Architecture and technical approach for DT-supported AI-based training and system optimization

[WP3; T3.2/T3.3; Deliverable: D3.2/D3.3; Version 2.0]

Non-Confidential

AI training using Simulated Instruments for Machine Optimization and Verification

Document Information

Project	ASIMOV
Grant Agreement No.	20216 ASIMOV - ITEA
Deliverable No.	D3.2/D3.3
Deliverable No. in WP	WP3; T3.2/T3.3
Deliverable Title	Architecture and technical approach for DT-based AI-training: state of the art
Dissemination Level	public
Document Version	2.0
Date	2024.04.11
Contact	Jilles van Hulst
Organization	TU/e
E-Mail	j.s.v.hulst@tue.nl

Commented [K(1]: 2024

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

Task Team (Contributors to this deliverable)

Commented [H(2]: needs to be updated

Name	Partner	E-Mail
Duarte Guerreiro Tomé Antunes	TU/e	d.antunes@tue.nl
Ilona Armengol	TNO	ilona.armengolthijs@tno.nl
Jan Willem Bikker	CQM	janwillem.bikker@cqm.nl
Niklas Braun	AVL	niklas.braun@avl.com
Maurits Diephuis	TFS	maurits.diephuis@thermofisher.com
Richard Doornbos	TNO	richard.doornbos@tno.nl
Lukas Schmidt	Norcom	lukas.schmidt@norcom.de
Jilles van Hulst	TU/e	j.s.v.hulst@tue.nl
Mehrnoush Hajnorouzi	DLR	mehrnoush.hajnorouzi@dlr.de
Elias Modrakowski	DLR	elias.modrakowski@dlr.de
Bram van der Sanden	TNO	bram.vandersanden@tno.nl
Richard Doornbos	TNO	richard.doornbos@tno.nl
Tabea Henning	DLR	tabea.henning@dlr.de
Jan van Doremalen	CQM	Jan.vandoremalen@cqm.nl

Formal Reviewers

Version	Date	Reviewer
1.3	2024.05.15	Thomas Kotschenreuther (RA Consulting); Holger Kohr (Thermo Fisher Scientific)

Change History

Version	Date	Reason for Change
0.1	2022.01.28	Initial version.
0.2	2022.02.02	Updated by Task 3.2 members. Ready for internal review.
0.3	2022.02.16	Updated after internal review by Task 3.2 members. Ready for 2nd team review.
0.4	2022.02.28	Updated after 2nd internal review by Task 3.2 members. Ready for formal review.
1.0	2022.02.28	Updated after formal review. Ready for publication.
1.1	2023.03.02	Merged with D3.3

1.2	2023.11.15	Updated by Task 3.2 members. Ready for internal review.
1.3	2024.04.11	Finalized by Task 3.2 members. Ready for internal review.
2.0	2024.05.17	Final version.

Abstract

WP3 is concerned with the development of a technical approach and a reference architecture for DT-supported AI-based system optimisation. System optimisation can be performed by connecting AI to both the physical system and its DT. By allowing the AI to take control over the DT, a learning cycle based on reward and punishment can be constructed to validate its actions. In order to establish a baseline for improvements, the state-of-the-art for existing solutions in AI-based system optimisation is reviewed. A framework for a cost-benefit analysis to compare methods will be constructed. The outcome of this task, combined with the state-of-the-art and prior requirements, is to produce a technical approach and a reference architecture for supporting DT-supported AI-based system optimisation suitable for use in industry use cases.

This report describes the state-of-the-art in reinforcement learning and digital twin-based learning, their application in the ASIMOV use cases, as well as a reference architecture to construct and build DT-supported AI. A low-threshold introduction to reinforcement learning and Q-learning is followed by an extensive and well-structured literature overview. The initial ideas about which techniques and approaches to use and how to apply them in the use cases are then described. Afterwards, the report deals with practical challenges by concentrating on the application to specific use cases. It puts the findings in the perspective of the developed reference architecture, and it summarizes the challenges and way forward towards the end goals: apply it to the real system, make it extendable and scalable, provide the details of the technical approach, reference architecture and tools and technology that may support the building of a practical application.

Chapter 4 concentrates on two use cases for UUV. The first use case addresses creating optimal test plans for testing vehicles on a test bed: improve data quality by adapting the test plan in a way that every tested scenario contains as much valuable new information as possible. The second use case addresses sensor optimization: how to tune sensor and perception parameters in a way that the vehicle can perceive its environment in the most accurate way.

Chapter 5 concentrates on the TEM use case. The use case addresses part of the parameter settings of the microscope that influence the quality of the final image by reducing the aberrations caused by electron beam deviations: astigmatism, spherical aberration, coma, etc. The research starts from the rich literature on DRL and the findings in D3.2 and moves to development of a prototype AI agent that may be linked to an actual TEM system. Several techniques have been investigated, implemented and tested. The results form the basis for (1) testing the solution in the real world of an electron microscope, (b) a step towards further improvement of the case and extension to other aspects of the microscope and (c) a more formal description of the technological approach and reference architecture.

Chapter 6 introduces an overarching view of AI implementations, referred to as the AI architecture. The chapter includes a reference architecture for AI implementations, containing generic elements, aspects and best practices. The content is aligned with the DT architecture discussed in WP2 and the full system architecture discussed in WP4.

This document ends by drawing overall conclusions on DT-supported AI-based training and system optimization.

Table of Figures

Table of Tables

1 Introduction

Performing tests and optimizations on physical systems is often time consuming and may require extensive domain knowledge. Using digital twins (DT) combined with state-of-the-art artificial intelligence (AI) systems instead is promising, as it may help to significantly reduce the costs and effort of these tasks. In the ASIMOV project [84], we design and create a DT-based AI solution suitable for different industrial use cases as a proof of concept.
A DT, as a surrogate system, can enable extensive system-level trials, model tunings, and adaptations. With a DT as basis, many repetitions and scenarios can be gone through effectively in a virtual environment. Next to that it is possible to exploit the higher speed, it ensures repeatability, and it offers a large potential for investigating the addition of noise, the impact of access to internal states, the use of abstractions instead of detailed parameters, and many other variations and rare cases. Due to these options, it is important to thoroughly understand if there are differences in training and operational phases required.
In this context, the basic formulation of the reinforcement learning (RL) as a promising AI technology, is that the model learns in an unsupervised way from rewards when taking actions in a virtual environment. The couplings between digital twins and reinforcement learning are fully justified by the intrinsic nature of both methodologies. Thus, the considered training requirements are given for reinforcement learning, but can be generalized for other methodologies also.

This document serves as a part of the Asimov documentation and is the deliverable for task 3.2 of the project. It focuses on the design and creation of a technical approach and a reference architecture for training DT-based AIs. In the next sections, we lay the foundation for the proposed reference architecture by reviewing the state-of-the-art of DT-based AI-training. First, we explore the general challenges of AI modelling based on data of DTs compared to training based on data of physical systems. Next, we provide an introduction to reinforcement learning and Q-learning. Thereafter, we will give an extensive look into state-of-the-art general training strategies and methodologies, and their applied usage combined with the DTs. In that section, the methodological perspective of reinforcement learning is emphasized. Finally, we describe our initial ideas regarding the use-cases of the project, the results of earlier tasks and the appointed requirements to the models with respect to reinforcement learning.

Erikstad [51] investigated the differences between creating a digital twin from physics-based (structural) models and from machine-learning models. He summarized his findings in a table, shown in Table 1.

Table 1 Pros and cons of creating digital twins in two distinct ways. Reproduced from [51].

	Machine learning based	Physics based
PRO	• Model derived from data only – no need for domain knowledge • Generic and flexible - handles heterogeneous data streams (also non-physics) • Model improves over time (reinforcement learning) • Good at discovering complex relations and patterns	• Models capture deep existing knowledge based on Newtonian physics • Causal relationships provide insight and understanding • Uncertainty controlled by input and modelling accuracy • Model has universal validity – predict any point covered by model
CON	• The availability of training data needed to develop model • Correlations, not causality. Blackbox, no explanations (in particular, deep learning) • Approximation methods, no exact mathematics • Predictive capabilities deteriorate quickly outside training set scope • Difficult to predict extreme/critical conditions (few observations)	• Require extensive domain (physics) knowledge • Computationally intensive, challenge in real-time • Complete assumptions about input-output must be made upfront

The table gives inspiration on the aspects to judge possible approaches. It may not be universally true for all applications as [51] has a certain application area in mind. Another axis on which to characterize approaches can be driven by observational data vs driven by domain knowledge. This partially coincides with the columns of **Error! Reference source not found.** but not exactly:

- Physics-based is an example of using domain knowledge; modeling a supply chain context is another example. Also, in some applications physics knowledge may be as simple as a mathematical equation, or otherwise be computationally un-intensive because the model is obtained without requiring significant computation.
- Data may be generated in different ways; an important distinction is between *observational* and *experimental*. The latter allows for a combined approach of using domain knowledge and then switching to a data-driven view. A computationally intensive model can be replaced by an approximating "compact model" by a one-time effort. For this, the digital twin's input space is carefully considered (vector of parameters, constraints on these) and a list of computer experiments is created (Design of Computer Experiments, DoCE). The computationally intensive digital twin is evaluated on each of these experiments; from there it is relatively straightforward to build a model that predicts the DT's outputs from inputs using statistics and/or machine learning. Papers describing this approach are [56], [57], [58], [59]. This approach has different pros and cons than the two columns in Table 1.

Commented [Dv3]: what do we mean with un-intensive?

As a general good practice, it may be beneficial to include domain knowledge in modelling / DT building whenever possible. There are several approaches for this including hybrid forms of using observational data, simulation, approximating models for which in a wider scope of application domains there are examples.

Generic best practices for establishing a good *practical* training strategy can be found in many publications. In [52] several important steps are listed:

- Establish a budget for training data. Determine type and amount of data to be used, if retraining is necessary, if labeling of data is needed, maintenance, etc. This is important for justification of the investments in the business case.
- Appropriate data. Select the right data and label it for the specific purpose.
- Ensure data quality. Accuracy and consistency of labeled, as well as timeliness, and correctness of (real-time) behavioral data.
- Be aware of and mitigate data biases. These biases come from blind spots or unconscious preferences in the project team or training data. Diversity in the team or assessment by independent external experts may counter this problem.
- When necessary, implement data security safeguards. Security and confidentiality may pose important restrictions on your system setup and training. Be aware of government regulations and company policies.
- Select appropriate technology. The tooling for capturing and managing data should fit the requirements on volume, speed and scale and flexibility of annotation.

These generic best practices also hold for the Asimov cases of digital twin-based AI training, so we should carefully address all of these topics in detail.

1.1 A brief introduction to reinforcement learning

Before diving into the topic of reinforcement learning (RL), RL is put into perspective with respect to other learning techniques [54]:

- ***Supervised Learning:*** *Supervised learning uses a set of input data with known responses to the data (output) to generate a model of the perceived reality, with the aim to then generate reasonable predictions as a response to new data*
- ***Unsupervised Learning:*** *Unsupervised learning is a process of finding hidden patterns or intrinsic structures in data, and establishing a model based on inferences drawn from datasets consisting of input data without labeled responses*
- ***Reinforcement Learning****: Reinforcement learning is a goal-oriented learning strategy. It consists of iterative cycles of observing – taking action – being rewarded or punished. Agents gradually optimize actions in an environment, to maximize the rewards*

Reinforcement learning (RL) solves a particular type of problem, where decision-making is sequential, and the final goal is typically optimizing the total (discounted) rewards over a longer period. In particular, RL problems are often modeled formally as a Markov Decision Process (MDP). An MDP comprises of a state S, an action A, a probabilistic transition diagram T and a reward R. Many practical problems can be modeled in this framework including computer games, robotics or supply chain logistics. RL aims to learn good strategies from experimental trials and relatively simple feedback from the environment. Ultimately, the learned strategy should lead to behavior that maximizes the future rewards in an environment.

What sets RL apart from other algorithm families such as optimal control or simulated annealing-like approaches is its ability to learn and make predictions without any requiring an explicit model (T) at all. To instead learn from simple feedback alone.

Commented [Dv4]: I propose to make a connection here with the Markov Decision Problem paradigm underlying the RL approach

Commented [Dv5]: this is a questionable statement

Slightly formalizing the RL concepts and terms:

- An RL agent acts in an environment. This environment must be provided and can simply be the game of chess, or an entire flight simulator.
- An agent takes sequential actions. What actions an agent may take is determined by the environment. In chess, the action space is comprised of all the legally available chess moves.
- Once an action is taken, the environment delivers a reward as feedback. Very often this reward is simple, such as the remaining distance to a target for example. For chess, it requires a more complicated function to evaluate the current state of the game.
- The principal goal of an agent is the maximize the sum of discounted future rewards. We will touch on discounting later, but for now, just imagine that short-term goals need to be balanced with an end goal. Driving from A to B (long term) whilst not hitting cyclists (short term.)
- Each action in the environment leads to a new state. In each state, the environment delivers a reward. In general, learning updates are only done at the end of an episode. For example, playing

a single game of chess is an episode. After a win or lose, the rewards are collected and used for learning.

- An episode over T steps can take on the following form: State(1)Action(1)-Reward(2)State(2)Action(2)-Reward(3)State(3)Action(3)...Reward(T)State(T)

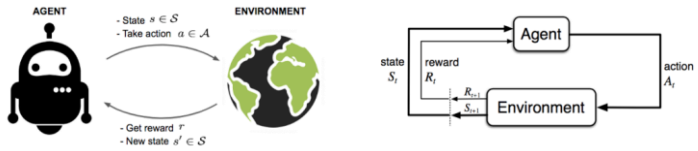


Figure 1 The basic elements of a reinforcement learning setup.

Before diving deeper, we will touch on a number of important issues that surround reinforcement learning in general. These are:

- The exploration-exploitation dilemma.
- The data inefficiency
- General performance of RL
- Reward function design
- Stability and repeatability

1.1.1 The exploration-exploitation dilemma

An agent learning in an environment has a bit of a dilemma in that sense that it must maximize rewards and learn based on what it currently knows. In chess openings for example, this may lead an agent to select a move that is not disastrous, yet far from optimal. To learn better moves, an agent needs to take a jump into the unknown from time to time, to explore if there are other moves that might be better. Exploitation is selecting the move you think currently best, exploration is trying new things. This dilemma is captured by what is known as “Bayes Bandits”.

Imagine a casino with multiple slot machines (the one-armed bandits). Each has a fixed but unknown probability of winning and an unknown reward. The reinforcement learning task is then to find the best strategy to achieve the maximum reward from these slot machines.

If the agent selects the first slot machine and only plays this machine, it will reap its rewards and learn its distribution, i.e. the probability of winning. That obviously leaves the possibility that there are many other machines in the building that have higher rewards or higher probabilities of winning.

Obviously, dropping the proverbial Monte Carlo hammer and trying all machines is not feasible. But this means an agent must learn a strategy for exploration. A relatively simple solution is to give the agent a probability parameter. In 95% of the cases, it selects the slot machine it has learned to be best, and with a small 5% change, it selects a completely random one. This strategy is known as “epsilon greedy”.

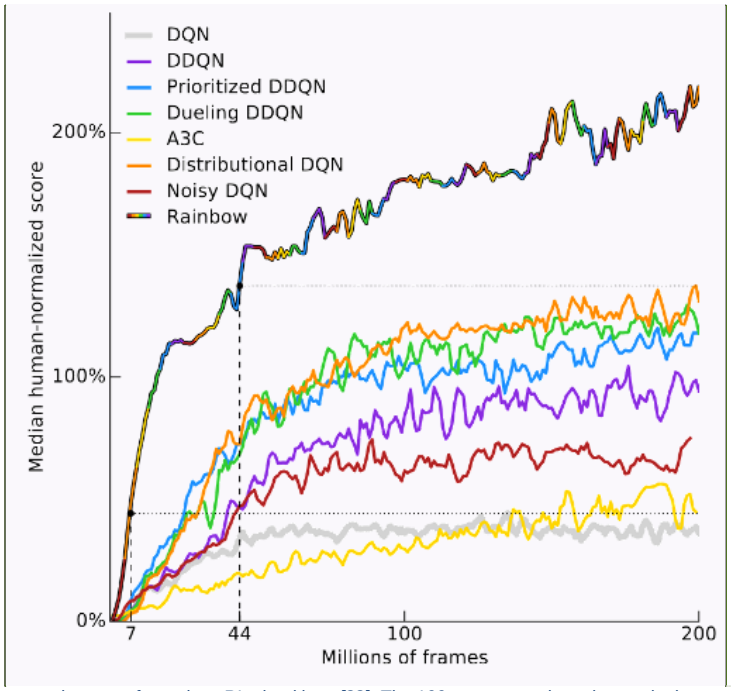


Figure 2 Data requirements for various RL algorithms [88]. The 100 percent mark on the y-axis denotes the human performance level. The x-axis shows the number of required training frames.

1.1.2 Data inefficiency

RL algorithms such as Q-learning, match and surpass human performance for Atari computer games like Pong. This is a landmark achievement for an algorithm that has no baked-in knowledge on what Pong is, or how to play it. From simple feedback it learns and learns to play it well, discovering typical human-like cheats and power-moves along the way. The downside of learning from simple feedback is the shear amount of data it requires. The current state-of-the-art algorithm requires 18 million frames to reach human performance [88] (Figure 2). That is 83 hours of equivalent gameplay. Mind you, this is a huge improvement over the first algorithm. That required 70 million frames to surpass human players..

1.1.3 Performance

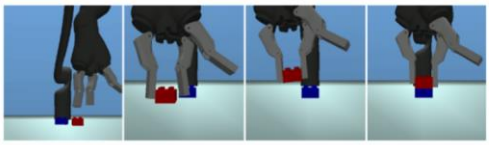
RL is a relatively new field, and this means that currently domain specific algorithms still tend to outperform RL within their domain. Atari games for example, can be solved better by more traditional Monte Carlo Tree Search algorithms. There are exceptions, most notable Deepmind's Alpha(GO)Zero and MuZero.

1.1.4 Reward function design

Designing a reward function that will instill correct or the desired behavior in an agent is non-trivial and a somewhat empirical exercise. In particular, it is hard to design reward functions such that an agent solves a problem well, and not just barely. An example of this can be seen in the 'half cheetah' environment. Here the virtual animal has to get to a tile on the right. The reward function gives increasingly larger rewards as the cheetah gets closer to this tile. This is in itself not enough for an agent to learn how to

Commented [D(6]: what is the connection to the text?
Commented [DR(7R6]: please include the source.
Commented [D(8R6]: done
Commented [K(9]: remove comma
Commented [K(10]: sheer
Commented [D(11]: picture?
Commented [D(12R11]: I agree, but you really need a movie to show that the animal is hopping on its back.

move its legs efficiently. And indeed, this environment produces disan endless number of creatures that hop upside down, crawl and do other weird and wonderful things.



$$r(b_z^{(1)}, s^P, s^{B1}, s^{B2}) = \begin{cases} 1 & \text{if } \text{stack}(b_z^{(1)}, s^P, s^{B1}, s^{B2}) \\ 0.25 & \text{if } \neg\text{stack}(b_z^{(1)}, s^P, s^{B1}, s^{B2}) \wedge \text{grasp}(b_z^{(1)}, s^P, s^{B1}, s^{B2}) \\ 0.125 & \text{if } \neg(\text{stack}(b_z^{(1)}, s^P, s^{B1}, s^{B2}) \vee \text{grasp}(b_z^{(1)}, s^P, s^{B1}, s^{B2})) \wedge \text{reach}(b_z^{(1)}, s^P, s^{B1}, s^{B2}) \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

Figure 3 Example of a reward function.

In Figure 3 an example reward function is shown. Here the agent needs to stack blocks on top of each other. Height is not sufficient as a reward function. If height is the only criteria, the agent never stacks any blocks, but simply places each of them on their side.

1.1.5 Stability

Agents specialize in solving a certain task in a particular environment. This means that overfitting is not really an issue. The problem that does arise is that trained agents are specialized, and their learning doesn't transfer to other environments. This goes so far that identical agents may learn completely different strategies in identical environments if started from different random seeds.

Unlike computer vision tasks, the bottleneck in RL is not feature representation. Lessons learned from super-vised learning in those domains don't apply to RL. Neither is there an ImageNet pre-training equivalent in RL.

Lastly, RL algorithms suffer from (massive) instability while training. Much more than, for example, image-based generative models. Hyperparameter (a parameter whose value is used to control the learning process) tuning and thus getting reproducibility is incredibly difficult. Doing research is hard when your algorithm is both sample inefficient and unstable.

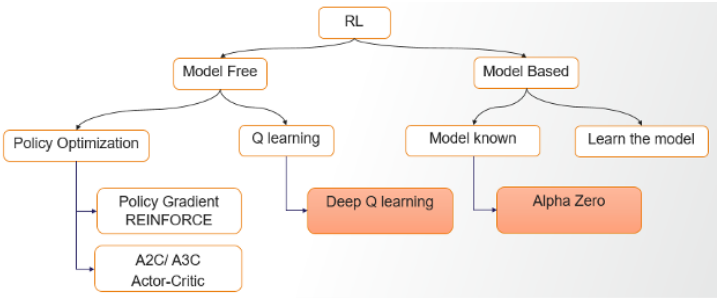


Figure 4 Taxonomy of reinforcement learning strategies.

Q-learning from zero

The world of RL may broadly be divided into model-free and model-based approaches. Remember, an environment provides an agent with a reward for the state the agent entered, next to the available actions that might be taken. The model denotes the agent's understanding of the universe it operates in. An example of a model-based approach is playing chess, where the agent is fully aware of the rules of the game. The initial AlphaZero release used traditional tree search to map out potential game moves, something that is only possible if you know the model (the rules of the game), next to neural networks to evaluate actions and states.

Conversely, model-free approaches assume nothing, and learn from reward alone. They are split into policy optimization methods and so-called Q-learning. We will use a Q-learning example to introduce this type of RL in a slightly more formal way.
Base concepts

1. The Return: G(t): Also known as future reward. It is the total sum of discounted rewards R through time.
2. The Value-function: V(s). Table or function telling you how good being in a certain state is.
3. The Q-function: Q(s, a). An imaginary score table telling you how good a certain action a in a certain state s is. As this table can be infinitely large for anything but simple toy examples, this is often a (trained) neural net.
4. The Advantage-function A(s, a) defined as Q(s, a) – V(s). This may seem a bit redundant, but it indicates that certain actions in a state are significantly better than the average.
5. The policy: π(a|s) is the distribution defining what action an agent should take, giving a state. Normally this is learned, but implicit in this q-learning example.

More formally, the return G is defined as:

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Note the discounting factor gamma. It is between 0 and 1 and balances future rewards as they may not have immediate benefits and might be more uncertain. There is no discounting for the first term! This will allow us to later reformulate and split G in terms of the immediate reward, and the discounted (estimated) future.

The value function V(s):

$$V_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] \quad Q_\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a]$$

Here π denotes a policy, which governs agent behavior. The E function denotes the calculation of the expectation value. Again informally: the state-value is the expected final return given the fact that the agent is in state s at time t.

Commented [D(13]: What is E? Expectation value?

