell, G. W., & Elkan, C. (2003). Principled methods for advising reinforcement learning agents. *Proceedings of the 20th international conference on machine learning*, (pp. 792-799).

Yamaguchi, A., Takamatsu, J., & Ogasawara, T. (2009). Constructing continuous action space from basis functions for fast and stable reinforcement learning. *RO-MAN 2009 - The 18th IEEE International Symposium on Robot and Human Interactive Communication*, (pp. 401-407).

Yu, D., Ma, H., Li, S., & Chen, J. (2022). Reachability Constrained Reinforcement Learning. *International Conference on Machine Learning*, (pp. 25636-25655).

Zhang, C., Vinyals, O., Munos, R., & Bengio, S. (2018). A study on overfitting in deep reinforcement learning. *arXiv preprint.*