Commented [D(14R13]: Yes, and in RL, mostly a fancy name for 'average over the batch'.

The Q-function Q(s, a):

$$Q_\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a]$$

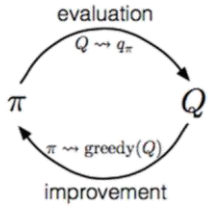
Informally: the expected final return if the agent takes action a in state s at time t
In Q-learning, the agent learns this Q(s, a) function. For toy-like problems, it is a table:

$$Q : S \times A \to \mathbb{R}$$

For every state and every action, it has a certain Q value. The higher the value, the better that particular action for that particular state is. A trained agent simply looks up the state he is in, and then selects the maximum Q-value action. Again, this approach is model-free because there is no extra a priori knowledge in this table about the environment or its rules.

Now learning this Q-function might seem very straightforward. The optimal policy is simply to always take the best action in a state, and these state-action pairs are stored in the Q-table or provided by the Q-function:

$$\pi(s) = \arg\max_{a \in \mathcal{A}} Q(s,a).$$



$$Q(s,a) = \frac{\sum_{t=1}^{T} 1[S_t = s, A_t = a] G_t}{\sum_{t=1}^{T} 1[S_t = s, A_t = a]}$$

Figure 5 Monte Carlo epsilon greedy Q learning.

After an episode is complete, you update the Q-values accordingly with all the discounted rewards that were collected for the actions you took and states you were in. And then normalize. To ensure exploration in the learning, you can use the epsilon-greedy approach. With a small probability the agent may also choose a random action in a state, and not the action with the maximum Q-value.

There are several significant downsides to this approach. Firstly, it needs complete episodes to learn from. Secondly, it suffers from high variance and is beyond inefficient. We will therefore introduce two more concepts that are central in RL: Temporal Difference Learning and the Replay-Buffer.

1.1.6 Temporal Difference Learning

Informally, we would like the agent to learn while it interacts with the environment. In the previous example, it needed complete episodes before updating the Q-table with the collected rewards. In temporal difference learning (TD), the agent learns or updates from rewards collected every action. This means that all learning is done from Q and V values at a time t and t+1. A pair of two. Instead of playing an entire game of chess from start to finish, the agent takes a (state, action, reward, next-state) pair and performs a learning update.

Slightly more formal, TD updates using existing estimates rather than actual rewards and complete returns

$$V(S_t) \quad \approx \quad R_{t+1} + \gamma V(S_{t+1})$$

Commented [D(15]: can this be left out?

Commented [D(16R15]: Hmm, yes, its a bit of a dilemma. On the one hand you have to introduce a lot of terms for an intro into RL, and TD-learning and the replay-buffer are landmark publications. On the other hand, pretty soon you are copying an entire RL textbook.

Commented [DR(17R15]: I only mean the 4 words here.

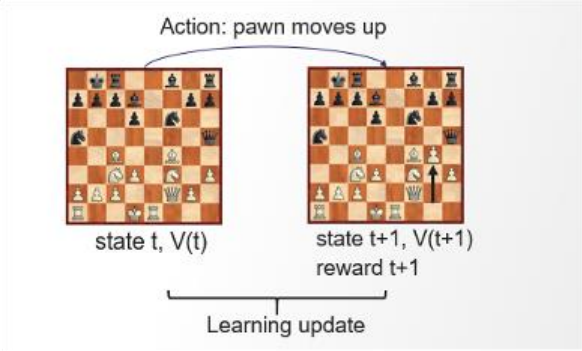


Figure 6 Learning strategy using only two states.

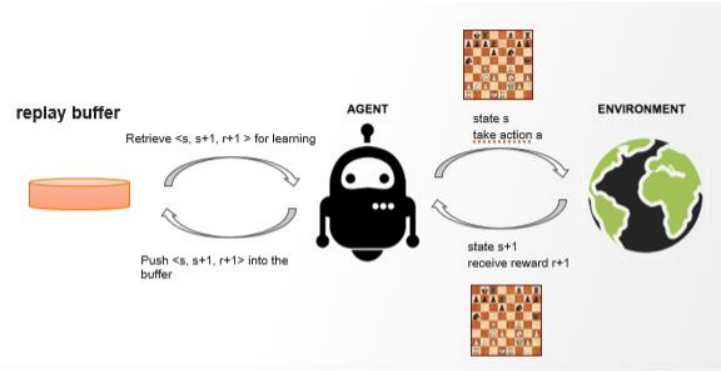


Figure 7 Setup using a replay buffer.

1.1.7 Replay-Buffer

In TD, an agent basically learns from sampled pairs of two. It turns out, that in practice it is much faster and better to learn from uncorrelated sampled pairs of states, actions and rewards. This means that if our agent learns to play chess, not only does it not learn from complete games. It learns from collected game moves, that come from different games and different moments in the game all together.

Commented [D(18]: rephrase?

This brings us the replay-buffer. The replay buffer is simply a place to store collected (state, action, reward, next-state) pairs that come from an environment. Initially the agent does nothing but collect these pairs. When the buffer is full, the agent samples those pairs, and performs a learning update.

2 Literature overview

2.1 Introduction to the state of the art

Artificial intelligence (AI) has evolved tremendously in the past decade. In recent years, data has proven to be precious as an enabler to enhance many engineering systems through AI. Examples include human face and voice recognizers, revenue forecasters for companies, mailing and calendar managers for individuals, among many others. Artificial intelligence can be divided into several categories as shown in Figure 8, which highlights the most recent trends: Deep Neural Networks, Ensemble Methods and Reinforcement Learning (RL).

In the scope of ASIMOV, while several of these techniques can be used, the focus is expected to be on RL, and thus RL will be considered here. This follows from: (i) traditional approaches to control and optimization problems that do not rely on simulators/digital twins or actual system data, such as optimal control, genetic optimization that have been extensively considered and in the applications at hand and their limits are already stretched; indeed in the applications considered in the ASIMOV project, AI techniques can bring significant breakthroughs; (ii) the fact that other forms of learning, and namely supervised learning, require a supervisor to label good and poor decisions. Labelling good and bad decisions to train a supervised learning algorithm would be impractical in the industrial use cases considered in ASIMOV. For example, for electron microscopes or for unmanned utility vehicle, it is very hard even for a human expert to access which actions are responsible for good or poor behavior of the system. It is rather a policy determining multitudes of actions for multitudes of system states that ultimately leads to a good or poor system behavior. Reinforcement Learning pertains to decision-making problems in real-time where decisions are taken based on previous experiences/data, based on very limited feedback information namely a reward for the overall behavior of the system, rather than an assessment on the usefulness of each action. This reward will most often be limited as it does not label good and bad decisions, and hence the problem of finding a policy for actions based on this reward is rather challenging. The real-time feature is crucial in the context of the use-cases considered in ASIMOV, e.g., the decisions on how to adjust a knob for an electron microscope need to happen while operating it.

Commented [H(19]: the references mentioned in this chapter are missing link to bibliography and some may need number update

Commented [H(20]: Link to figure should be added and updated.
It refers to Figure 8. *Machine learning taxonomy*

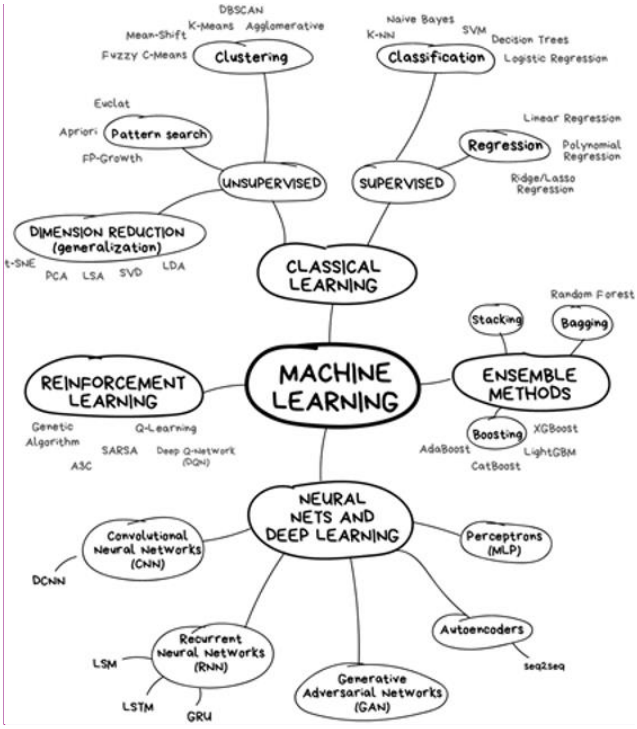


Figure 8 Machine learning taxonomy. Reproduced from [87].

Commented [D(21): VIlle: Fig 1 check source

Commented [L(22R21): Done - although it was taken from the blog article >> could be probably used as it is with the reference to the source

Reinforcement Learning (RL) can be carried out completely independently from models [3], i.e., the learning process for decisions policies just relies on experience/data from a real system. This is perhaps the most well-known form of Reinforcement Learning, where one can include algorithms such as Q-Learning, Temporal Difference, Actor Critic, Policy Gradients among many others. An introduction to Reinforcement Learning where these concepts are detailed can be found in Chapter 1 of this document – Introduction and more extensively in references [2] and [3]. However, on the one hand, the amount of data that is needed to run such algorithms is typically many orders of magnitude larger than what is physically/computationally possible, limiting their applicability in real-world scenarios; this is commonly known in the literature as the poor sample efficiency of the associated RL methods. On the other hand, it seems rather ineffective to ignore the models obtained from years or research following model-based approaches, when they often provide acceptable and even close to optimal performance.

However, there are also many RL approaches that rely both on models (either analytical models or simulation-based models) and data, or just on models, which will be surveyed next. As a simple approach one can replace the real system with a simulator, apply one of the aforementioned methods (Q-Learning, temporal difference, etc.), and transfer the resulting policy to make decisions for the real system. Yet, there are many other methods that depart significantly from this, as discussed in the sequel.

If we interpret a digital twin as a simulator, many of these methods directly apply to digital twins. However, thinking of a digital twin as a simulator is often reductive. Digital twins (DT) should be understood as 'living' extensions of models that mimic the behavior of their digital twin based on real-time data. In fact,

Commented [H(23): what does it mean? what is subject here?
I guess it should be physical twin or counterpart:
"Digital twins (DT) should be understood as 'living' extensions of models that mimic the behavior of their physical counter part based on real-time data."

Commented [A(24R23): I think the explanation comes just a bit later so I would leave it as-is.

a DT is a multi-physics & multiscale virtual model of a component, product, system and/or process, which is connected to real world by ways of data through its entire lifecycle and can contain closed-loop and open-loop block components. Rather than a one directional digital shadow of the process, a digital twin is a two-way process: it mimics the process and influences the process. The differences between a digital model, a digital shadow and a digital twin are illustrated in Figure 9.

Commented [H(25]: wrong reference.
Link to Figure *Figure 9 Illustration of the ...*

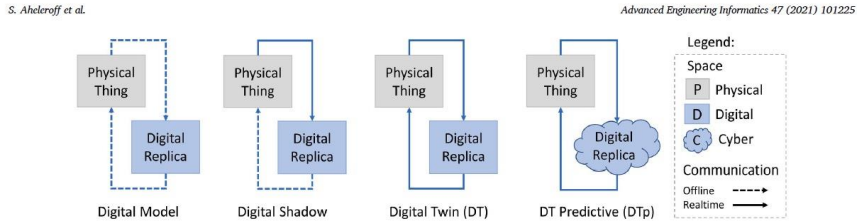


Figure 9 Illustration of the differences between a digital model, a digital shadow and a digital twin. Reproduced from [50].

More precisely, digital twins integrate physical, software and hardware models to form an up-to-date virtual representation of actual cyber-physical systems (CPSs) in operation updated based on data. The three main ingredients according to [1] are

Commented [H(26]: it would be clearer to write sth like:
According to [1], the three main ingredients of a DT are

- a model of the process/system,
- an evolving set of data relating to the process/system, and
- a means of dynamically updating or adjusting the model in accordance with the data.

Reinforcement Learning approaches that rely on (static) models are primarily surveyed in [8], but some works that rely on ('living' or online) combined online and offline approaches are surveyed in [12] and [13]. The reasons for this are twofold. First, the limited number of Reinforcement Learning approaches that directly rely on a digital twin but rather on (static) models/simulators; this is also related to the difficulties of modeling digital twins. Second, approaches that rely on models/simulators can be adapted to deal with digital twins and incorporate their extended features. For instance, if the process being replicated by the digital twin changes slowly (compared to the time constants of the system dynamics) one can rely on Reinforcement Learning techniques that assume static models/simulators, and furthermore set up an adaptation loop to account for the slow process changes.

In the remainder of the chapter an overview of reinforcement learning approaches and methods is given. The literature about Reinforcement Learning is by now very extensive and it is beyond the scope of this document to provide a complete survey. Thus, only the references deemed closer to the project are surveyed. Then more details on the methods deemed to be closer in spirit to the goals of ASIMOV are given, and finally some methods that combine RL and DT are mentioned.

2.2 Overview of Reinforcement Learning approaches and methods

It is important to start by mentioning that RL is not a mature field; there are many scattered methods and variants, and researchers refer to the same methods by different names. Hence, the perspective and highlighted methods provided in this overview might not be consensual, even among experts in the field. Here, we follow more closely [2] rather than the perhaps most standard reference [3], although some methods from [3] are also highlighted. This choice is motivated by the fact that [2] considers mostly simulation-based settings whereas [3] focusses on experimental-based setting. Therefore [2] is closer in spirit to the present project ASIMOV. However, there are many other important methods in the literature not covered in [2] and for those we rely mostly on the survey paper [21]. Specific methods that use digital twins are also discussed.

In [2], reinforcement learning is divided into several subfields. First, there are approaches that rely on:

1. *Approximations in value space*, where the cost-to-go or value function that encapsulates the reward/cost associated with a given action is the main focus. Once the value function is determined or approximated, the control policy (specifying actions as function of states) follows easily from searching for the decision that optimizes such value function. The approximation technique can be based on (deep) neural networks [14] or other forms of parametric approximation (an architecture that is off-line fitted based on data), or on-line simulation-based, which is especially interesting in the context of ASIMOV.
2. *Approximation in control/policy space*, where the control policy (decisions) that lead to best performance/reward for the system are directly searched. Policy gradient methods are prime examples, typically building on the key policy theorem [4]. Another example is expert learning where the system learns from the decisions of a human operator, which can be very interesting in the context of ASIMOV (e.g. a system can rely on experience from an operator calibrating an electron microscope).
3. A*pproximations in value space and in control/policy space*, which are combinations of the two previous approaches. Actor-critic methods are prime examples.

Second, there are optimal and approximate methods. For simple (low-dimensional) problems, tabular methods can lead to optimal behaviour. However, since for high-dimensional problems, such as the ones addressed in ASIMOV, some form of approximation must be carried out due to the curse of dimensionality, we shall focus on approximate methods.

Third, according to [2], one can divide RL into (i) model-based approaches, where *at least* some form of prediction (e.g. in computing expected values if the model is stochastic) uses *analytical computations based on a model* and (ii) model-free approaches, where *all* forms of prediction rely on data and simulators (and not on analytical methods based on models). This is a confusing nomenclature in the context of ASIMOV. In the context of ASIMOV, a digital twin can be a simulator and thus methods that would rely simply on this simulator and not on analytical models would be considered as model-free methods. Moreover, model-free methods do not distinguish from methods that use a virtual simulator/digital twin from methods that use real data from system experiments/experiences. Thus, this nomenclature will not be used here, I.e., here methods that use digital twins will be referred to a model-based reinforcement learning methods, rather than model-free methods.

While [2] sets the groundwork for methods used in Reinforcement Learning and provides several methods that rely of simulations of models such as stochastic rollout and Monte-Carlo tree search, it does not exhaustively survey the model-based RL approaches. Hence, we rely not only on [2] but also on the recent survey paper [21] to provide a short overview of such methods. This overview is given in Figure 10 and explained next. For convenience the references provided next will also include the author and year information to match with the ones provided in Figure 10.

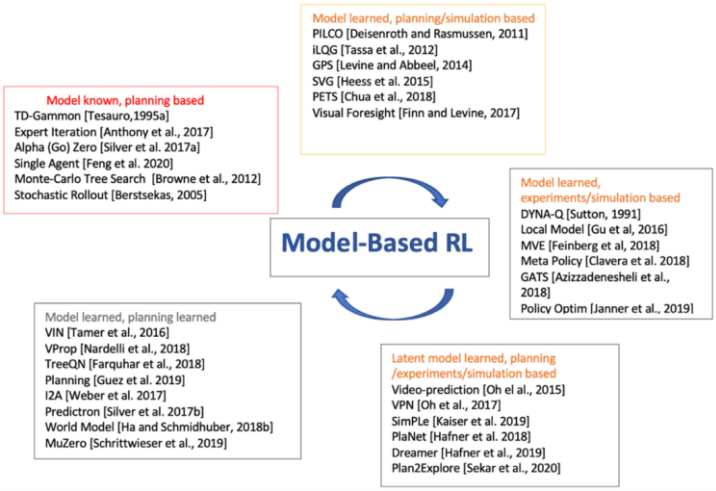


Figure 10 Summary of model-based reinforcement paper considered in the present document.

Model-based reinforcement learning methods are divided as follows in reference [21].

Commented [H(27]: link and update the following references

Commented [A(28R27]: This is up to date and fine

1) *Model known, planning based* (denoted by *Explicit Planning on Given Transitions in [21]).* This term pertains to problems where the model is known, and planning methods can be carried out based on this model. Especially interesting are planning methods that rely on simulations. Prime examples are games where the transitions/model is completely known. Model-based RL methods that have been proposed in the literature include the stochastic rollout for TD-Gammon [22, Tesauro, 1995a], also further developed in [2, Berstsekas, 2005], Monte-Carlo Tree Search [9, Browne et al., 2012] used in [24, Silver et al. 2017a], a famous paper that applies model-based RL for the Alpha-Go game. Also for the Alpha-Go game, a method known by Expert Iteration is proposed in [23, Anthony et al., 2017], see also [25, Feng et al. 2020].

2) *Model learned, planning/simulation based* (referred to as *Explicit Planning on Learned Transitions* in [21]). When the model is not known it can be learned from data, via system identification techniques. Then planning can be applied to obtain control policies. The methods under this class differ from the system identification technique. PILCO [6,Deisenroth and Rasmussen, 2011] is a famous method which has led to extraordinary results in some common benchmarks control problems such as the inverted pendulum. PILCO uses Gaussian processes for (nonlinear) system identification and then relies on policy search over a class of parameterized policies with sophisticated tools to compute policy gradients. However, Gaussian Processes do not scale to high dimensional systems, and the method is limited to applications with low-dimensional state sp aces. Differently from PILCO, iLQG [26, Tassa et al., 2012] uses quadratic approximation of the reward/cost function, linear approximation of the transition function (model), and online trajectory optimization, typically based on model-predictive control. Another trajectory optimization method is Guided Policy Search (GPS) [27, Levine and Abbeel, 2014]. GPS trains a parameterized policy in a supervised way by generating guiding samples with differential dynamic programming. Guided Policy Search (GPS) can be seen as a way of transforming the iLQG controller into a neural network policy. Stochastic Value Gradients (SVG) [28, Heess, 2015] is a variant which aims at reducing learned model inaccuracy by computing value gradients along the real environment trajectories instead of planned ones. Probabilistic Ensembles with trajectory sampling PETS [29, Chua et al., 2018] consists of an uncertainty-aware deep network to further model uncertainty in the transition probabilities (model), and combining this with sampling-based uncertainty propagation through probabilistic

ensembles. For a policy combining some of the ideas mentioned but using video as input to obtain decisions, see Visual Foresight [30, Finn and Levine, 2017].

3) *Model learned, experiments/simulation based* (referred to as *Explicit Planning on Learned Transitions* with *Hybrid Model-Free/Model-Based Imagination* in [21]). Rather than simply learning and using model to plan, the model can be used to generate virtual experimental data and combine it with the typically reduced already existing experimental data. This leads to a considerable increase of sample efficiency. A prime example of this framework is Dyna [12,13, Sutton, 1990, 1991], see Figure 4. Dyna not only uses the samples from real experimental to update the policy function but also uses these same real experiments to learn a transition model. Model-based imagined "virtual samples" are added to the real samples to improve the policy. There are many improvements and variants in the literature. For instance, Local Model [31, Gu et al, 2016] merges the backpropagation iLQG approaches with Dyna. Model-based value expansion (MVE) [32, Feinberg et al, 2018] is similar to the algorithm in [31, Gu et al., 2016], but controls for uncertainty in the deep model by only allowing imagination to fixed depth. Model-based Reinforcement Learning via Meta-Policy Optimization (MP-MPO) [33, Clavera et al., 2018] learns an ensemble of dynamics models and then learns a policy that can be adapted quickly to any of the fitted dynamics models with one gradient step. GATS [34, Azizzadenesheli et al., 2018] uses generative adversarial network in a similar context for obtaining a dynamic model and Model- based Policy Optimization (MBPO) uses short predictions with ensembles [35, Janner et al., 2019].

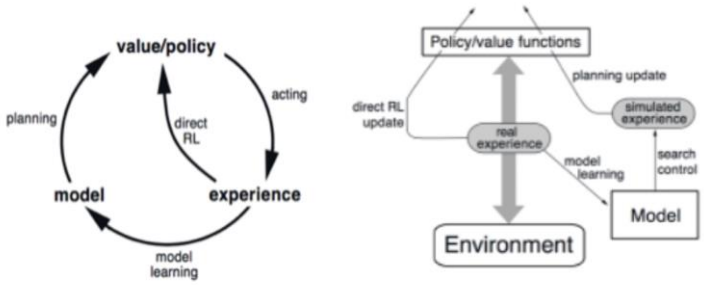


Figure 11 Dyna-Q, is a simple architecture that integrates models and experience [3].

4) *Latent model learned, planning/experiments/simulation based* (referred to as *Explicit Planning on Learned Transitions* with *Latent Models* in [21]). Latent models is a more compact alternative to the standard probability transition model. A latent model can simply be a state space representation, and their parameters suffice to fully characterize a model, rather than requiring transition probabilities to be known for every state. Latent models can be used for the different functions in a reinforcement learning algorithm wherein planning occurs in terms of this latent model. Latent models are important in applications where measurements are obtained based on video inputs. A popular application is in the Atari games [36, Oh et al., 2015] which was developed further into a more general framework Value Prediction Network (VPN) [37, Oh et al., 2017]. SimPLe [38, Kaiser et al. 2019] also uses video input but the latent model is formed with a variational autoencoder that is used to deal with the limited horizon of past observation frames. In turn, PlaNet [39, Hafner et al. 2018] trains a model-based agent to learn the dynamics from images and choose actions through planning in latent space with both deterministic and stochastic transition elements. Dream to Control is a concept introduced in [40, Hafner et al., 2019] by which world models enable interpolating between past experience, and latent models predict both actions and values. Plan2Explore [41, Sekar et al., 2020] is a recent work that aims at leveraging reinforcement learning with latent models for transfer learning.

5) *Model learned, planning learned* (referred to as *End-to-end Learning of Planning and Transitions in [21])* remarkably aims at learning both the (transition) model and the planning procedure. In other words, the neural network represents both the transition model and runs the planning steps. There are several recent methods in this direction, see VIN [Tamer et al., 2016], VProp [Nardelli et al., 2018], TreeQN [Farquhar et al., 2018], Planning [Guez et al. 2019], I2A [Weber et al. 2017], Predictron [Silver et al. 2017b], World Model [Ha and Schmidhuber, 2018b], MuZero [Schrittwieser et al., 2019]. Details are omitted since these methods are not in the spirit of ASIMOV.

2.3 Methods of special interest to ASIMOV

In the scope of ASIMOV the idea is to design first a digital twin of the process (e.g. Electron Microscope) and then use learning tools to design policies based on simulations from this digital twin. This description fits very well within the methods described above that fall into the second and third categories of model-based reinforcement learning, i.e., model learned, planning/simulation based (referred to as Explicit Planning on Learned Transitions in [21]; and model learned, experiments/simulation based (referred to as Explicit Planning on Learned Transitions with Hybrid Model-Free/Model-Based Imagination in [21]). All the methods described under these categories can be used for ASIMOV.

Of special interest for models where real data can be combined with synthetic data obtained with the digital twin is Dyna-Q, the simple architecture proposed in [12,13] that integrates models and experience as summarized in Fig. 11. Experience can both be used to make better models, which can then be used to plan decisions, or to apply direct decisions based on experimental data based methods. This is a fundamental paradigm useful in the ASIMOV use-cases that have very limited experimental data and can leverage this framework for sample efficiency. Another important method is PILCOS [6] for problems with low dimension. In essence, it is a policy gradient method which uses smart approximation to estimate the influence of changing the policy in terms of performance through a gradient based numerical method. The main bottleneck is the curse of dimensionality. For higher dimensional problems iLQG [Tassa et al., 2012] and Guided Policy Search (GPS) are very strong options.

While the full model is often not available in the use-cases of ASIMOV, the methods that assume that the model is known are still powerful methods such as stochastic rollout and Monte Carlo tree search and can either be directly useful or adjusted for the problems at hand. Stochastic rollout [10,11] is an approximate method for optimal control problems with stochastic disturbances in which (Monte-Carlo) simulations are used to approximate the costs-to-go when a so-called based policy is used. The consequence of each action at a given stage is evaluated based on these simulations and best decisions are selected. Monte-Carlo tree search methods [7,8,9] operate similarly but consider larger depths or lookahead horizons for decisions and prune which decisions to consider based on adaptive sampling schemes.

2.4 Digital twin based RL methods

Compared to model/simulator based RL there are relatively few works that exploit the enhanced features of a digital twin. Here a few of these works reviewed.

The use of digital twins is motivated by the fact that RL methods require large volumes of data. Digital twins can be used for accelerating the training phase in RL by creating suitable training datasets. These synthetic datasets can be cross-validated with real-world information. In [20] a framework for implementing a DT-driven approach for developing ML models is presented for an industrial use case . In [19], a similar framework to the one pursued in ASIMOV is applied to complex production and logistic systems, see Figure 12. Digital twins have also been used in the aforementioned approach of first designing the digital twin, then applying a standard RL method that learns based on the digital twin data, and finally applying it to the real system in many contexts. See [16] of such an approach for resource allocation policies to maximize the long- term energy efficiency, [17] for a robot arm application, and [15] for an application to manufacturing plants. While DT can be obtained from AI methods (see, e.g., the survey [18]), this is not the focus of ASIMOV.

Commented [K(29]: Monte Carlo

Commented [H(30]: Is it correct?
Fig. 5 is *Monte Carlo epsilon greedy Q learning.*

Commented [A(31R30]: corrected

Commented [H(32]: please check, shouldn't be "to maximize ..."?

Commented [H(33]: Not clear. Does it mean using RL to generate DT is not the ASIMOV focus?

Commented [A(34R33]: They mean to use AI to generate correct data for the DT, that is correct it was discussed much in the beginning of the process.

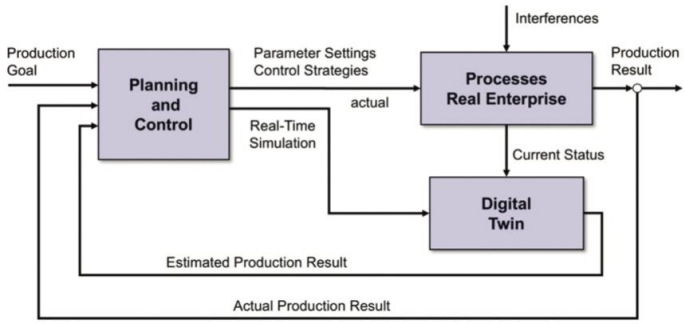


Figure 12 Reproduced from [19] in which this framework is proposed.

An established platform in the field of Digital Twin and AI is Bonsai [85]. It is a semi-automated platform for training AI systems using simulators. Bonsai intends to enable non-data scientists/engineers to implement industrial AI solutions.
Although superficially Bonsai tries to address partially similar concepts as ASIMOV, the platform lacks transparency and adaptability to address the complex systems we are addressing in ASIMOV and is only available on Azure infrastructures. The level of control over the training and the simulators needed for ASIMOV is significantly higher than provided by Bonsai. For example, if the Unmanned Utility Vehicle use case of AVL acts out of its operational bounds, it may lead to significant damages or even loss of lives. This requires control beyond a closed single-vendor system. However, Bonsai may serve as a quick prototyping platform for some use cases and as an inspiration for system architecture decisions, as will be worked out in WP4.

2.4.1 Functional view

From an abstract, functional perspective the training process is shown as an optimization process in Figure 13 (using the IDEF0 format [55]: input arrows impinging from left into the function box, outputs exiting at right, control inputs into the top; arrow from below indicates the system performing the function). From this perspective three major functions can be distinguished: the controller function, the modelled behaviour function, and the 'AI-function'. The controller (Optimization Control System) decides how well the training proceeds, and if the process should be stopped. The function F1 (performed by the Digital Twin) is responsible for creating the (modelled/simulated) system behaviour (expressed as 'system output'). Function F2 is responsible for learning from the behaviour provided to it, and for suggesting new settings in order to learn more. This is obviously performed by the AI system. The diagram also indicates on the arrows the essential information needed to enable the training. To realize a real solution for industrial application, these functions and information streams must be mapped to an implementation, see section 2.4.2.

Commented [H(35]: - the figure isnot referred or described in this chapter.
- what is "this framework"?

Commented [A(36R35]: no

Commented [H(37]: Is it introduced earlier?

Commented [A(38R37]: they mention the logistic application above, which is an industrial application.

Commented [H(39]: needs a description or reference here

Commented [A(40R39]: it is oke, it is a known microsoft produt, when going to Bonsai page it is all there, and we have reference to Bonsai.

Commented [H(41]: not relevant

Commented [A(42]: something went wrong here, I believe it is Section 3

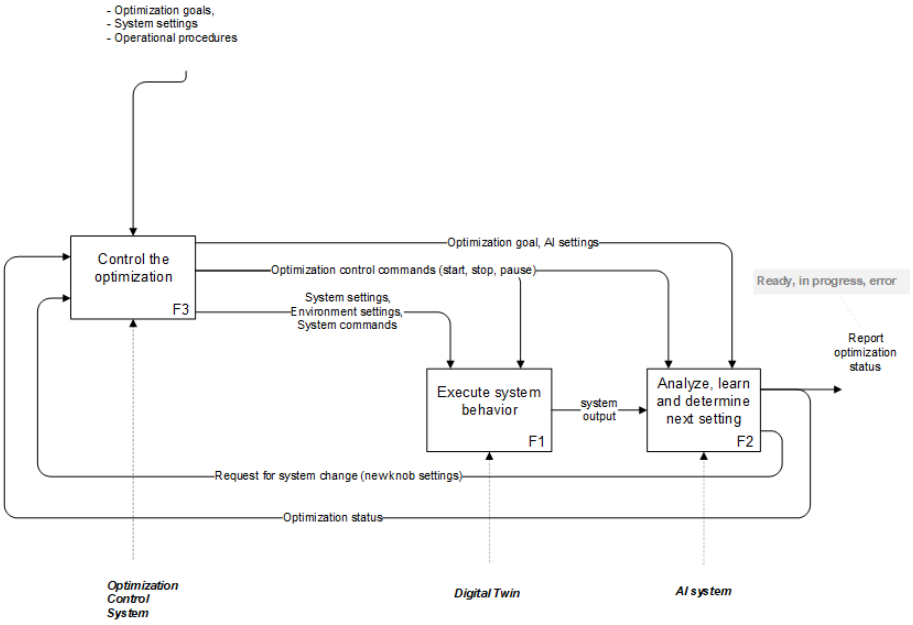


Figure 13 Functional diagram of the training process.

2.4.2 Information flow structure

Reinforcement learning is a cause-and-effect learning using the interaction with the environment. Learning by interaction is a fundamental idea in all learning strategies. The objective is how to convert a situation into action and to maximize a numerical **reward** signal. Trial-and-error search and delayed reward are two important features of reinforcement learning. At high level, reinforcement learning has the following main elements: an agent, an environment, a policy, a reward signal, a value function and optionally a model of the environment.

An agent is both learner and decision maker; it must sense the state of its environment at some level and take actions with a goal (objective) to achieve the desired impact on the state. The agent must be able to learn from its own experience and make a trade-off **between exploration and exploitation**.

The thing an agent interact with is called environment. An agent selects **actions** and the environment responds to it and presents its new situation to the agent. The agent cannot influence/impact the rules of the environment.

Commented [K(43]: selects

A **Policy** defines the learning behaviour of the agent at a given time. It is kind of stimulus and response rules. It can be stochastics i.e., specifying probabilities for each action.

A Reward signal defines the goals of the reinforcement learning problem. The main goal of the agent is to maximize the reward signal in long run.

A **value function**, reward signal tells what is good in intermediate terms value functions tells what is good in log run. Rewards are primary sense and value is the main goals in long term. Reward is given by environment, but value must be estimated and re-estimated from the sequence of actions taken by agent.

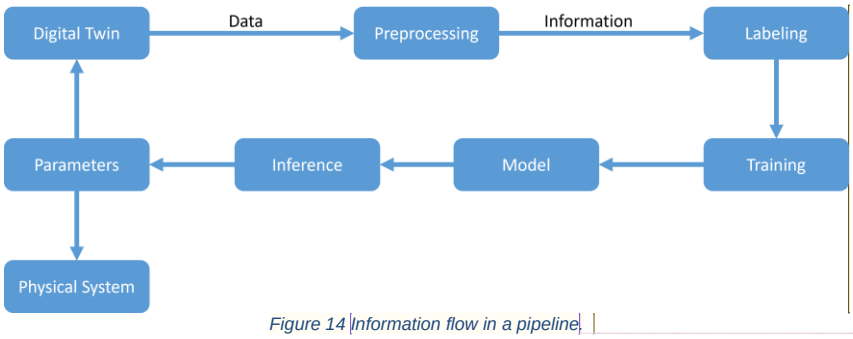


Figure 14 Information flow in a pipeline

Digital twin will generate data that will be pre-processed and in labeling step, policy, environment, actions, reward, value function and model is defined. Learning happens in training. Later it is used in inference to provide input to the digital twin and physical system as well. Digital twin will get the physical system response and generate new data for the next iteration.
In the multiple iterations there can happen three scenarios 1) data drift 2) concept drift 3) either small or no change happen in data or concept. In the first case retraining helps and either online training or incremental training can be employed. In the second case usually retraining is used to train a new model for the new concept. In the third case normally, no training is required, usually post processing is added to adjust the output.
To manage the iteration cycle, Machine Learning (ML) life cycle management is used. That is built on machine learning operation formally called MLOps. ML lifecycle management [78] is essential part of continues integration and continues testing (CI/CD). ML lifecycle management on public cloud [79] is also supported. Azure [79], Amazon SageMaker [80], Google Cloud AI Platform [81], MLflow [82], TensorFlow Extended (TFX) [83] is mainly for deployment.

2.5 An initial assessment of strategies

The different Reinforcement Learning algorithms provide benefits and disadvantages when used in specific situations. For scenarios which differ from the standard RL testing environments, their performance cannot be estimated at the current state of planning. The identified generally suitable algorithms can however be used to have a basis of possible algorithms that need to be looked at. As more use-case-specific benefits and disadvantages can only be observed when working with these algorithms, their applicability inside the ASIMOV solution were. Some properties, like the dimensionality of the state and action spaces, as well as information if the actions need to be discrete or continuous will determine the usability of each algorithm. Computational efficiency and their robustness to model imperfections, like the ones provided by a digital twin, will further determine which algorithms can be used for ASIMOV. The choices made in the mentioned respect for the specific use-cases and the results can be read in WP1.

Commented [H(44]: - Rectangles in diagram are representing distinct types of concept. Could you differentiate them? like differentiating processes and products
- the arrows are missing labels, for example what is going from Model to Inference?

Commented [A(45R44]: Later on the architecture of the AI will be detailed further, I think that is the place to follow up and imporve

Commented [B(46]: Replace with a "cleaner" picture

Commented [DR(47R46]: Iftikhar: can you adapt this picture? typo: inference

Commented [B(48R46]: done

Commented [H(49]: It is not depicted by the figure, there is no arrow from physical system

Commented [A(50R49]: I agree those were some initial ideas, maybe we should erase the graph and paragraph but I leave it like this as a legacy thing, it is also not very inconvenient to have it in

Commented [H(51]: they are not explained here. Only solutions are briefly described

Commented [A(52R51]: same as I mentioned above, this is a legacy thing, we did not actually work on this but for sake of legacy we can just leave it.

Commented [H(53]: since it is the last version of this deliverable, I'd suggest to include only the results and not the plans

Commented [A(54R53]: I am hesitating as that is a fair remark. Maybe we should point to where the examination lead, and that information is in WP1. I will do that

Commented [A(55]: This is not even considering the broader context of RL, I would take that away in favor of the more context rich introduction above where all different approaches within RL are mentioned.

Commented [B(56R55]: I agree, that we should also include the more conventional RL algorithms you mentioned above

Commented [DR(57R55]: We can move this section before 3.5.3. as an intro to the piece of Ilona. (We can be pragmatic here, this doc is not the final one!)

3 Preliminary Approach

3.1 Unmanned Utility Vehicle - Sub Use Case 1

Looking at the first sub use case of the Unmanned Utility Vehicle, the focus lies on creating an efficient test plan for vehicles in a given environment. The goal of these test plans can either be identifying critical scenarios, in which faulty perception or acting leads to dangerous traffic situations, or it can serve the purpose of efficiently gathering data from the vehicle's sensors in different driving situations and operational areas. The last goal is also to gain representative data that can be used to efficiently create behavioural models or find the parameters for physical models of certain vehicle components.

A test plan will consist of multiple individual test cases, which describe concrete instances of traffic scenarios. These traffic scenarios describe the dynamic surroundings of the vehicle under test. Such could be the speed of an overtaking car, the moment a pedestrian walks on the street, etc. In addition to the definition of these dynamic scenario descriptions, also static properties of the surroundings are described in a test case. Such properties can e.g. define the time of day, weather, road pavement, tree density at the roadside, etc.

There are endless possible combinations of different parameters to define instances of these test cases and since testing time is limited, not every combination can be tested. Hence, a system is needed, that systematically suggests test cases, based on the results of already tested combinations. Of course, as more measurement data is generated by more test cases, always leading to new data with some kind of new information, certain stop requirements need to be implemented. This can either be expressed as something like the overall time of testbed usage, or some quality metric of the collected data.

To evaluate the effectiveness of a test plan, the resulting data has to be analysed. Ideally, a non-redundant data set is acquired. Meaning that every datapoint inside the entire data set inherits information which is not included elsewhere in the data set.

To realise such a system, an AI-learning technique is needed, which can optimise the sequence of test cases. In the end, the overall generated dataset is most important, while the individual test cases themselves play a lesser role. This, as well as the sequential character of the task, makes reinforcement learning agents suitable for this task.

In order to make the system robust against variations of the vehicle under test, the training of such an agent will use a virtual vehicle and not a real vehicle on a testbed. That way, the training process can be sped up and variation of the vehicles under test can be implemented and automated. The use of such a virtual vehicle can be seen as the digital twin, which is used to pre-optimize the system, the test plan generation in this case, before applying it on the real test system. A direct link between the digital twin and the real vehicle is unlikely, as the purpose of the use case lies in testing the vehicle and having a digital twin of this exact vehicle before starting is usually not the case.

When it comes to selecting a suitable reinforcement learning algorithm, we need to take the following challenges into account:

- Continuous and discrete action spaces: when creating the variations of test cases, several parameters that define such a specific test case are discrete values (e.g. selecting the type of Traffic participant, road segments, etc.) while others are continuous (ego vehicles initial speed, time of day, etc.)
- High dimensional action spaces: The description of a test case requires a lot of variables. The RL agent shall be able to cope with the resulting high dimensional action space.

Unmanned Utility Vehicle - Sub Use Case 2

The second sub use case focuses on the optimization of a sensor's perception, which is essential for the vehicle's safe operation. To measure the quality of the perception, the sensor’s output has to be compared with the actual surroundings of the vehicle, while it moves through an environment.
In the training and operational phase, the environment of the vehicle will be virtual, which makes the segmentation process very effective.

During the actual optimization process, the results from the comparison of these segmentation images, serve as input for the reinforcement learning agent, which proposes new internal parameters for the sensor, as well as varies the positioning and orientation of the sensor on the vehicle. As especially the repositioning of the sensor on the real vehicle cannot be done easily, the optimization is initially done on a digital twin of the vehicle and afterwards transferred and tested on the real vehicle.

3.2 Electron Microscope

Commented [B(58]: We need STEM Input here

Commented [D(59R58]: True true :)

Properly setting up electron microscopes is a labour intensive and repetitive job. It is also a continuous process, depending on a whole host of factors, from the microscope itself, to its operating environment, or simply the time it has been in operation. Short of human intuition there isn't a fixed recipe to tune microscopes. Operators do commonly look at the acquired images to infer clues on the current state of the microscope, one of these being the so called Ronchigrams.

In short there are three components in play. Firstly, there must be a form of feature extraction that takes microscope images and maps or transforms them into parameters. Secondly, these parameters can then be used as input for a reinforcement learning algorithm. Lastly, this means that a digital twin also needs to be able to generate synthetic images that adhere to the same (unknown) distribution as the real images.

Currently the only digital twin (model) we have deployed is a surrogate simulator that generates synthetic Ronchigram images according to pre-set aberration parameters. We also have a trained regressor that can blindly ascertain the original aberration parameters from a single synthetic image. Given that there is no complete digital twin as of yet, and the fact that the simulator is governed by known equations, we currently see a number of ways forward.

3.2.1 Offline reinforcement learning

Looking solely at the (S)TEM use case, the current Thermo Fisher Scientific simulators are not nearly real-time. They can, however, deliver sufficient amounts of data covering a wide range of settings or parameter-space suitable for machine learning.

Offline reinforcement learning seeks to find policies without any live interaction with an environment. Instead, it has to learn from previously logged transactions. Obviously, this is very promising for RL systems that will be deployed and for which a learning environment doesn't exist.

However, prior collected training data will naturally never cover the complete state-space of the environment almost by definition. This means that agents need to learn how to deal with new unseen state-action pairs. Often this means that agents should not drift into unknown states and avoid actions whose rewards or consequences can't be predicted from the logged transactions. Common problems include agents being overly optimistic for new unseen actions, resulting in poor policies. This is countered by balancing the need to learn policies that maximize the return, whilst making sure they remain close to the support of the logged transactions. [60,61,62,63].

As the environment (or digital twin) has yet to be built, training RL algorithms on existing static datasets could be a good first step.

3.2.2 Physics (informed) Machine learning

Recently there have been developments where machine learning algorithms either completely replace traditional partial differential equation (PDE) solvers [8] or incorporate (differentiable) physical models in their architecture or loss function [9].

Purely data driven supervised ML models that replace solvers, for example for solving Navier-Stokes equations have been shown to generalize surprisingly well [64, 65, 66, 67]. Their other immediate advantage is that once trained, they are significantly more computationally efficient.

Backpropagation may also be used to find solutions for physical models by defining a network that explicitly encodes the differential equations [68]. Other options are to use an existing physical model in

the loss function, during training. This gives the ML algorithm a priori information and ensures the solution space gets restricted in certain locations. Work from [69] leveraged a neural network next to an iterative PDE solver to correct its numerical errors.

Although novel, this subfield is promising for Thermo Fisher Scientific cases. It currently uses traditional physics-based models to simulate certain parts of the (S)TEM. Potentially, both the RL agent that learns from the digital twin, next to the digital twin itself, may profit from the interplay between ML and a known physics model. This also means that the border between the twin and the agent gets blurred.

3.2.3 Distribution mismatch

Currently, there is a significant gap between real microscope images and those from the Thermo Fisher Scientific simulators. We expect this to remain the case for the foreseeable future. Currently research is underway to close this gap. The following avenues are currently explored:

- Generative models
- Metric learning
- Semi-supervised learning

Generative and in particular generative adversarial nets (GAN) based models [74] may be trained to generate image-to-image transformations [73], whilst keeping certain aspects intact. The primary challenge here is that TEM images are measurements of real physical phenomena. These properties must remain intact, irrespective if the image looks 'real' to a casual observer. Work from [72] added such constraints and is an avenue that will be explored. This is however a relatively novel field with little published work.

Metric learning aims to ascertain embeddings where certain feature vectors are close. Here this would mean finding embeddings that are dependent on lens aberrations alone, and less on the visual discrepancies between real and synthetic data.

Lastly, the family of recently developed semi-supervised methods offers a possibility to combine real and synthetic data. In particular as this data doesn't need to be labelled for the most part, and due to the fact that it is much easier to generate enormous amounts of such data with a twin or simulator. Robust features might be learned from siam-network alike approaches such as SimCLR [75] or its descendants [76]. Lastly there might be lessons learned from the progress in language modelling, in particular BERT, for attention-based unsupervised learning of features [77]. Ideally this will lead to a situation where the bulk of the feature learning can be done from synthetic data and a pretext task.

3.3 Input from WP1: commonalities

The use cases, as described in the WP1 document IR1.1 [86] are: STEM (calibration to get good image), UUV.1 (test generation), UUV.2 (sensor placement). The TEM and UUV.2 use cases have more in common as the simpler versions of the use case have low dimensionality of inputs; and as intermediate output an image with likely a small vector describing the various aspects of image quality (STEM) or deviation from ground truth (UUV.2. For TEM and UUV.2, the DT-trained AI is used to optimize the physical system. Use case UUV.1 is more elusive: the AI is optimizing the DT and the ability to create good test plans.

A commonality aspect not explicitly mentioned in the document is whether the DT is modelled as having inherent randomness, or not. This was touched upon in some Asimov internal meetings, and also by the fleet aspect and sensitivity of an uncontrollable physical environment in 4.1.2 in the WP1 doc. Also, 4.4.2 mentions for TEM hysteresis, drift, pollution in the DT and dynamic environment based on limited nr of pars for UUV, which from the AI's perspective could be considered as random or an unknown state. Possibly, simpler versions of the use cases have no randomness, in which case simpler techniques can be considered. However, there could many reasons why calibration of a different microscope in different settings may give different results and the "internal state" or machine properties could be viewed and modelled as random. For the UUV use cases, the scenario could be fixed on a higher level (number of

pedestrians, type of weather) and details could be random (exact path over time, exact density of rain droplets and wind gusts). Parameters governing the randomness (e.g. standard deviation, correlation matrices) may be part of the DT input vector. This randomness could perhaps be viewed as an unknown state.

The use cases have in common that efficient computation times are needed (4.3.1), however this seems slightly contradicted by the later statements; TEM is applied over and over in operation and speed is a low priority (not much slower than a human); the UUV's speed is called high priority, and it is applied during development ("deployment for a specific domain of operation by small companies") in 4.2.2. The documents mentions that different control parameters play together towards an outcome ("control parameters are dependent") which likely means the presence of interaction effects; i.e., the effect of x1 on an outcome depends on the values of x2, x3,…

The output of a DT and reward system are complex in STEM (image based) and UUV.1 (what constitutes new, valuable information) for which a reward function must be constructed. "Good output" for UUV.2 will be more straightforward: a comparison of sensor measurement versus ground truth.

The consequences for the architecture of learning / system are that:

- The DT needs to have efficient computation time
- The DT quite possibly are non-linear in the inputs
- Reward system for uses cases TEM and UUV.2 need considerable effort
- Role of "inherent randomness", and whether it needs to be part of a DT, and to what extent it can be handled by RL.
- Real time decisions are needed as the use cases deal with dynamical systems

There is limited information about the environment, it is sequential in nature and there is no supervisor available

The final DT+ AI solutions will be judged on the following generic characteristics:

- The amount of training data needed to achieve acceptable results (data efficiency)
- The speed of learning (convergence speed)
- The stability/instability of the learning and subsequent operational behavior
- The success rate of finding an optimum as such (robustness)
- The scalability with respect to computational resources / distributed training & merging

3.4 Suitability of different learning algorithms

3.4.1 The Markov Decision Process as starting point

Reinforcement learning problems are often modeled formally as a Markov Decision Process (MDP). An MDP comprises state S, action A, transition T and reward R; policy and value are other important concepts of the MDP. The definitions of these concepts can be found in the introductory part of the chapter. MDPs can be described as model-based and model-free solution approaches.

Commented [A(60]: I see my part and my feedback were not incorporated. I provided them orally. So hereby I put it.

Commented [A(61R60]: I also sent an email about this

The goal of reinforcement learning is to find the optimal policy, which is the function that provides the best action a given a state $s \in S$. The policy contains the actions of the answer to a sequential decision problem: a step-by-step prescription of which action must be taken in which state, in order to maximize reward for any given state. In deep learning the policy is determined by the parameters (or weights) of a neural network. This policy can be found directly—model-free—or with the help of a transition model—model-based.

To find the policy by planning, models for T and R must be known. When they are not known, an environment is assumed to be present for the agent to query in order to get the necessary reinforcing information. The samples can be used to build the model of T and R (model-based reinforcement learning) or they can be used to find the policy without first building the model (direct or model-free reinforcement learning).

3.4.2 The learning and the planning problems

3.4.2.1 The learning problem

Capturing the probabilistic transitions T from state S and action A to state S' of the environment of choice is a natural way of capturing the core of the environment dynamics (model). Therefore in model-free RL, there is no learning problem, and all algorithms known to model-free RL are typically solving the optimal policy.

Commented [T(62]: Could be a randomized function described by a transition matrix (Markov Kernel) dependence on the action.

In the use case for ASIMOV, the learning part could comprise the T and R functions of the Digital Twin of the TEM or UUV, or the relevant processes within them. In the literature, a digital Twin is mostly a digital behaving copy of a real device, system or machine with an online data feeding connection to the real twin.

When a good transition model of the environment of interest is captured, model-based RL is much more sample efficient than its counterpart model-free RL, because we are explicitly capturing rules of interest, human knowledge, physics rules etc. When working with a faulty or incomplete model of the environment of choice, the RL will not succeed at the task of finding a solution of the given model/DT.

How to solve the learning problem is naturally not easy, there are many methods available to construct these functions at different levels of knowledge and abstraction. In the case of RL, building a transition model of (parts of) the Digital Twin can comprise many approaches of which we will cover a few in this document.

If we assume that we want to work model-free (without the agent knowing from the transition or reward models) as just described, then the environment will just be sampled and the next paragraphs can be ignored. Note that the samples here will be used just as the environment generates them and afterwards 'thrown away'.

But if there is any value in trying to be data efficient while having confidence in the fact that a model can be given or built for the T and R functions (which can be complex), then the following approaches can be considered:

- Given transitions: T and R functions are known. The transition rules can be derived from the problem directly. Those rules are documented and ready to be used.
- Learned transitions: the environment can be sampled to build the T and R models.

Combined approaches lead to use the environment and the model samples to train the policy function. (Hybrid model-free/model-based imagination).

3.4.2.2 The planning problem

Commented [B(63]: This seems to be a (Sub-Heading)

Solving the planning problem can be done by a computational process that uses a model to create or improve a policy. In this process the optimization is usually based on state-space planning or plan-space planning. The planning problem, thus, in a sense solves the same problem as model-free RL, but as models are used to generate the samples (totally or partially), different methods are used than in model-free RL.

Note: Some approaches integrate the learning and planning into an end-to-end approach. While no further explanation is provided at this stage about the workings on this procedure more information can be found at [2]. In the next section some approaches in this category will be mentioned too.

3.4.3 Which approaches should we try and why?

Commented [B(64]: Incoporate into document structe by using the respective headings

As can be observed, both model-free RL and model-based RL have proven to be useful in solving a variety of problems. A model-free RL approach is attractive as the model building for the use cases can

be complex and introduce too many errors. A model-based RL approach has the advantage of the promise of a cleaner policy given the sample efficiency.

As the TEM case has components of stochasticity and determinism, and it is not clear which component is the primitive it is worth trying an algorithm within each model-free approach.

From the model-free RL perspective, the approaches that can be tried are as follows:

- For stochastic problems, often policy optimization algorithms work best à Try PPO or similar
- For deterministic problems, often value optimization methods work best. à Try DQN or similar.
- Combinations of both are also possible. à Try SAC or similar.

From the model-based perspective, a table is provided with multiple approaches, from which the most promising should be selected depending on the use case of interest.

In the TEM use case, given transitions and rewards functions can be considered very unlikely, as far as the knowledge today dictates. So approaches based on learned transitions are considered more likely, and maybe in the future, end-to-end approaches deserve a try, but should surely be seen as less likely given the knowledge build up it requires.

Table 2 Learned transition explicit planning approaches.

Commented [B(65]: We need to align table numbers with the rest of the document

Approach	*Learning*	*Planning*	*Application*
PILCO [Deisenroth and Rasmussen, 2011]	Gaussian Processes	Gradient based	Pendulum
iLQG [Tassa et al., 2012]	Quadratic Non-linear	MPC	Humanoid
GPS [Levine and Abbeel, 2014]	iLQG	Trajectory	Swimmer
SVG [Heess et al., 2015]	Value Gradients	Trajectory	Swimmer
PETS [Chua et al., 2018]	Uncertainty Ensemble	MPC	Cheetah
Visual Foresight [Finn and Levine, 2017]	Video Prediction	MPC	Manipulation

Table 3 End-to-end model-based RL approaches.

Approach	*Learning*	*Planning*	*Reinforcement Learning*	*Application*
VIN [Tamar et al., 2016]	CNN	Rollout in network	Value Iteration	Mazes
VProp [Nardelli et al., 2018]	CNN	Hierarch Rollouts	Value Iteration	Navigation
TreeQN [Farquhar et al., 2018]	Tree-shape Net	Plan-functions	DQN/Actor-Critic	Box pushing
ConvLSTM [Guez et al., 2019]	CNN+LSTM	Rollouts in network	A3C	Sokoban
I2A [Weber et al., 2017]	CNN/LSTM encoder	Meta-controller	A3C	Sokoban
Predictron [Silver et al., 2017b]	k, γ, λ-CNN-predictr	k-rollout	λ-accum	Mazes
World Model [Ha and Schmidhuber, 2018b]	VAE	CMA-ES	MDN-RNN	Car Racing
MuZero [Schrittwieser et al., 2019]	Latent	MCTS	Curriculum	Go/chess/shogi+Atari

There is another category of learning in RL algorithms that was not mentioned but that can be of use in the currently treated use cases. This approach is an imitation approach, in which an agent learns by imitating the human expert. In the future months this option should be looked at and considered too.

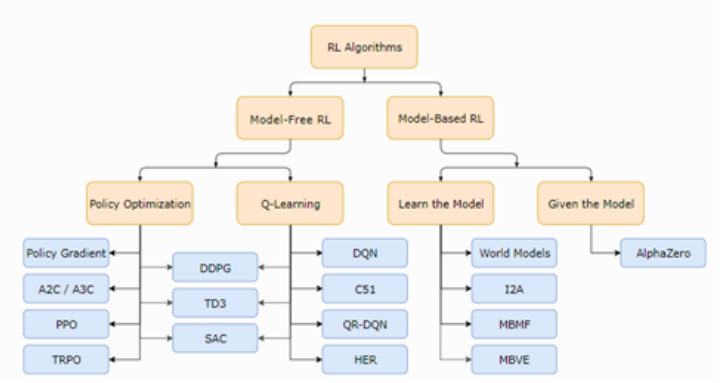


Figure 15 A taxonomy of reinforcement learning algorithms.

4 UUV Use Case

4.1 General Introduction

4.1.1 Sub Use Case UUV.1: Scenario Generation/Optimization

Usually, for testing vehicles on a testbed, a test plan has to be created to reflect the needs of the function/component under test. The resulting test data gathered often contains many datapoints with little new information. The UUV.1 use case aims to improve data quality by adapting the test plan in a way that every tested scenario contains as much information as possible. To achieve this higher information density, certain elements of the scenario, like the time of day, placement of cars on the roadside or density of trees are varied automatically to create a scenario that fulfils the requirements. The two main ingredients for evaluating the data are novelty and criticality. Ideally, the gathered data should be as novel and therefore as dissimilar to already seen data as possible, while also containing a traffic situation that is quantified as critical, by a combination of safety Key-Performance-Indicators (KPIs).
The optimization will be done using a Reinforcement Learning (RL) agent that proposes modifications to a basic scenario as actions and receives an estimate of the novelty value as well as the criticality of the modified scenario as a reward. An adaptation to also include comfort KPIs regarding the driving function besides criticality KPIs is possible. The basic structure of this use case can be seen in Figure 16.

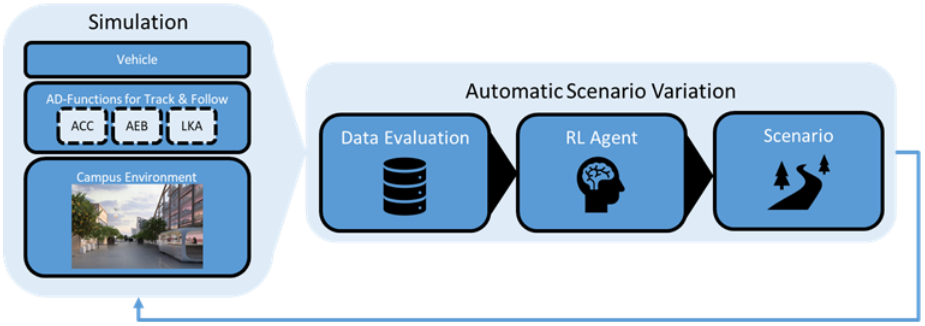


Figure 16 Simplified overview of the UUV.1 use case, that features a Reinforcement Learning-based Automatic Scenario Variation to find critical versions of a Scenario

As the character of the state and action vector, the transfer of learned techniques to propose the wanted scenarios, and the amount of generated data highly influence the ideal RL algorithm to solve this optimization problem, T3.3 shall be used to develop concepts which fit the use case.
To further elaborate on the use case, the following section will expand the view on the way the RL agent shall interact with its environment.

4.1.2 Action, State and Reward

The *action*, being the modifications to the scenario the agent wants to take, is the input for the configuration of the environment simulation, which can be coupled with the Digital Twin (DT) or the real Cyber-Physical System (CPS). The configuration settings for the environment can be divided into dynamic parameters, that define the movement of the traffic participants (cars, pedestrians, etc.) and into static parameters that define stationary objects like parked cars, trees or other assets that do not contribute directly to the driving scenario itself. The dynamic parts of the scenario shall be considered fixed and will not be changed by the RL agent, as there are already methods available to identify critical values for these parameters. To limit the changes the agent can propose, initially only a few selected static parameters of the environment are changed. These are parked cars on the roadside, the density of trees in predefined areas, as well as the time of day. Later in the course of the project, additional parameters will be controlled by the agent. Further expansions could be the road surface or even parts

of the dynamic parts of the scenario like the velocity of the ego vehicle, as well as the velocity of other traffic participants.
These parameters will be written into a JSON formatted file, which is used to parameterize the simulation. Some of the parameters are discrete, others of a continuous type. This has to be taken into account when choosing the appropriate algorithm. Additionally, if the number of parked cars is one discrete parameter, for instance, this parameter affects the number of additionally needed parameters for positioning the cars. These hierarchical parameters are only optional, however, as there are alternatives with conventional fixed length action vectors possible, by only considering a fixed number of cars.
The *reward* is partially calculated based on criticality metrics. A detailed description of the reward and postprocessing in general can be found in the ASIMOV Deliverable D2.2. In short, the criticality part of the reward calculation is based on the metrics listed in [A1], where multiple suitable KPIs were selected, normalized and combined in a weighted sum. The novelty detection part of the reward is also described in D2.2. It is an autoencoder neural network which tries to replicate its input data as output, while going through multiple compression and decompression steps. This principle leads to the neural net learning the relations between its input. Therefore, the reconstruction error of the network can be seen as a quantification of the dissimilarity of the current input compared to the training data of the autoencoder. A low reconstruction error represents low novelty of the data and vice versa. By continuously retraining the autoencoder, the basis of training data is expanded, and new information can be incorporated in the neural network. The anomaly value is then combined with the criticality KPI via multiplication.
The *state* provides the basis of information for choosing the next action. In our case, it is represented by multiple intermediate results for the KPI calculation as well as the anomaly value of individual data channels. That way, it is possible to pinpoint which modifications need to be made in order to improve the test plan.
The *objective of sub-use case UUV.1* is to create efficient test plans for vehicles under test in a given environment, i.e., to optimize the set of test scenarios for a test plan with respect to the degree of challenging the UUV with safety and comfort, e.g., smooth ACC speed change.
In the first iteration of the sub-use case, we create scenarios that are as (safety-)critical as possible, while the next iterations will yield scenarios that are as (safety-)critical as possible and have the maximum dissimilarity.
The RL framework for UUV.1 is an episodic RL task that is using a model-free approach. Within one episode, the characteristics of the vehicle do not change. From episode to episode, vehicle characteristics are modified slightly for obtaining a robust RL agent (see Section 4.2.2). However, we stay inside the vehicle's operational design domain (ODD). We apply a model-free approach, meaning that we do not assume to know the states' transition probability, which provides the estimate of the next state given the actual state and action. This environment model could be learned through experience. However, this is generally too difficult to achieve in practice, especially for complex systems.

4.1.3 Sub Use Case UUV.2: Sensor Optimization

The tuning of sensor and perception parameters in a way that the vehicle can perceive its surroundings most accurately is no easy task. To automate this parameter tuning for calibration of sensors, sub-use case UUV.2 uses the test plan created in UUV.1 to run a series of tests, in which the ground truth position and orientation is used as a reference for comparison with the perceived object positions and orientations. As the environment can be virtual even when paired with a real sensor setup, information about the ground truth of all objects is available. The *goal in sub-use case UUV.2* is to tune the parameters in a way that the perceived objects match the ground-truth objects as well as possible, i.e., to optimize the sensor's perception.
For comparison of the perception results, the resulting segmentation images could be compared. However, as the UUV.2 case is dependent on results from sub-use case UUV.1, we proceed in a sequential way, and hence a detailed description of input and output datatypes has not yet been created.

4.1.4 Overall Structure

For a better understanding of the applied RL approach, we first introduce a mapping of both use cases to the ASIMOV reference architecture, developed by the Working Group. This architecture comprises a mapping of the UUV.1 and UUV.2 use cases and is shown in Figure 17.

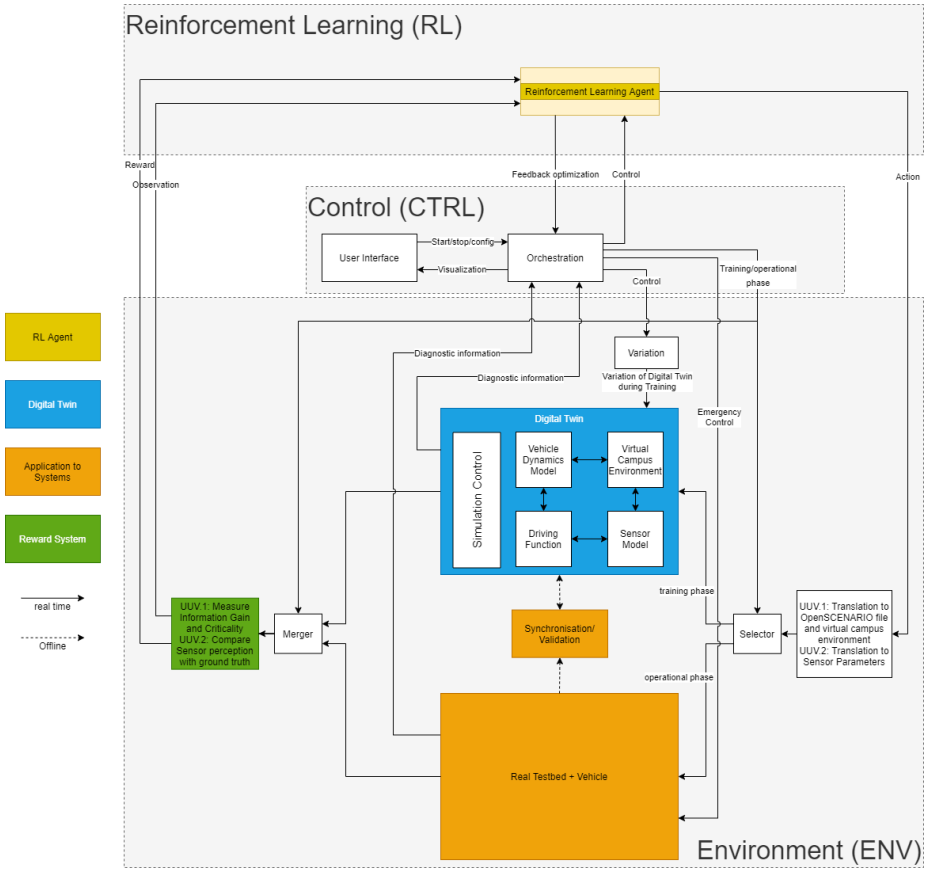


Figure 17 Architecture for the UUV use case (following the ASIMOV reference architecture). Boxes denote components, arrows denote data flow. Details of the artefacts most relevant to RL are provided in the text below.

A thorough explanation of the detailed architecture is covered in the ASIMOV Reference Architecture. In the following section, we extract only the relevant components for the environment consisting of the DT and optimization which is realized in the RL and control parts. The individual components of the detailed architecture are categorized in three different groups.

- The *Reinforcement Learning (RL)* category includes the RL agent, and it is the component where the optimization happens. The RL agent takes reward and observation as input from the Environment to generate a corresponding action, which goes back into the Environment.
- The *Control (CTRL)* category includes orchestration and user interface. To have control over the RL agent, e.g. to start, stop or track its optimization progress, it is connected to the control block. It also controls the training process, switches between operational and training mode, and receives diagnostic information. The control user interface (CTRL_UI) processes the interaction with the user, e.g. start or stop requests, or switch from training mode to operational mode. The control orchestration (CTRL_ORCH) plays a crucial role in controlling the entire process. First, it receives diagnostic information from the Digital Twin (DT) and the Cyber Physical System (CPS). This can be used to save the CPS from severe damage, when

its condition changes or unexpected behaviour is detected. If such an event occurs, the CTRL_ORCH needs to interact directly with the physical system to bring it into a safe state. In this case, it bypasses the actions provided by the RL agent via the Emergency Control arrow. User output in the form of visualized data is also a possibility. The orchestration also controls the signal flow of the RL agent together with the selector and merger blocks. Depending on the selected phase (training or operational phase), the actions proposed by the agent are either input for the CPS or for the DT (or both). Depending on the selected phase, the variation of DT is also controlled. Information about the current state of the optimization, as well as the general control of the RL agent is also processed by the orchestration.

- The *Environment (ENV) element* denotes the surroundings that the RL agent interacts with. Variation (ENV_VAR) is used to create variations in the DT during the training phase, to train a robust RL agent. In the UUV case this would reflect slightly different vehicles that could be tested. Selector (ENV_SEL) sends the signals it receives to one or multiple blocks. These can be (combinations of) the DT and the CPS depending on being in the operational or training phase. Digital Twin (ENV_DT) is the virtual representation of the CPS. Its key external interfaces, i.e., its controllable inputs and outputs, are identical to those of the CPS. The DT has additional interfaces that are used to change the inner parameters of the DT to adapt its system behaviour. Such an interface is used by the Variation during the training phase. The DT may consist of many different DT Components that interact with each other. In the UUV case this could be a vehicle dynamics model interacting with a sensor model, a model of a driving function and with the virtual campus environment. Cyber Physical System (ENV_CPS) represents the physical system. For safety purposes, it can also be controlled directly via the Emergency Control arrow, to bring it into a safe state. Synchronization/Validation (ENV_SYNC) block takes the data generated by the DT and CPS and compares these. This ensures that the DT is always up to date and that its outputs are comparable to those of the CPS. Measurement data post processing (ENV_POST) processes the outputs of the DT or the CPS so that they can serve as input for the RL Agent. The outputs of this block are observation and reward.

4.2 System Optimization Using RL Approaches

The purpose of the training phase of the ASIMOV solution is to train optimization sub-systems using the DT driven data. The trained optimization AI thereafter constructs the CPS control in the operational and fine-tunning phase. In the following, we sketch approaches from related work which serve our RL approach as adequate starting points, along with their respective advantages and limitations. Subsequently, we outline our RL approach for the UUV use case.

4.2.1 RL Approaches with Simulated Environment

A deep RL-designed magnetic controller for tokamak plasma (nuclear fusion technology) is recently introduced by [A2], which is learned by interacting with a simulated environment and subsequently applied to the physical system of the tokamak in a zero-shot fashion (see Figure 18). The approach comprises an actor-critic framework to learn appropriate voltage control commands, based on the current plasma state and control targets.

The authors propose three main phases for their zero-shot approach, i.e. (i) *objective/target design*: a designer specifies objectives for the experiment, potentially with time-varying control targets (purple boxes), (ii) *deep RL training on the simulator*: a deep RL algorithm interacts with the tokamak simulator (green box) to find a near-optimal control policy (red box) to meet the specified goals, and (iii) *application to hardware*: after training, the control policy is run directly on the tokamak HW in real time without further tuning of the weights of the control policy network (zero shot).

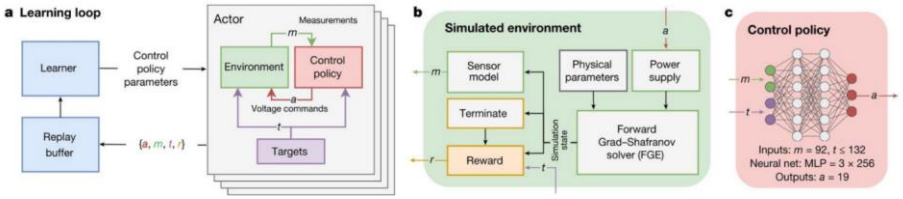


Figure 18 Representation of controller design architecture in [A2]. a) Depiction of learning loop. b) Environment interaction loop, consisting of models, parameter variation, and reward computation. c) Control policy represented by a multi-layer perceptron (MLP).

While this approach has some limitations w.r.t. applicability in the UUV use case (see below), it is still very interesting for our work. In the following, we list the *most relevant aspects* with regard to our approach:

- Using an asymmetric actor-critic framework with large recurrent critic neural network (NN) to compensate for the non-Markovian properties of the environment and relatively small feedforward actor NN;
- Learning loops for episodic RL, using a distributed architecture with a single learner instance and several actors (in [A2]: 5,000 instances) each running an independent instance of the simulator;
- Applying the Maximum a Posteriori Policy Optimization (MPO) algorithm by Abdolmaleki et al. [A3] as RL approach, and possibly combining this with relative entropy regularized policy iteration [A4]. However, as we consider a hybrid RL problem in the UUV UC, we need to adopt MPO by an extension for hybrid RL, e.g. as suggested by Neunert et al. [A5];
- Targeted parameter variation through analysis of experiment data, possibly combined with learned-region avoidance to avoid regimes where the dynamics are known to be poorly represented by the digital model through the use of rewards and termination conditions.

Limitations of the approach suggested by Degrave et al. to the UUV use case (hence to be tackled) comprise:

- Transferability of training results from virtual model to real system: In the above-outlined work, this can be assumed to be given due to the simulation model being a numerical solver for a set of partial differential equations that govern the plasma's dynamics. Here, the bridging of the 'real-sim gap' seems sufficiently accomplished without further fine-tuning after application of the trained RL controller to the real system. This, however, requires further consideration in the UUV use case.
- Fairly simple network of the control policy: The control objectives for the tokamak could apparently be reached with a fairly simple and small network architecture. In the UUV UC, the question remains whether we can achieve our system configuration/optimization objectives with a similarly simple network, or if we will need a more sophisticated architecture.
- Learned-region avoidance: In contrast to [A2], we do not have knowledge about regions where the digital model represents the PS poorly at hand. Hence, we would have to learn these and feed them back into the training process if we choose to apply learned-region avoidance. This is a task to be investigated in the course of developing the Control (CTRL) component (see Figure 17).

The deep RL algorithm QR-SAC (quantile regression soft actor-critic) is applied in [A6] to train an RL-agent that can play against the world's best human e-sports drivers and win. In order to find the solution for the multi-objective optimization problem, two different types of rewards are defined to give quick positive feedback to staying on track and driving fast, and to make the agent win the race instead of just driving on the track. The latter reward aiming to make the agent overtake others is calculated as proportional to the improved distance relative to the opponents. Furthermore, using a mixed scenario training with noisy selections of the critical situations provided the possibility to let the agent learn rare skills.

The application of RL for the UUV use case requires dealing with both *discrete and continuous state spaces and action spaces (hybrid RL)*. While there are a large number of state-of-the-art RL approaches to handle either discrete or continuous spaces, only very few of them can handle both simultaneously, e.g., the policy gradient methods. One solution is to homogenize the spaces and transform them in order to have a single paradigm; either by discretizing the continuous variables, or by approximating the discrete ones with continuous variables/probability distributions. The main drawback of such approaches is the limitation caused by changing the problem structure on the effectiveness and accuracy of applicable solution methods. For the UUV use case converting the discrete actions to continuous would rather promote more (unnecessary) complications, while discretizing the continuous actions would produce a huge action space.

Another approach to address the tasks with hybrid space is to split the actions among continuous and discrete components and train two different RL agents. Parameterized Action Space Markov Decision Processes (PAMDP) [A7] is one of the subcategories in hybrid RL that look at hybrid tasks as a hierarchical problem. The RL agent starts with selecting an action from one of the paradigms, either discrete or continuous, and continues with the other paradigm for the next step. For instance, [A7] first selects an action from a discrete set and goes through the continuous space to select a continuous set of parameters for that action. A different approach is taken by [A8] where the RL agent selects continuous actions and weights them by the discrete choices.

In order to address the task with hybrid dynamics and action space in their native form, a hybrid approach is proposed by Neunert et al. [A5]. They developed a model-free algorithm that optimizes for discrete and continuous actions simultaneously, by using a hybrid parametric policy π_θ, which is a state dependent distribution that jointly models discrete and continuous random variables. The hybrid policy optimization is based on the state-of-the-art Maximum a Posteriori Policy Optimization (MPO) algorithm, introduced by [A3] and [A4]. MPO is a two-step approach for off-policy policy optimization. In the first step, an approximation to the Q-function is learned from experience (*policy evaluation*). This is done by minimizing a squared loss which is dependent on a behavioral policy $b(s,a)$ (off-policy). Here, *b(s,a)* corresponds to the action probabilities of the actions carried out at that point in time and stored in the replay buffer. The second step (*policy update*), in turn, is divided into two sub-steps (E-step: estimation, M-step: maximization). (i) *E-step*: first, a non-parametric improved policy q is constructed and updated in an iteration procedure by maximizing the approximated Q-function for the states from a replay buffer, while ensuring to stay close to the current policy. The replay buffer stores the transitions and log probabilities of actions that resulted from interacting with the environment through the sampled discrete and continuous actions. (ii) *M-step:* The second step is to fit a parametric policy to the improved policy calculated in the first step using supervised learning. The fitting of the next policy parameter θ_{i+1} is done by solving a weighted maximum likelihood problem while requiring the policy change from one iteration to the next is constrained. In this step, the hybrid policy is optimized in a decoupled form for its discrete and continuous parts separately.

The adaptation of hybrid policy optimization using the actor-critic framework for the UUV use case is depicted in Figure 19.

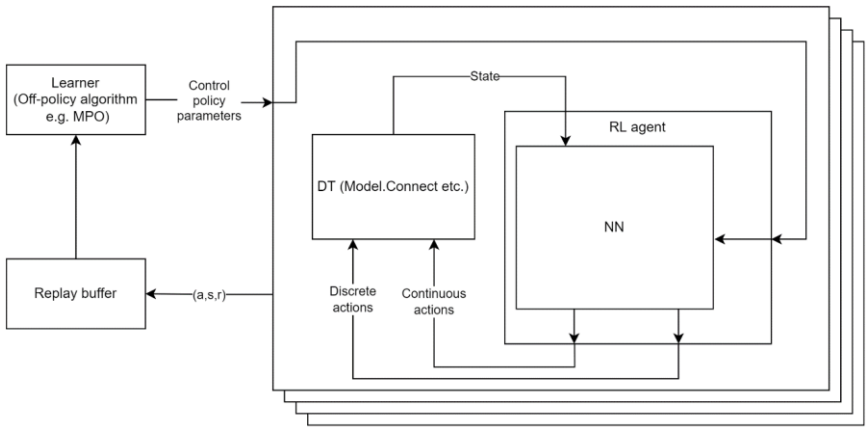


Figure 19 Architecture for the hybrid RL approach (actor-critic) in the UUV use case, inspired by [A2] and [A5]. Boxes denote components, arrows depict data flow. The layers on the right represent a set of distributed actor instances.

In each iteration step, the actor instances (stack of layers) comprising the RL agent gather experience in episodes of a fixed length (number of steps) through interacting with the respectively instantiated version of the environment (= DT; see Section 4.2.2) while following the current policy π_{θ_i} constructed by the learner. At each layer, the discrete and continuous actions, i.e., scenario parameters, are fed into the DT simultaneously by the RL agent and based on the current state of the DT. The experience data (trajectories of selected actions and their log probabilities, and transitions, i.e., successor states and obtained rewards) from all actor instances are stored into the replay buffer and fed to the learner. Then, after some stopping criterion is fulfilled, the learner updates the policy based on the gathered experience data according to the above-described procedure.
In the *distributed actor framework* ACME (see [A9]) also applied by Degrave et al. [A2], it is designated that the actors gain experience by interacting with their own instance of the environment (=DT) in parallel and pull policy parameter updates from the learner asynchronously to accelerate the data generation process. In addition, this distributed architecture allows for the learning process to proceed as quickly as possible regardless of the speed of data gathering. However, to start in a simple, less error-prone way, we aim to set up the learning architecture first with a single actor instance and consider extending this to a distributed actor setup at a later point in time.

4.2.2 Bridging the Gap Between Synthetic and Real Data

First, we assume that in UUV.1 sub-UC the DT delivers quite valid outputs (state, reward) compared to the physical system. Hence, for the first iteration of the use case we propose to *combine zero-shot approach and variation during training* as described in the following.
The zero-shot approach trains the RL agent on the DT only, and applies the learned policy to the physical system without a feedback loop to the RL agent (as in [A2]). This approach requires (i) a sufficiently valid DT w.r.t. interaction with the RL agent, that is, for the same test case parameters, we obtain very similar states [cumulated anomaly values, cumulated criticality values, basic vehicle configurations] and rewards [weighted sum of measured criticality/anomaly values], (ii) very close input distributions to the DT and physical system.
In the UUV UC, we may overcome possible deviations between simulated and real system behaviour by varying across many vehicle parameters during training to widen the range of examples the agent has seen. Some DT components of the UUV are already in good shape, regarding the alignment between simulated and measured data. However, as there is always some kind of mismatch between these two, it is important that the agent can bridge the gap when confronted with real data and trained with simulated data. To achieve this robustness, we plan to vary some aspects of the vehicle's responses during the training phase by changing the configuration parameters from episode to episode. These could be

parameters that define the driving function, some physical parameters like weight and power, which influence the behaviour of the vehicle dynamics and, finally, parameters that affect the virtual sensor model and therefore lead to varying perception results.
If we can ensure that the range of variation during the training phase is larger than the uncertainty of the model of the vehicle to be tested, the RL agent should be able to cope with the resulting differences in behaviour.
The variation will be subtle and will be applied between episodes in training. No variation will be done during the episode itself, as the results of scenario simulations with the same vehicle parameters shall lead to consistent results. Furthermore, the agent's actions depend on the state defined by their predecessor action.
With this kind of variation, the training phase can be adapted to the knowledge-level of the vehicle under test. A well-known vehicle with multiple parameters being certain, the parameters to be varied are fewer compared to a lesser-known vehicle, which has more uncertainties. Expert knowledge in the form of typical parameter ranges for certain vehicle parameters can also be incorporated into the training process by restricting the variation to these ranges. In v*ariation distribution:* We first vary the parameters purely randomly, that is, choosing one single variant of the DT drawn from some given probability distribution (e.g., uniformly on the set of feasible DT parameters), running one single episode with that variant and re-setting/changing this for every new episode. Later, a more targeted variation adapting the RL agent results might be considered.
If the DT-RL stack does not deliver realistic and near-optimal scenario test sets (e.g., only uncritical scenarios and/or with low pairwise dissimilarity), we have to take further approaches into account, such as Transfer learning/domain adaptation approaches: (i) Inductive transfer learning: Reasoning from observed training cases to general rules, which are then applied to the test cases (traditional supervised learning). However, in sub-UC UUV.1 we do not have labelled data/ground truth data at hand to compare with. Applying transfer learning would need some sort of labelled data which we would first have to produce by running a set of certain training scenarios (to be defined) with the physical system, i.e., with the real UUV in the test bed. (ii) Transductive transfer learning: Reasoning from observed, specific training cases to specific test cases (see e.g. [Af10]).
The transfer learning in RL concerns speeding up the learning process [A11], e.g., to avoid letting the RL agent spend many episodes before reaching a reasonable Q-function. A number of promoting transfer learning methods to the UUV UC are listed here. Within all approaches, the main task addresses the optimization of the CPS, and the so-called source task corresponds to the optimization of the DT, the individual components that are interacting inside it, and different variations of the DT. Depending on the DT and the RL agent that are developed for the UUV.1 use case, we can use and adapt these methods in further stages, for instance for the UUV.2 use case which is subsequent to the results from UUV.1. The other potential application is to first develop the RL agent to optimize the scenarios with respect to challenging the UUV with safety, and further transfer the learning to develop the scenarios for the comfort objectives.
The s*tarting point methods* set the learned knowledge from a source task in form of initial solution into the main task. The starting-point methods can begin the RL process at a point close to a (good) solution. In general, the RL algorithm in the main task is not changed. They require a mapping of features and actions between the tasks (see for example [A12] for a brief introduction).
The *hierarchical methods* view the source task as a subtask of the main task, and use the solution as a building block in learning. One approach is to use the temporal concatenations of several tasks with less complexity and compose the main task solution by combining them. The option framework [A13], which includes temporally extended or abstracted actions, can also be adapted to a hierarchical method. In this way, the entire or part of the learned policy from the first task is introduced as an option to the main task (see [A14]).
Alteration methods make changes in the RL framework of the main task according to the existing source task, e.g. by altering state or action spaces or reward function. The option-based framework [A13] provides an approach to change the action space by involving temporal abstraction or temporal extended actions. The option promotes the advantage of the simplicities and efficiencies sometimes available at higher levels of temporal abstraction.

- Altering the state space can happen by two means in the main task; state abstraction such to have comparable set of states with the first task, and state expansion by adding new states (see [A15], [A16]).
- Minifying the action space is another approach to decrease the complexity of the value function.

- Reward shaping is an example of altering for the reward methods. The RL agent learns to augment its reward structures by learning what sensory patterns predict reward across episodes or tasks (see [A17]). Reward shaping is a design technique in RL that speeds up the learning by finding immediate rewards that are more indicative of cumulative rewards. In automatic reward shaping, the agent learns to predict rewards and use them to create a shaped reward function in the unseen task. The agent uses the information as a shaping function that provides a first estimate for the value of newly discovered states when learning a value function for a new task. The agent experiences a sequence of environments generated by the same underlying generative environment model.

Additionally, a temporal-difference algorithm in which value functions are influenced by observations of expert agents is proposed by [A18]. The algorithm uses a variant of the value-function update that includes an expert's experience, weighted by the agent's confidence in itself and in the expert. A knowledge-based kernel regression (KBKR) that allows transfer via advice-taking, is introduced by [A19]. The "advice" in this algorithm is a rule telling the agent which action to prefer in a set of states described by a conjunct of predicates. KBKR approximates the Q-function with a support vector machine and includes advice as a soft constraint. The relearned Q-function in batches using temporal-difference updates trades off between matching the agent's experience and matching the advice.

4.3 Application and Integration

4.3.1 Integration and verification

The above-mentioned algorithm has been implemented into the toolchain and tested. A step-by-step implementation of MPO can be found in D1.3.1. In short, as shown in the diagram 20, the training gets initiated, and first trajectories/episodes are sampled. Each step in the episode corresponds to a loop in the toolchain. If a predetermined number of episodes has been gathered and stored in an experience buffer, then the critic and actor are updated using samples from that buffer. Samples can be re-used multiple times. However, over-fitting can become an issue here. In continuation, the sampling and updating is repeated.

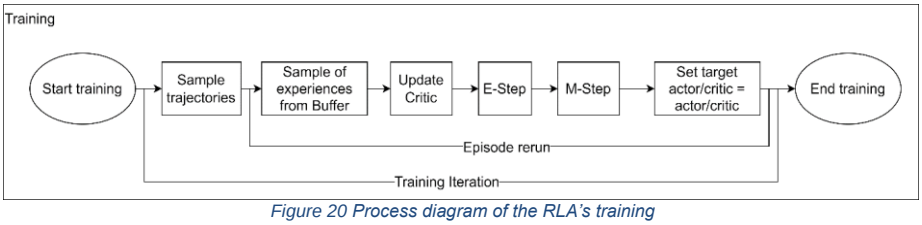


Figure 20 Process diagram of the RLA's training

To ensure the algorithm's accurate performance and isolate any errors caused by the environment (e.g., other components in the pipeline), rather than from the RL algorithm's implementation, it's essential to first conduct tests during early development in a controlled and well-understood setting. This is especially important as the training environment in this use case is extensive and feedback data is expensive. To achieve this, we employed the standard [A20] to validate the training process. However, since this library doesn't offer an environment capable of handling both discrete and continuous actions simultaneously as required by the use case, we utilized a superposition of a discrete and continuous environment as it can be seen in Figure 21. Specifically, we employed the "LunarLander" environment, which has both discrete and continuous action versions.

In this hybrid setup, we concatenated observations, while actions were separated based on their affiliation to the discrete or continuous environment. The reward from the two environments was added together. Initially, we used default values as a baseline. Nevertheless, we also explored variations in the training process and adjustments to the actor's neural network.

Commented [v(66]: Do I understand correctly that you're referring to potential bugs in other components of the pipeline?

Commented [M(67R66]: You are right... I rephrased it. Is it clearer now?

Commented [v(68R66]: Yes!

Commented [v(69]: Might be nice to include something like a figure here to illustrate this concept

Commented [M(70R69]: Please review =)

Commented [v(71R69]: Very nice!

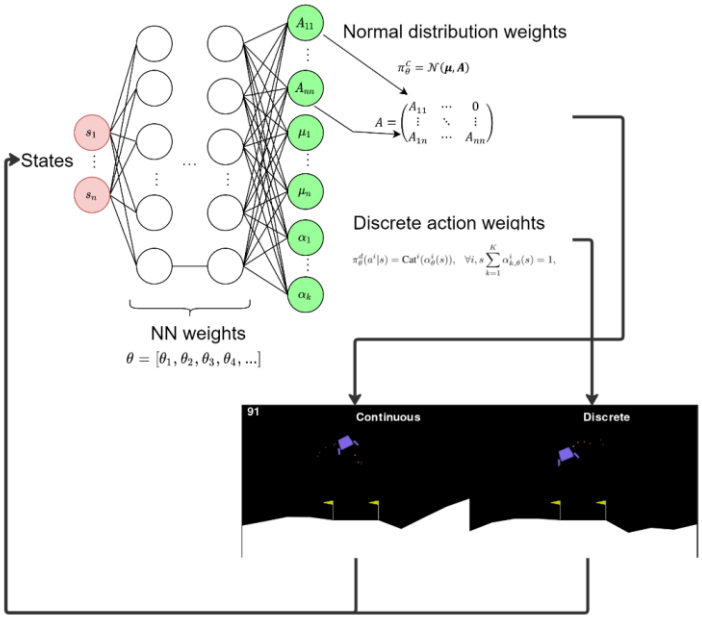


Figure 21 Visualization of the adapted LunarLander gym environment for early algorithm verification and the actor network controlling the environments as well as receiving the environment's state.

The tests encompassed 20,000 training iterations and the algorithm appeared to converge to the maximum achievable reward after 1,000 runs. Thus, learning of the algorithm's implementation could be verified. As shown in Figure 22, we repeated this process with diverse training strategies and actor modifications. The outcomes of these training variations demonstrated that maximizing the reuse of sampled data is highly effective, while expanding the size of the actor's neural network in this context proves unnecessary and detrimental to the training process.

Commented [v(72]: Can we create a figure to summarize the results?

Commented [v(73R72]: How does the hybrid setup compare to the case where we discretize the continuous action space or to the case where we assume continuous action spaces and then round off to discrete steps?

Commented [M(74R72]: Figure added.

The question is interesting though. Might need to test this in later
Discretization of the continous action space might require too many output nodes on the actor. But the other way around could be a good trade-off.

Figure 22 Results over the first 1000 training runs for different network configurations.

Interestingly, the resulting policy of the lunar lander was to hover over the landing site in both environments as this appeared to be the best option due to the rudimentary reward design. The suspected cause of this behavior is that while one environment could achieve a successful landing, if the other lunar lander would fail, the reward would still be negative even though in the first environment achieved a successful landing.
To be capable of easily switching between the use case and the gym environment and potentially other applications we re-used the architecture of the gym's environment class as a wrapper for a "standardized" interface between reinforcement learning agent and the tool chain. This also enables the usage of off-the-shelf agents using different algorithms which often are implemented to work with the gym package as a running example. Here, we have the standard interaction functions such as reset and step. In our case, the RLA only interacts with that wrapper's methods using vectors for action, state and reward, the wrapper writes it into files and triggers the other docker containers of the tool chain.

4.3.2 Experiment tracking

The next application topic is the assessment of learning tracking software, more specifically [A21]. MLflow is an open-source platform that claims to streamline the end-to-end machine learning lifecycle. It offers tools for experiment tracking, project packaging, model management, and deployment. Data scientists and machine learning engineers use MLflow to log and compare experiments, package code into reusable projects, manage machine learning models, and transition from development to deployment. Of interest in the use case is the capability of managing experiments (i.e., training runs) and attach the resulting actor and critic models to it. Its server-client-structure enables tracking across machines including remote hosting of databases.
Implementation-wise, a new experiment is initialized at the tracking server by posting training parameters during at the start of each training, as shown in Figure 23. Each experiment possesses a unique id which is attached to posts of values of interest (loss, return, etc.) to the tracking server during the training. The tracking server stores the parameter and values in SQL storage as well as files in an artifact store if desired. Examples of files of interest would be the weights of the trained NNs. In the UUV UC, the storage databases are in the cloud for better sharing of experimental results between consortium partners.

Commented [M(75]: Reference:

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

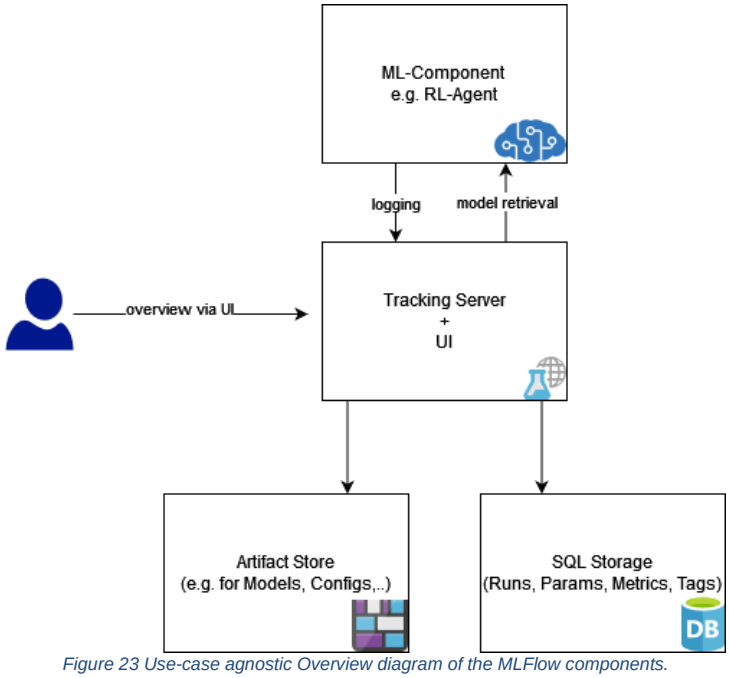


Figure 23 Use-case agnostic Overview diagram of the MLFlow components.

4.3.3 State design

The process of post-processing in the UUV.1 use case involves two key components: anomaly detection and criticality metrics. An autoencoder neural network is employed for anomaly detection, measuring information gain in simulated scenarios. The autoencoder's goal is to reconstruct input data with minimal error, and it achieves this by compressing the data and extracting relevant features. The compression results in a loss of information, making it suitable for detecting anomalies in data it has encountered before (see e.g., Chen et al. 2018 [A22]). High reconstruction error signifies high information value and contributes to the reward function. To enhance the network's performance, it is periodically retrained with new data which is then considered not new anymore. However, a challenge was identified because the anomaly detection process depends on previously seen states which are embedded into the weights of the auto encoder's network. This situation would violate the Markov property, creating difficulties in reinforcement learning. A solution would be to account for this in the state by keeping track of all of the previous datapoints it has encountered so far. As a result, a significant state vector to store the history of previously evaluated scenarios would be required, which is not feasible due to the already extensive state space. For now, the anomaly detection component has been removed from the reinforcement learning loop. Instead, it is employed as a standalone tool for data analysis, not directly influencing the reward calculation. This decision helps maintain the Markov property in the reinforcement learning process.
To solve this issue, it could be tested whether techniques as seen in large-language models can be used. These models are capable of keeping track of long conversations. A suggested scenario can be seen analogously to an entry to the conversation. However, this will not be pursued within the ASIMOV project.

Commented [H(76]: Z. Chen, C. K. Yeo, B. S. Lee and C. T. Lau, "Autoencoder-based network anomaly detection," *2018 Wireless Telecommunications Symposium (WTS)*, Phoenix, AZ, USA, 2018, pp. 1-5, doi: 10.1109/WTS.2018.8363930.

Commented [v(77]: Is there a source for this? Has this technique been proposed/used by others?

Commented [H(78R77]: I think this was written by Niklas but we added a relevant reference.

4.4 Conclusions

Commented [H(79]: @Mehrnoush write the summary and existing limitation before 22.02.

The UUV.1 use case involves the automatic adaptation of traffic scenarios in a 3D environment based on vehicle interaction. The aim is to create critical and novel scenarios for automated test plans that are tailored to individual characteristics of the tested vehicle.
The selected RL framework for this use case is a model-free approach that works episodically. The algorithm keeps the vehicle characteristics consistent within the episodes, but modifies them slightly between episodes to develop a robust RL agent. This is done within the vehicle's ODD by using a model-free approach that does not require knowledge of the state transition probabilities. Due to the mixture of discrete and continuous state and action spaces, hybrid RL is required for this use case.
To this end, we have applied a hybrid approach that optimises for discrete and continuous actions simultaneously using a hybrid parametric policy that models both types of random variables together. This hybrid policy optimisation is based on MPO algorithm, a two-stage approach for optimising off-policy policies. The algorithm was implemented in the toolchain, where the implementation process includes the initiation of training, the sampling of trajectories/episodes, the updating of the critic and actor using samples from an experience buffer.

5 Scanning Transmission Electron Microscopy Use Case

5.1 General introduction

(S)TEM plays a vital role in many fields from material sciences, physics, chemistry to biology. It can visualize nearly everything ranging from micro-meter to angstrom scale (atomic resolution). We will briefly summarize some technical challenges to imaging with electrons, in particular the optics, as this is the ASIMOV focus [B1, B2, B3].

Electrons may be described by the wave-particle duality. Unlike for example, light, the wavelength of an electron is a function of its speed. Increasing the accelerator voltage of a TEM microscope will decrease the wavelength of the electrons and thus increase the resolving power. As they are charged, electrons can be focused by electromagnetic lenses. Images are formed by propagating a beam on a sample. The waves will interact with the sample via scattering, which is sample specific. Afterwards a series of lenses form the final image.

Electrons need to be spatially and temporally coherent. The first means that all electrons need to come from the same direction as they hit the sample. If this is not the case, the resulting image will blur. Secondly, all electrons need to have the same energy. If not, the lens corrections will differ. In short, electrons need to have the same wavelength and phase, and originate from a single spot.

Broadly speaking electromagnetic lenses are affected by three types of aberrations. Spherical aberration, chromatic aberration and astigmatism. Spherical aberrations are caused by the fact that an electron going through the centre of an electromagnetic field of a lens, will be subjected to a weaker force, than one passing closer to the coils. Chromatic aberration is caused by temporal incoherence. High energy electrons are influenced less by lens current than their slower counterparts. Lastly, astigmatism is caused by a lens not being equal in strength over the x and y axis.

5.2 Ronchigrams

A ronchigram is the diffraction pattern of a convergent beam that is focused on an amorphous sample. Diffraction refers to all phenomena that occur when a wave hits an obstacle or an opening. This opening, or aperture, effectively becomes a second source of the propagating wave.

Imagine an optical lens focusing light with a convergent beam. It will appear as a uniform bright disc. If you however insert a grating pattern with light and dark stripes spaced about 100 times the length of the used wavelength, the resulting interference patterns will contain clues on the imperfections of the used lens. [B3]

In similar jest, an amorphous sample has an atomic structure with a random assortment of potentials. It thus serves as a noisy random grating for the electron beam of a microscope. And just as its optical cousin, the resulting image with interference patterns gives clues to the present lens aberrations.[B3]

Examples of ronchigram images from a microscope are shown in Figure 40 and 41. Each row has different lens aberrations, whereas the columns show the effect of moving the beam from under focus, to in focus and finally over focus.

Commented [K(80)]: References to all figures appear broken. I assume here you mean the figure in page 58

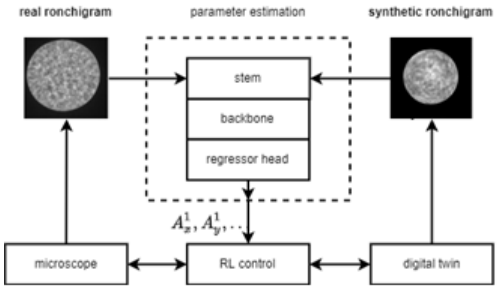


Figure 24 Initial TFS ASIMOV architecture.

5.3 General Architecture

As shown in Figure 24, the general ASIMOV TFS goal is as follows. We wish to deduce lens aberration parameters from Ronchigram images and deploy an agent that not only interprets these images, but also learns how to calibrate the microscope, much like a human operator. To train all these components, TFS

will develop a digital twin of the microscope, which will serve as the environment in which agents are trained prior to deployment to the real machine.

5.4 Image based inference

There are several necessary basic steppingstones that need be researched and built which we will address first. As imaging is such a huge component of the agent, we will address feature learning separately. In particular:

- Base parameter inference on simulated data
- Base parameter inference on real data
- Challenges in bridging the gap between real and synthetic data

5.4.1 Base inference

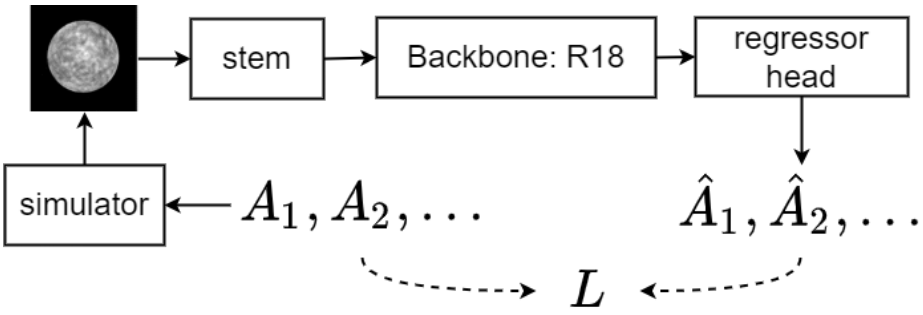


Figure 25 Base supervised parameter inference for Ronchigram images.

The current digital twin is an inhouse-developed Ronchigram simulator. Examples can be seen in Figure 29. A basic requirement that needs to be met, is answering the question can lens aberration parameters be learned from Ronchigram images in a supervised manor.
A basic supervised architecture, seen in Figures 24 and 25 was developed. It consists of a stem, backbone and regressor head. The stem is used to multiply the number of channels from 1 to 3. The backbone can be anything, but in practice is Resnet18 [B4]. Lastly the regressor head is a stack of Affine layers without any output layers.
There are several interesting differences from traditional computer vision training such as seen with imagenet classification [B5]:

- As the frequency content of the Ronchigram contains vital clues, standard data augmentations such as cropping or scaling should not be used. The same applies to horizontal and vertical flipping. Changing the frequency content pushes images to a different labels space.
- Any type of activation in the regressor head destroys performance.
- Training was significantly more sensitive to hyper-parameter tuning. In particular to the learning rate schedular of the optimizer.
- The loss function was a scaled combination of the L1 and L2 loss. The first is important because we need the network to be able to learn that some regressors might be (near) zero.
- There was no benefit from pre-training.
- There was no benefit from using larger networks, such as R50.

5.4.2 Results

Figure 28 shows the validation loss on our synthetic and real datasets, whereas Figure 26 shows the spread of the estimated parameters against the ground truth. As expected, the network struggles more with learning higher order aberrations
Figure 27 shows that the network managed to estimate the parameters up to a combined L1-L2 loss of 0.02nm. The problem with this result is two-fold.
Firstly, this performance surpasses human ability. This means the network is picking up high (frequency) detail to learn from.
Secondly, the validation data is also provided by the simulator. Therefor it has the exact same (though unknown) distribution as the training-set. Consequently, this network may be overtrained without the validation error going up. In essence, the network has learned to invert the simulator. And this is a problem, as the goal of the digital twin is to allow training systems that generalize to real data.
The latter is further illustrated by Figure 28. This shows training and validation where the initial data used is synthetic before switching over to real data. The sudden jump in loss shows there is no benefit to (pre) training the network on synthetic data, prior to switching to real data for fine-tuning.
In conclusion, we need an architecture that can learn domain invariant features, that are only indicative of present lens aberrations.

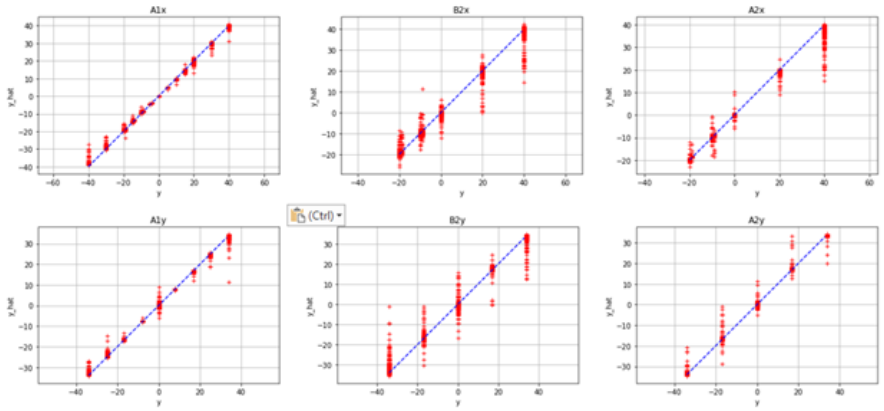


Figure 26 Parameter inference for synthetic data. Shown are the estimated aberrations versus the ground truth. As expected, the network has more difficulty estimating higher-order parameters, resulting in a larger spread.

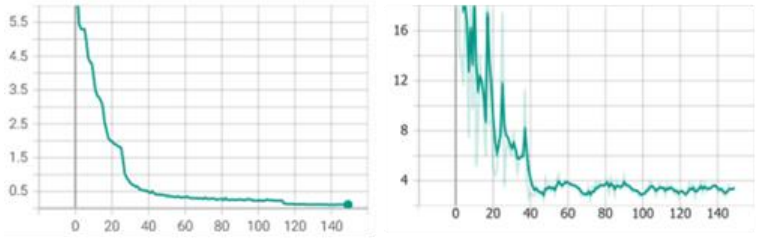


Figure 27 L1-L2 validation loss for synthetic data (left) and real data (right)

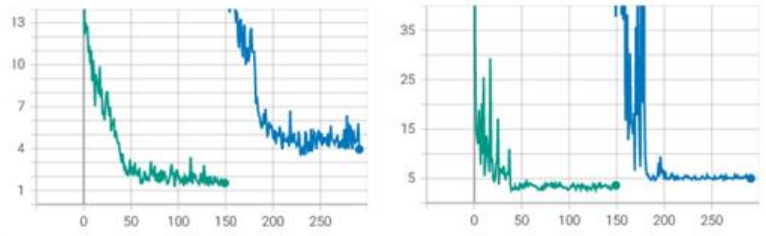


Figure 28 Training (left) and validation (right) on synthetic data, followed by real data. The jump in the loss curves, the moment the data source changes, shows there is no benefit to pre-training a network on synthetic data, prior to learning on real data.

5.5 Domain mismatch

As illustrated in Figure 29, there is a visual mis-match between the images originating from the simulator and those from a real microscope. Short of development of a new simulator, we will explore two data driven methods in this chapter to close this gap. Self Supervised Learning (SSL) and Domain adaptation (DA).

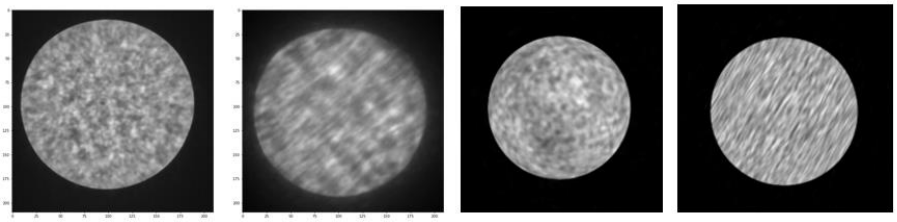


Figure 29 Domain mismatch example. The two left images are real Ronchigrams, the others are synthetic.

Commented [K(81]: mismatch

5.5.1 Self (Semi) Supervised Learning

Given a task and enough high-quality data, supervised learning can achieve fantastic results. Collecting labels, however, is both expensive and error prone. SSL aims to learn without labels by formulating a so-called pretext task. This task or learning objective is defined based on the data itself, thus omitting the need for labels. The pretext task is not the final goal. Rather, ones hopes that it will allow a network to learn representations that will be useful for other final downstream tasks.

A simple example would be to take an image, and a rotated copy, and have the network learn that these are in fact the same image by predicting the rotation [B6]. This image does not need a label. Furthermore, we are not interested in the actual rotation, but hope that the network learns a useful latent representation that may be deployed elsewhere. Other approaches extracted multiple patches in random order from an image, and then let the model infer their relative positions [B7].

A subset of SSL is formed by contrastive representation learning. These methods aim to learn some embedding space in which similar pairs of images are close, and dissimilar ones are far away. Early examples of this approach are contrastive [B10] and triplet loss. [B11] These methods have several similar components. Firstly, they need a notion of matching or positive pairs, and non-matching or negative pairs. Most rely on heavy data augmentation to create noisy versions of a data sample for the formation of the positive pairs. Secondly, most algorithms need huge batch sizes to function. This is needed to give the loss function enough diverse (negative) examples to learn from. Lastly, hard negative mining is used. This means that one explicitly collects negative pairs that erroneously have close embedding features to force the network to learn. [B11, B12]

5.5.2 SSL in the ASIMOV context

We seek (learned) image features, from Ronchigrams, that are indicative of any lens aberration present in the image, irrespective of the image source. Be it from the digital twin, or an actual microscope. As such we are investigating and modifying two contrastive SSL methods: SimCLR [B13] and BYOL. [B14]

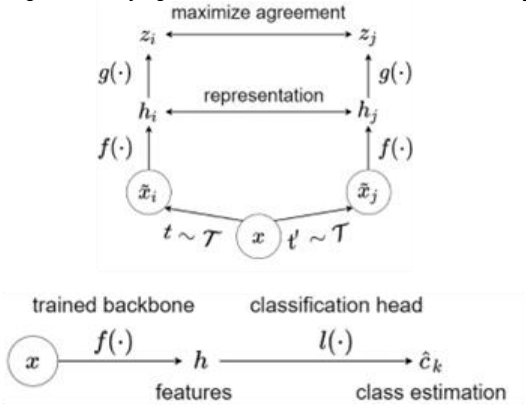


Figure 30 The original SIMCLR [B13] backbone training and supervised fine-tuning.

Figure 30 shows the original SimCLR architecture for the representation learning phase using the pre-text task, and the final supervised stage. The backbone *f* is trained by presenting the network with positive image pairs, consisting of an image and a heavily modified copy, next to negative pairs, formed by different images al together. BYOL not only surpassed SimCLR in performance, but also does not need contrastive learning: there is no need for negative pairs.
As shown in Figure 31 we are adapting these architectures in a number of ways:

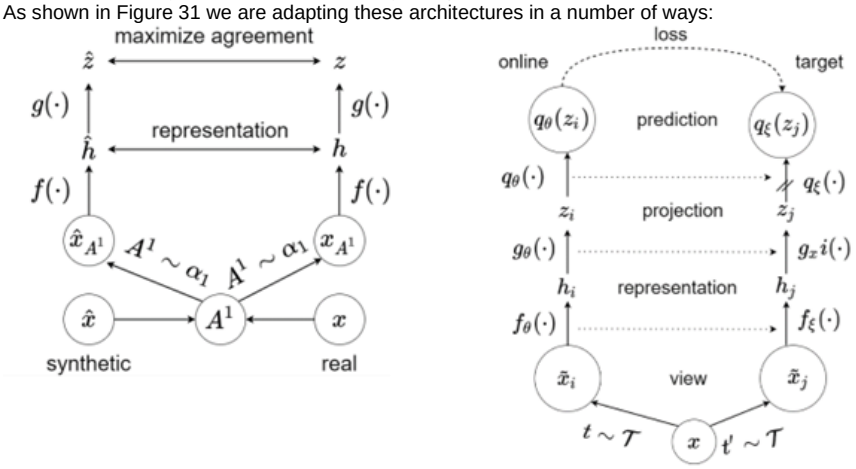


Figure 31 Adapted semi supervised learning architectures. SIMCLR [B13] and BYOL [B14].

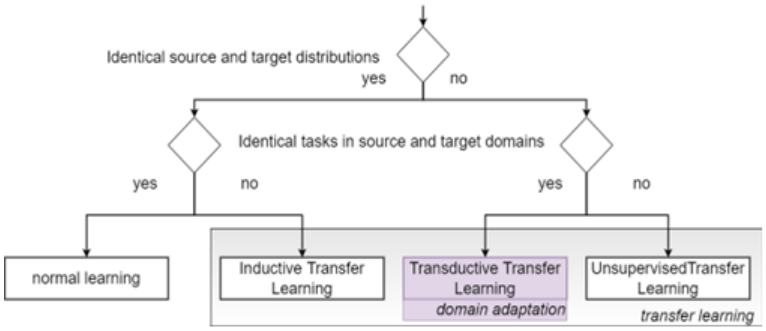


Figure 32 Difference between transfer learning and domain adaptation. Adapted from [B16].

We use a SimCLR architecture where similar and dissimilar synthetic ronchigram images are fed through the network. The mayor challenge is to find data augmentation techniques that will force the network to learn representations that will prove useful for the ultimate downstream task: predicting the aberrations. Experiments so far have shown that taking the augmentations as used for example on the CIFAR10 dataset, do not work on Ronchigram images.
We are training SimCLR/BYOL derived architectures, where the positive pairs are formed by a synthetic and a real Ronchigram images, but with identical aberrations. The negative pairs are formed by images that differ in their aberrations. The two major challenges with this approach are the fact that the aberration parameters are not an exact match. For the simulator they are known, whereas for the real images they need to be estimated. And since this approach requires the labels, it is in fact, fully supervised.
Other avenues that are currently underway are modifying the backbones to use ADA-in [B16] or to combine batch-normalization (BN) with instance-normalization (IN). Again, the goal is to eliminate appearance variance, while maintaining feature discrimination. Informally, BN preserves discrimination between samples, whereas IN eliminates individual contrast, yet diminishes useful information at the same time.

5.6 Domain Adaptation

Domain adaptation refers to the family of methods where the training data comes from a similar but different distribution than the test data [B15]. Figure 32 showcases the difference between transfer learning and domain adaptation. There are various flavours of this idea, but in the ASIMOV context we are focusing on the situation in which the labels for the source domain are available and those from the target domain are not. This matches our situation in which the source domain is formed by data generated from the digital twin, and thus the labels are known, and the target domain is formed by real microscope data with unknown aberrations.

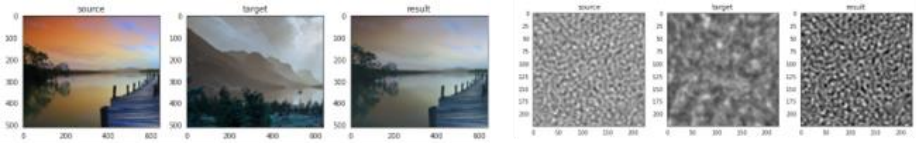


Figure 33 Optimal transport example for remapping (colour) values in images. Adapted from [B21].

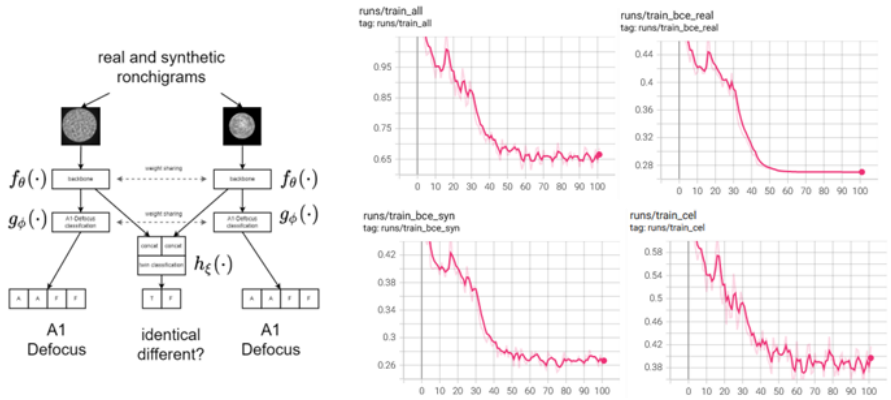


Figure 34 Initial domain adaption architecture next to the training and validation results.

An initial architecture where both source and target labels are available is shown in Figure 34. This network is tasked with classifying the A1 aberration parameter next to the defocus label. It has an extra head that needs to find out if the image pair it was fed, came from an identical domain (be it synthetic or real) or from different domains. These gradients are fed back to the network, thus forcing it to find features that are domain invariant. There are two disadvantages to this approach. Firstly, it still requires the source–and the target dataset at the same time. The added head to discover if the domain matches between the pairs is regrettably not the vital component.

Commented [K(82]: It requires the source and target dataset (ie synthetic and real) to be available at the same time?

Commented [K(83R82]: Also it is not clear what the second disadvantage is

Commented [D(84R82]: yes, and clarified

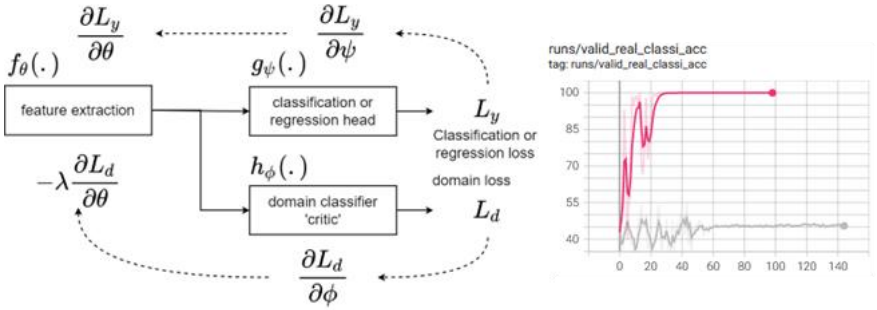


Figure 35 Left: domain adversarial training of neural networks [B14]. The source data is the labelled synthetic data. The target data is the real data. On the right the classification accuracy is shown using the target data labels (red) and without any labels (grey). Classification performance drops to 40 percent, showing that bridging the gap between labelled synthetic data and unlabeled real data remains a significant challenge.

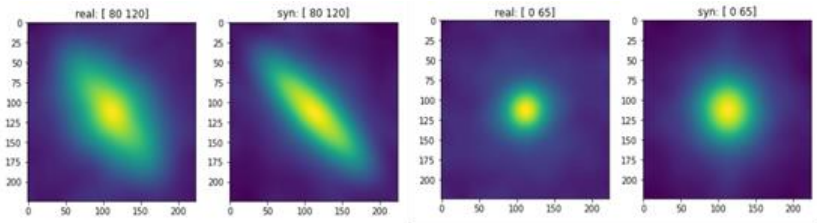


Figure 36 Real and synthetic windowed low-pass-filtered power spectra from real and synthetic images with identical lens aberrations.

Currently, the following methods are under consideration for the ASIMOV applications. Domain-adversarial neural networks (DANN) [B14]. Its architecture and initial results are showcased in Figure 35. The main idea driving this architecture, is the fact that the classification head should not be able to distinguish if the features from the backbone originate from the source or target distribution. The features should only carry information of their class label, not their originating domain. To achieve this, an adversarial component is added to the network. This adversarial net tries to estimate the originating domain from the features. If it achieves this task, it punishes the backbone via an inverted gradient. Work from [B19] (CDAN) expanded this approach. The authors claim that methods such as [B14] are prone to under-matching and propose further conditioning of the adversarial component by forcing alignment of the multimodal distributions. They observe that when the joint distributions of the features and classes are non-identical across domains, adapting only the feature distribution might be insufficient. More so when this feature representation is also multi-modal. The authors propose to condition the generative component with the available label information, next to deploying a multilinear map [B20] based on the cross product from embedded feature and class label vectors.
A third family of potential useful methods is based on optimal transport using the Wasserstein distance [B21]. Optimal transport can itself be used to match the RGB or grayscale histograms between two dissimilar images, independent of the semantic content. An example of this is shown in Figure 33. As a pre-processing step, however, we did not find this helpful. In [B22] the Wasserstein distance is used in the domain critic network to distinguish between features originating from different domains after passing through the backbone. They claim superior results over DANN [B14], using the Maximum Mean Discrepancy (MMD) [B23] deployed in [B25, B26] and the so-called metric or deep correlation aligment (CORAL) [B24].
To test viability, a proof-of-concept implementation was built, based on DANN [B14]. Its architecture and initial results are shown in Figure 35. When labels are available for both domains, performance is excellent. However, the ultimate ASIMOV goal is to only use labels from the digital twin. In this case, our networks performance drops to 48% classification performance over 8 classes. Given the significant difference between our current digital twin and the actual microscope, this result is quite promising, though insufficient at this time. Future work will explore and implement the ideas of the here mentioned literature.

5.7 Intermediate Conclusions and Future directions

We aim to distil visual clues from Ronchigram images to estimate lens aberrations. Currently this is perfectly possible for both the synthetic data from our digital twin and for real microscope data. This section focused on current and future endeavours to learn a feature that may be used across domains without using any real microscope data that is labelled. In the next section, we will divert to a hand-crafted feature that generalized to show-case and investigate the design of a digital twin trained reinforcement agent.

5.8 Reinforcement learning

5.8.1 Introduction

This section will introduce the current proof-of-concept of a digital twin trained agent. Firstly, we will introduce the hand-crafted feature the agent uses from the digital twin. Secondly, we will survey literature in order to gather the necessary components needed to build upon this initial prototype.

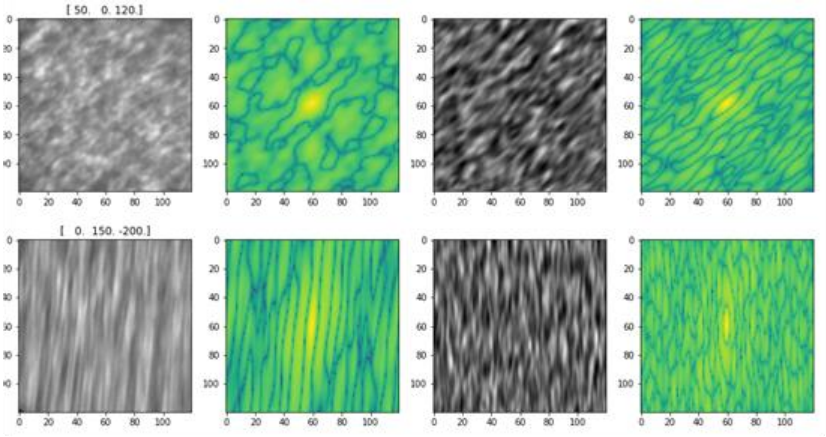


Figure 37 Real and synthetic images with identical lens aberrations shown next to their auto-correlation function image (ACF).

5.8.2 Hand crafting a feature

In order to train an agent on synthetic data from a digital twin and then deploy that agent on an actual microscope, we need a feature that is domain invariant yet contains sufficient information to be able to infer lens aberrations.
Traditional computer vision features are all to a certain extend depended on chosen parameters and therefore not completely invariant to scale and lighting changes. We therefore looked at basic signal processing methods that use all available information and do not require a priory set parameters. We investigated using the Auto Correlation Function (ACF) and the Fourier based power spectrum.
An ACF example of synthetic and real images with identical parameters is shown in Figure 37. A similar example but with a windowed low-pass filter spectrum, is show in Figure 36.
For both the ACF and the FFT spectra methods and data from both the real and the synthetic domain, it is possible to infer lower order lens aberrations (A1, defocus) via traditional supervised learning as shown in the previous section. An example is shown in Figure 38. The fact however remains, that even with aggressive low pass filtering over the power-spectra, differences remain between spectra originating from the digital twin and those from real data. Supervised learning methods remain sensitive to these differences, even when pro-processing methods like ZCA whitening or Instance-norm (IN) are deployed.
The final feature we decided to use, which bridges the gap between data domains is known as the eccentricity. In effect, we will treat the power spectra as realizations of a two-dimensional bivariate normal distribution from which we can derive the 2x2 covariance matrix and thus also the two eigenvectors. The extracted feature is simply the proportion between these two axis. It indicates how circular or how much of an ellipse the shape is, though lacks angle information. An example is shown in Figure 39.

5.8.3 Basic Agent Design

Figure 40 and 41 showcase a set of Ronchigrams for both real and synthetic data. Here each row has a distinct A1 lens aberration, whereas the columns are the so-called defocus stack. That is, the image is

under-focused on the left, and then as you progress to the right, the images come in focus and finally over-focused. The first row has zero lens aberration and is optimal in that sense. This checkerboard illustrates several important things.

- For a human observer it is not possible to ascertain the A1 lens aberration from single ronchigram alone.
- A human operator will therefor look at the images as the microscope is taken from under focus to over focus.
- The focus stack should therefore be used by an agent to infer the A1 lens parameter.

The real and synthetic environment are therefore defined based on the power spectra illustrated in Figure 42 and 43.

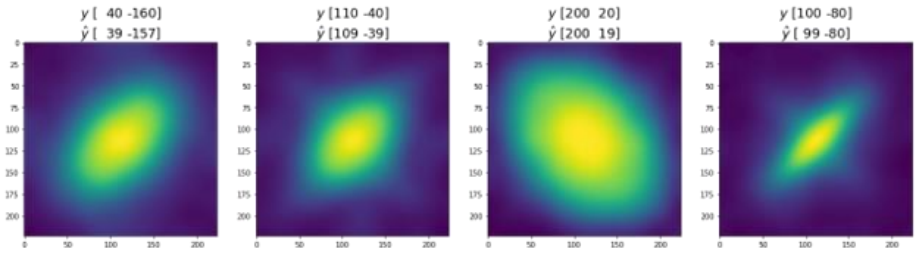


Figure 38 A1 lens aberration estimation using supervised learning based on the power spectra of synthetic data.

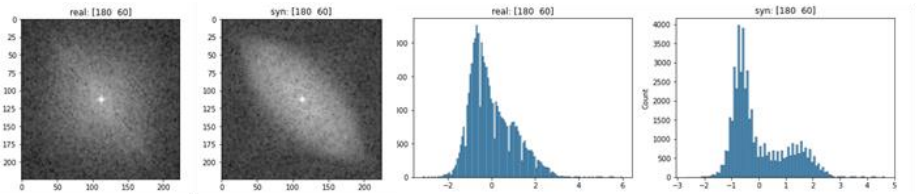


Figure 39 Differences between a synthetic and real image based power spectra for identical lens aberrations.

The reward function is based directly on the eccentricity values. They are shown in Figure 44 and clearly close to convex. To be able to have a reward function for combinations of A1 and defocus values that are not present in our dataset, they are simply interpolated via a Gaussian radial basis function. The final architecture is shown in Figure 45. This particular reward is an example of a so-called shaping reward.

5.8.4 Results

The proof-of-concept agent is based on (double) Q-learning with a separate target network. [B27, B28] The environment is formed by the dataset of available Ronchigrams, the reward function is based on the inverted interpolated eccentricity values. Each turn the agent my turn either the A1 or defocus knob up or down. The goal state is the zero-zero state. In the checkerboard images, this is simply the central column in the first row.
As the input is a single scalar, this agent is a tiny multilinear perceptron (MLP) network and therefore fast and easy to train.
The agent is trained on the digital twin, and then deployed on the real data. Results can be seen in Figures 46 and 47.

Although successful, there are a few caveats with this initial model:

- Reinforcement learning is plagued by high variance in the learning process, and this agent is no exception.

- Results on the real data are not stable. It currently manages to reach the goal state in about 1/3 of the runs.
- The dataset is currently quite small.
- The reward function is a shaping reward. One of the drawbacks of such a reward, in contrast to simply assigning a -1 and 1 to a loss and a win respectively, is that any time you introduce shaping, there is a probability for learning a non-optimal policy that optimizes the wrong objective. [B40, B41]. Secondly, if the reward must be shaped, it should be such that there is little delay between action and consequence. The faster the feedback mechanism, the easier it will be to learn a path to the high reward.
- Although the agent deploys to real data, it's still deployed to an environment which knows the goal state. To deploy to an actual microscope, the agent will have to learn how to stop by itself.

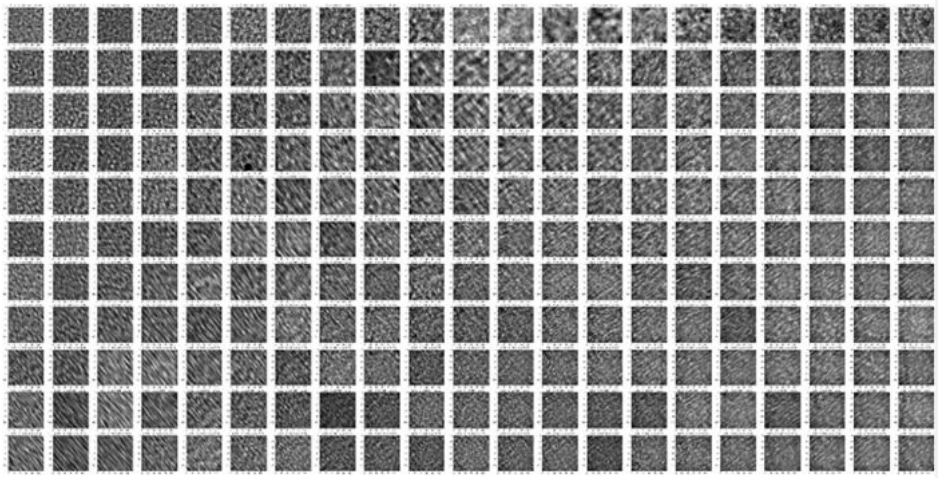


Figure 40 Real Ronchigrams with different defocus and aberrations. Each row represents a so-called focus stack, where focus was changed from under, in, to over focus. The top row has zero A1 aberrations, the bottom row the worst. Note that these are estimations done with a separate tool. The real microscope has no fixed state, all corrections are relative. Hence there is no real way to state that a lens is causing a certain known aberration.

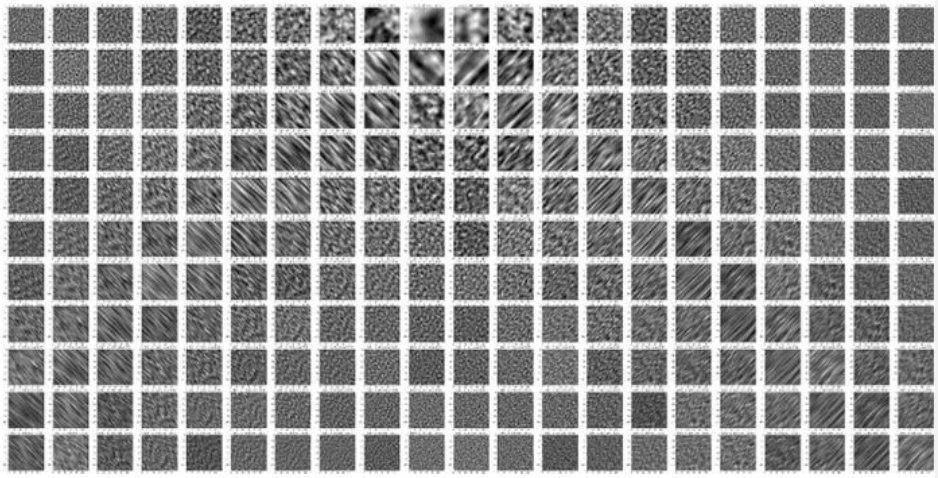


Figure 41 Synthetic Ronchigrams with different defocus and aberrations. Each row represents a so-called focus stack, where focus was changed from under, in, to over focus. The top row has zero A1 aberrations, the bottom row the worst.

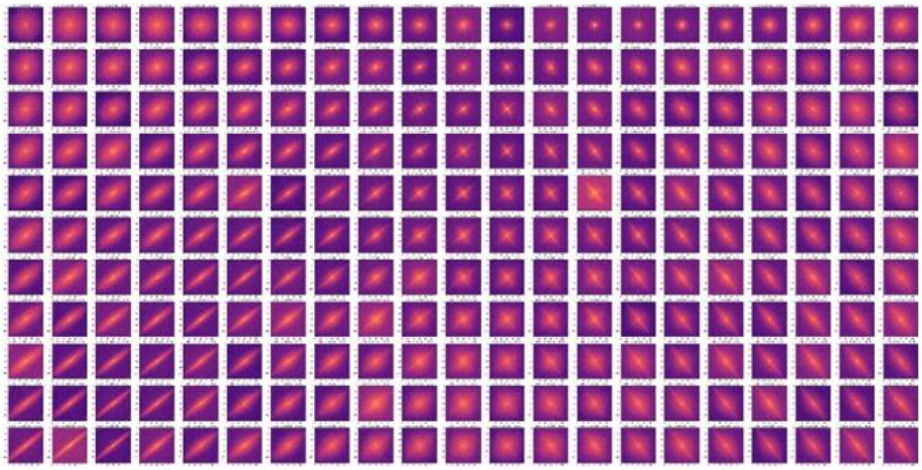


Figure 42 FFT spectra of real Ronchigrams with different defocus and aberrations. Each row represents a so-called focus stack, where focus was changed from under, in, to over focus. The top row has zero A1 aberrations, the bottom row the worst.

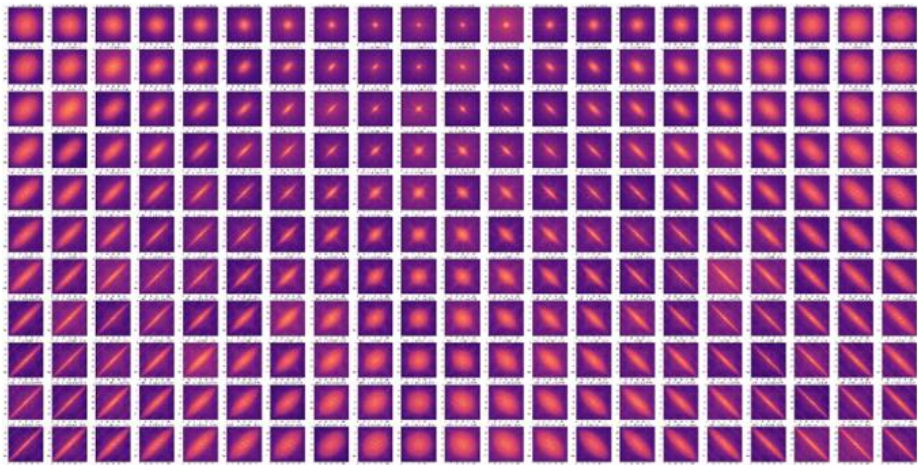


Figure 43 FFT spectra of synthetic Ronchigrams with different defocus and aberrations. Each row represents a so-called focus stack, where focus was changed from under, in, to over focus. The top row has zero A1 aberrations, the bottom row the worst.

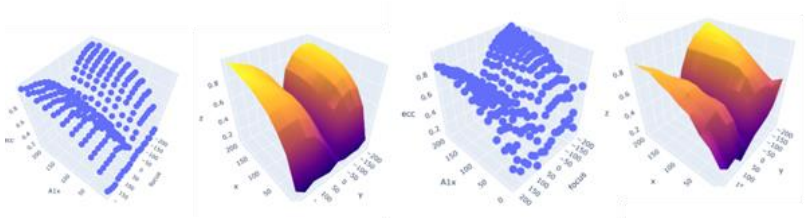


Figure 44 Raw and fitted eccentricity values for synthetic and real data. The fitted surfaces will form the reward function for any A1x and defocus knob value.

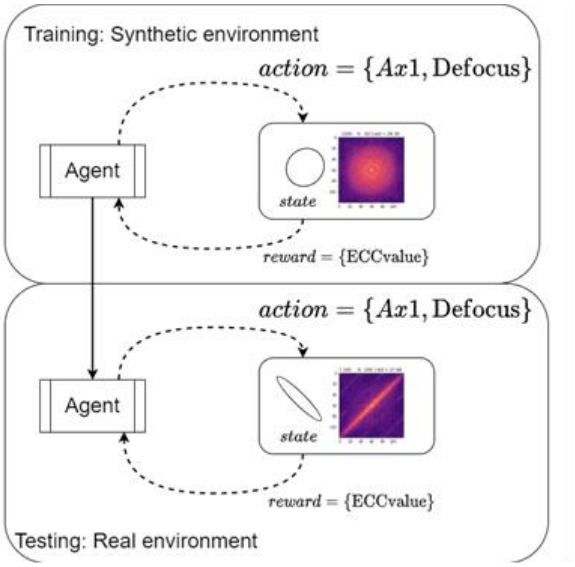


Figure 45 Digital twin-based agent training, where the reward function is based on the eccentricity value of the smoothed Ronchigram power spectrum.

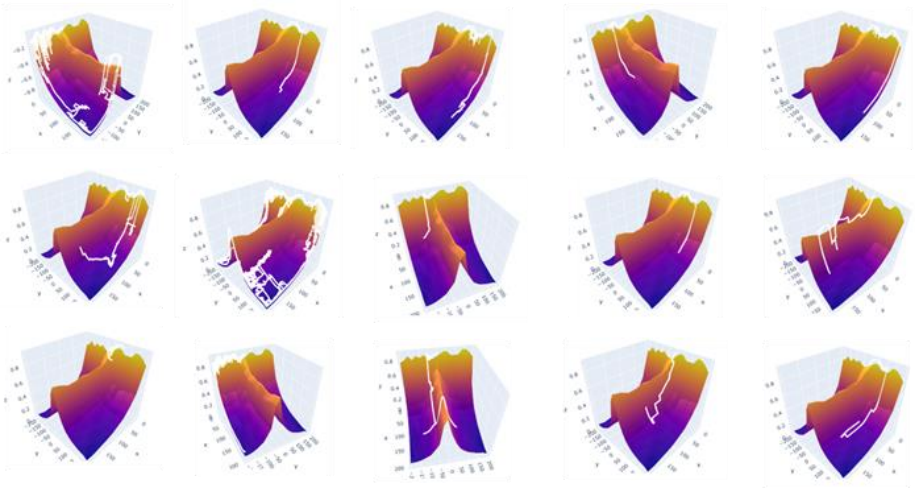


Figure 46 Example learning episodes of a Q-learning agent. The white path is superimposed over the reward landscape. The goal state is (0,0).

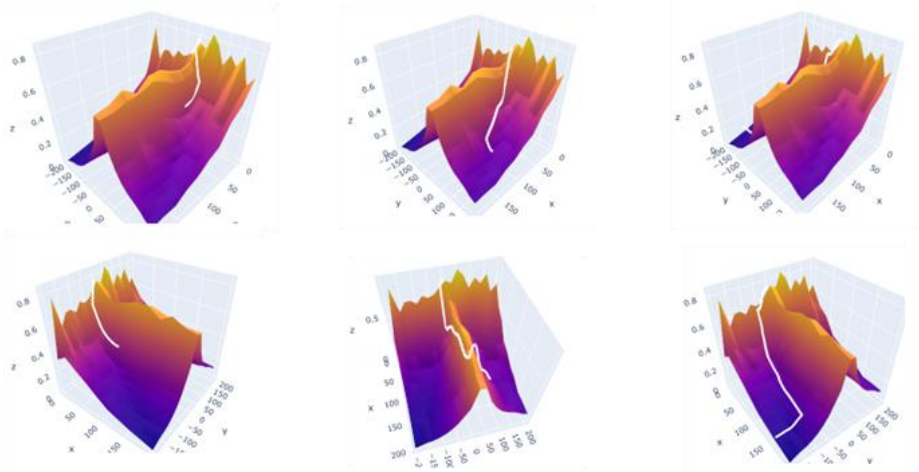


Figure 47 Deployment of a digital twin-trained Q-learning agent of real data. The agent has never seen this data before.

Figure 48 Ronchigram focus series data, raw (top) and FFT (bottom)

Commented [K(85]: This figure is way too small to make sense of

5.9 Image based RL

The next step towards automatic microscope calibration is to forgo the handcrafted features and have an RL agent learn explicitly from input images alone. [B27] Convolutional neural networks have proven themselves to be excellent in this regard, be it at a computational cost. In the initial proof of concept architecture, the agent is fed 120x120 pixel Ronchigram images. Similar to the previous design, these images are pre-processed by taking a windowed Fourier transform, followed by a low pass filter. This pre-processing introduces human knowledge, making the feature extraction for the Neural Network simpler. Initially the pre-processing was not performed to be able to compare baseline results.

Also, in this scenario we aim at training the RL agent with synthetic data and deploying the results on real data.
For this set of experiments one-dimensional datasets of focus series datasets were used. In future steps this will be extended to multiple dimensional data. This simple setup was chosen, as before, to get a better grip on the behaviour of the agent, to understand the reliability and behaviour variations, and to experiment with RL algorithms. It is also very much desired to achieve a small success with RL agents in an environment and circumstances that are both understandable and explainable before moving on to a more complex problem.
Another important variation on this set of experiments is given by the reward function variations. The environment can choose to return intermediate step rewards (e.g. by reaching some improvement but not yet being at the goal), or we can decide that the environment only returns rewards when the goal is reached. In RL both approaches are used depending on the circumstances.

5.9.1 Results

One of the primary causes of the agent results were driven by reward engineering, which is a known challenge for RL practitioners [B62]. As such, simple empirical experiments were done to measure what methods works best in the current environment. In this case four different reward functions were tested:

- With mid-step rewards:

1. `+1 if |focus_current|<|focus_old| (v0)`

2. [1,…,max,…,1] if |focus_current|<|focus_old| (v2)
3. 1 – penalty; penalty = steps/buffer_length (v3)

- End reward
  1. +1 when done (v4)

The focus series comprises 81 images, the middle one being the one in focus, and varying from [-200, 200] from left to right. The starting position for the agent is random at any position of the series. The agent can choose to go left [0] or to go right [1] in every discrete time step. Figure 49 shows an image of the Ronchigram series: synthetic Ronchigram on top and synthetic FFT Ronchigram on the bottom.
In the specific A1 case, a +1 value was rewarded, any time that the agent was progressing in the right direction. This is made possible via access to the metadata. Of course, in the end it is undesirable to access metadata.

Figure 49 Ronchigram focus series data, raw (top) and FFT (bottom)

Commented [K(86]: Too small

In case A2, the same approach is followed as in A1, except that the gains while approaching the focus are exponential (starting by 1 at the extreme and doubling at each step). How the gains are distributed also influences the outcomes.

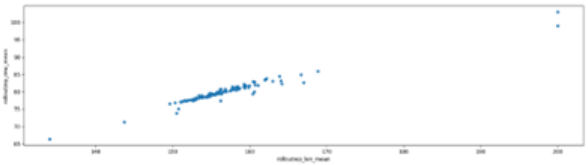


Figure 50 Example of training data with DQN 20 000 timesteps and reward function A1.

In A3 another approach is tried, trying to minimize the steps used by the agent to reach its goal. Every extra step taken trying to reach the goal adds less reward as in the expression seen above. Of interest is noting that when the penalty increases too fast or too slow, the performance also is reduced. There is thus also a sweet point in this approach.
The approach in B1 is different and well known in RL as well, rewarding the agent with a +1 only when reaching its goal, with no more intermittent clues.

In running the experiments, a series of steps are taken. The training step needs to be defined by choosing an algorithm and passing it the desired hyperparameters (which is not a trivial task either, and needs many trials to become satisficing). The only universal metrics to monitor if the training is running successfully are the reward against a defined time metric (epochs, timesteps). Yet no hard conclusions can be drawn from this kind of data. Many other outputs can be expected from a given algorithm, all those other output parameters in general have little to make with performance monitoring and mostly to make with algorithm internal functioning; these kinds of parameters might be useful when debugging or changing implementations within the algorithm itself.
After training, the behaviour of the agent can be tested by measuring performance with a battery of experiments. For example, inferring the model on 100, 1000 or 10000 tests. Only extracting the averages on these tests is not enough to draw conclusions, and taking a look at the distributions (e.g. histogram), and having an idea of the expectations on the results gives an approximate idea on what is going on.
Especially depending on the reward function different behaviours are to be expected. Let us see a few examples in the graphs below:

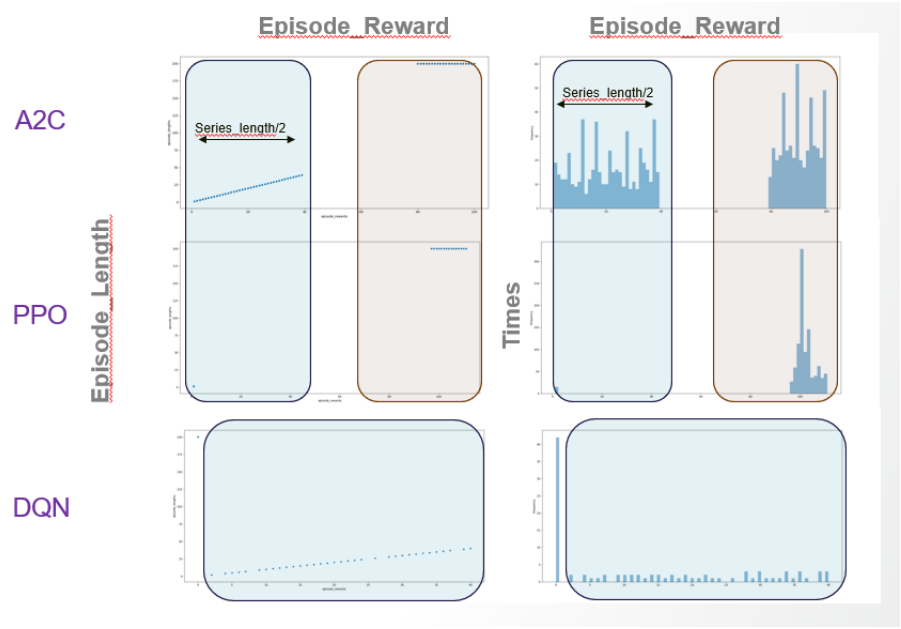


Figure 51 Reward function evaluation with A2C, PPO and DQN model training 20000 steps and 1000 inferences. Episode length against episode reward on right graphs and histogram on rewards right graphs. Reward function A1.

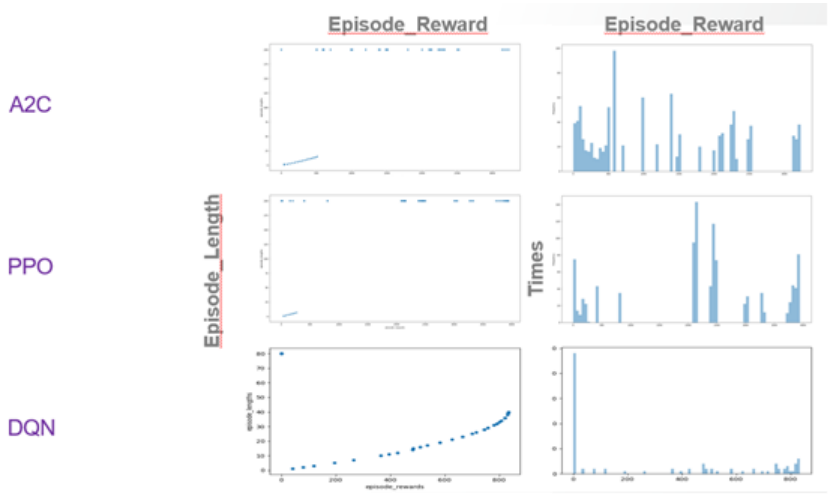


Figure 52 Reward function evaluation with A2C, PPO, and DQN model training 20 000 steps and 1 000 inferences. Episode length against episode reward on right graphs and histogram on rewards right graphs. Reward function A2.

The most important learning from the graphs above is that in the brown regions a policy-hack can be observed. A policy hack is an undesired behaviour of the agent with respect to what the desired behaviour is defined by the human programmer and reflected as well as possible by the reward function.
The agent should be finishing its exercise within 40 steps taking into account that its needs to discover the correct direction first. What happens in the brown steps, is that the agents learns to maximize rewards, pivoting left and right until reaching the maximum allowed number of steps (which is also defined in configurations). This happens for DQN and PPO.
For DQN we also can observe that in a number of occasions the end was not reached within the desired number of steps. Sometimes it is hard to interpret if what we are observing is unwanted behaviour or just failure of training.
Of course, this behaviour hack could not happen by only reaching the goal at the end. So certainly defining the reward function is key in problem design. And it is one of the most difficult steps in the process, especially when the problem takes on complexity.
The behaviour hack becomes increasingly worse, as training progresses.

Figure 53 Inference 100 repetitions after model training with 20 000 steps. Performance in DQN, PPO, A2C is comparable on synthetic data.

If now the reward function A2 is taken and the same experiment is repeated the results observed are a little bit different.

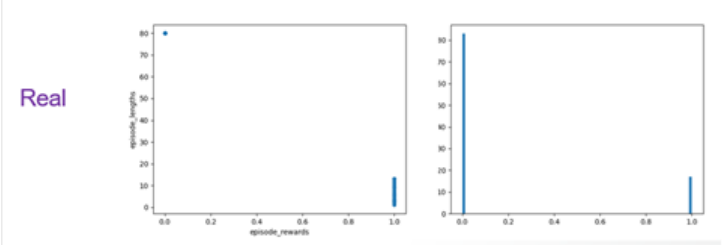


Figure 54 Transfer of model to real data. Performance drops to an average of 20-30%.

The distribution on the histograms gets more spread, but still some of the unwanted behaviour can be observed.
What can also be noted from the previous experiments is that the success rate varied from experiment to experiment. Before going into the performance numbers, the right reward function needs to be defined. In the previous cases we observed unwanted behaviour. Let the focus be on the direct reward only when reaching the goal; reward function B1.

Performance in this scenario improved, after a series of inferences of 100 runs, the performance is a little bit above 50% of successful runs. When trying to use the same model with real data, the performance drops badly.

This means that the gap between the synthetic and the real data is still big, and we cannot speak of generalization.

Another important fact to note is that when training on real data, the model performs much better even only after the same 20000 steps, achieving close to 100% success rate. This should raise a lot of questions about the existing simulated data.
–What about when using the FFT images? The training data gets better, and the inference performance gets slightly better. The most important fact is that the transfer to real data when using FFT images works. Introducing this human knowledge allows for much better transfer, and that is a promising result.

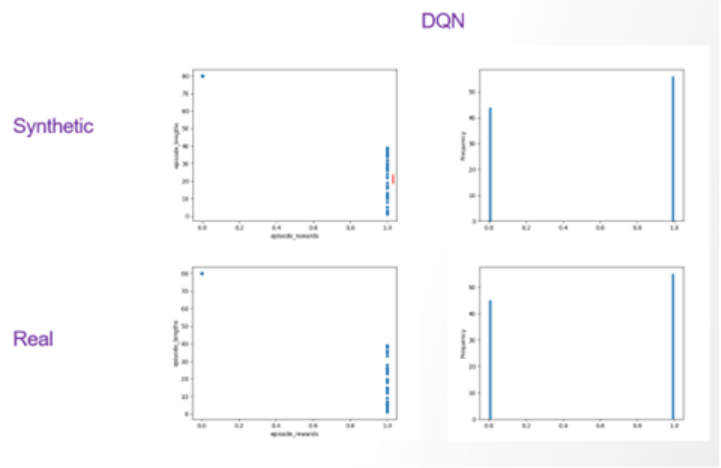


Figure 55 Training and inference on FFT images (top) and transfer to real FFT data (bottom) with DQN.

The tools in RL are young and not utterly developed, meaning that bugs can be found in open-source libraries and the programmer also needs to cope with limited functionalities. From design to experiment completion many parameters need to be repeatedly changed, and thus all experiments should then be rerun with exactly those parameters. This process takes time, and automation needs to be in place. For example, in the experiments mentioned along the way, the number of allowed maximum steps was limited first to steps=200 and then to the series length steps=81. This was done to avoid policy hacks. All changes should be kept on track carefully. This is just an example to illustrate the myriad of possibilities present from the start. Other changes along the experiments included stepping from 'MlpPolicy' to 'CnnPolicy', changing a few hyperparameters of the algorithms used just for practical reasons (e.g., memory capacity) as would be the case of the DQN buffer length. No focused hyperparameter tuning towards performance was carried out and this is one of the next steps.

5.9.2 Summary

- Training losses can't be used to draw any conclusions about the desired behaviour, but it serves as a surrogate to monitor performance.
- Reward engineering is a hard problem in RL, even for straightforward problems.
  - Undesired behaviours result from policy reward maximization problem (policy hacks).
  - Mid-term rewards are a doubled-edged weapon (introducing knowledge with risk of undesired behaviours), commonly used in more complex environments (e.g. robotics).
  - Evaluating environments with mid-term goals makes it harder to evaluate goal reaching.
  - If only a goal reaching reward is given, less knowledge is introduced, making it harder to reach this goal, but the risk of policy hacking is greatly reduced.
  - Constrained MDPs can provide a solution *[B63]*

Commented [K(87)]: straightforward

- Different reward functions will potentially cause different outcomes, in behaviour and performance.
- Loggings and statistics are the only tool to monitor agent behaviour, and even then:
  - Variability in evaluation performance can have a wide range.
- Ronchigram training results in poor generalization to real data.
- FFT image as input offers equivalent results with synthetic data (slightly better).
- When adding more training steps FFT improves better than Ronchigram data.
- FFT images offer good generalization to real data.
- Introducing human knowledge, when possible, makes the problem more tractable, especially in places with no risk of reward behaviour changing.
  - Reducing steps of freedom by introducing knowledge when possible.
  - Middle step rewards are a dangerous tool to use, easing convergence but potentially adding lots of reward engineering and undesired behaviour from the agent side.
- Real data training works remarkably well (vs synthetic data)
- Choice of algorithm:
  - Choose max 2 potentially good algorithms and concentrate on optimizing the workings.
  - Hyperparameter tuning is key.
  - Understanding of internal working of algorithms of interest to be able to build and improve code when it needs personalization.

In the image-based RL setting, the points above are interesting first results in researching the self-calibration problem. As RL is a young field with immature tools, the road to making the desired software operational and reliable will be beset with questions and challenges like the ones mentioned already. On top of that, other problems independent of the RL setting still need to be further improved (as the gap from synthetic to real data) to reach the desired performance.

5.10 Future directions in Reinforcement learning

In this section will we survey the literature where relevant to build upon our initial reinforcement learning prototype. We will cover the following themes:

- Meta RL
- Model-free based RL for dynamical models
- Policy based reinforcement learning
- Memory augmented models for vision problems
- Use of experience replay
- Reward strategies:
  - Objectives specified as individual rewards components and as simple as possible.
- Data sampling strategies:
  - Environment based automatic domain randomization

5.10.1 Meta Reinforcement learning

A huge drawback of scores of RL algorithms is the fact that they are intimately tied to the environment they are trained and tested in. The upside is that you may overfit an algorithm to do well for a single task, such as an Atari game. But the consequence is that these agents do not generalize at all to slightly different tasks. This goes so far, that agents learn different behaviours pending the random seed in an otherwise identical environment [B30].

Meta RL tries the seemingly impossible, namely, to train agents that generalize to different environments that have never been seen during training. This is accomplished with a limited amount of finetuning, where a meta model adapts its internal configuration to the new environment.

Early work from [B31] uses an LSTM cell for adaptation to new Markov Decision Processes (MDP), which was further developed in [B32] and [B33]. They train their model over a set of MDP's. These tasks are somewhat different though similar in nature. Such as a robot with slightly different physical parameters, or a maze that differs. The main difference to traditional RL is the fact that the policy not only observes the state, but also the last reward and the last action. This mechanism is used so that the agent may

learn from a history of states, actions and rewards and adjust the dynamics when needed. Key components are:

- Deploying a recurrent model with a memory state. The hidden state is used to encapsulate knowledge on the current task. It is updated during roll outs
- A meta-learning algorithm. In [B32], [B33], this can be gradient descent to update an LSTM next to a reset of the hidden state, the moment a new MDP is encountered.
- A distribution of MDP's.

Work from [B34] treats the hyperparameters as learnable parameters: specifically, the discount factor and bootstrapping parameter are learned. These are optimized via a second (meta) objective function and using cross correlation over a sequence of consecutive episodes.
As stated earlier, the exploitation versus exploration dilemma is central to RL. Common solutions include epsilon-greedy action selection, adding random noise to actions, or using some type of stochastic policy. Work from [B36] aims to learn structured action noise by conditioning it on a pre-task (latent) random variable. The variable is sampled per episode and should determine the exploration behaviour best for this particular roll out. This latent variable is also learned using the total rewards. It also stipulates, similar to VAE ELBO [B37], that the learned latent variable its distribution is close to a normal Gaussian.
A similar idea is explored in [B35], where the authors use a learned latent vector conditioned on encoded actions and states. During rollout actions are then sampled based on the encountered state and the learned latent vector. This is done to prevent an agent to explore to much outside its learned path.

5.10.2 Model based RL for dynamical models

Model based RL is typically more sample efficient than their model-free counterparts, but usually lag in terms of performance. In a sense they suffer from opposite problems. Model based methods such as Gaussian processes can learn fast from little data, but struggle with highly complex or discontinuous systems. In contrast, neural networks are excellent approximators, but overfit in low data regimes [B39, B44].
Model based approaches may roughly be divided into:

- Methods that use analytical gradients
- Sampling based planning
- Model based data generation
- Value equivalence prediction

5.10.2.1 Analytical gradients

Model assumptions about the dynamics or cost function are convenient because they may yield closed-form solutions for optimal control. [B48]. Similarly, a dynamics model parameterized as gaussian process will have analytical gradients. Control models may also be used to generate guiding samples for training more complex policies. [B49]

5.10.2.2 Sample based planning

If models are completely non-linear, local optimality may not be guaranteed and one must resort to sampling sequences. So called random shooting [B50] samples candidates from a fixed distribution and uses a model for evaluation to choose the best action. More advanced variants iteratively adjust the sampling distribution. For example, work from PETS [B39], combines a model-based approach together with an ensemble to forward sample trajectories. Each (dynamical) model encodes a probability distribution from which one may later sample. This algorithm was later used by [B38] to continuously control an adaptive optical mirror, whilst learning to adapt at the same time.
In discrete environments, its more common to use search over tree-structures, than to iteratively refine a (single) trajectory. Monte Carlo Tree Search (MCTS) [B51] has formed to basis for many impressive results in game playing. Famously, AlphaGo [B52]

5.10.2.3 Model based data generation

Data augmentation is widely used in ML to increase the size of a dataset and combat overfitting. In RL one can envision using a predictive model to generate synthetic data. The Dyna algorithm by Sutton [B54]

alternates between learning, data generation by the model and finally policy learning using the model data. Similar strategies are used by iLQG [B53], in meta-learning in general and even in applications that depend fully on image observations. [B55, B56, B57]
Lastly, a variant of this technique is commonly seen in temporal difference learning when a separate model is used to improve target value estimations. [B58]

5.10.2.4 Value equivalence prediction

Finally, an intermediate approach between model based and model free methods is offered by value iteration [B59]. Here generated trajectories are only constrained to the real environment in the sense that they should have the same cumulative reward. They have proven to be effective in high dimensional observation spaces. [B60]
To summarize, work from [B49] shows that predictive models generalize well enough to overcome their implicit bias but suffer from compounding errors when making long horizon rollouts. Like PETS [B39] they use probabilistic model ensembles combined with a stable off-policy model free optimizer. [B61]

5.11 Pixel-based RL introduction

The watershed paper for image or pixel-based reinforcement learning was the work of Volodymyr Mnih et al. in 2013, *Playing atari with deep reinforcement learning*, which showed that it was possible for an an agent to learn how to play a whole suit of Atari computer games from the screen captures only. It surpassed sota on six out of seven available games and human performance in three. In particular, it combined a CNN-based encoder for processing the images, deployed a replay-buffer [BB2] and stochastic mini-batches.
This work was extended in 2016 by them using A3C (Asynchronous Advantage Actor-Critic), which allowed for parallel learning [BB3] and extended by [BB4] with the introduction of IMPALA (importance weighted actor-learner architecture). Proximal Policy Optimization (PPO) was introduced by [BB5], that has proven to be effective in other pixel-based environments such as Vizdoom [BB6].
A special category is formed by the so-called *world models*. Conceived by Ha and Schmidhuber [BB7] these methods learn an internal model of the world from raw pixels. They typically contain a separate encoder structure to process raw image pixels, which in some cases may be trained separately from the model and behavioral parts of the architecture.
Dreamer type reinforcement learning borrows elements from both model-based and model free RL. It was conceived by [BB8]. These methods incorporate a world model next to a mechanism to predict future states. The latter can use an extra loss function to validate if these predictions hold. Dreamer type models have reached sota over a wide variety of domains [BB7, BB10, BB11].

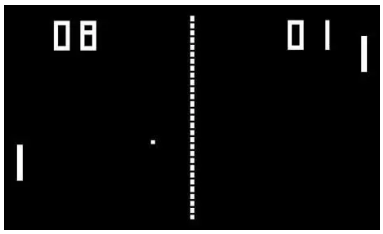


Figure 56 A screenshot from the game Pong. What is the problem if an Agent only receives this frame?

5.12 ASIMOV Base Architecture

The base case for ASIMOV follows the work of [BB1] closely with the exception that we deploy a much heavier backbone model (Resnet18) and the inverse Fourier transform. An overview can be seen in Figure 57.

- The encoder architectures from [BB1, BB4] proved to be insufficient for learning. Conversely, architectures heavier than Resnet 50 gave little to no benefit.
- We train end-to-end using ADAM [BB12] instead of the RMSprob [BB21] method used by [BB1].
- It uses a variant for the frame-stacking from [BB1], which we call *focus-stacking.*

5.12.1 Focus-stacking

Basic Q-learning is fundamentally based on the assumption that the state adheres to the *Markov property*. All information the agent needs to make an optimal decision is encapsulated in that state. (Illustrated in Figure 57). Our problem needs to be able to be framed as a Markov Decision Problem [BB17] (MDP), where the future only depends on the current state, not the history. Formally:

$$P[S_{t+1} \mid S_t] = P[S_{t+1} \mid S_1 \dots S_t].$$

Or put differently, the future and past are *conditionally independent* given the present, as the current state encapsulates all information [BB22].
For Ronchigrams this is clearly not the case. This can be seen in Figures 40 and 41. The fact that a multitude of aberrations may lead to the same wavefront is grounded in physical reality and thus something we must deal with. This situation may be resolved by either using a sequence of measurements and logged microscope settings through time, or by taking multiple measurements while willfully changing the so-called defocus setting on the microscope. These combined Ronchigrams uniquely identify the A1 magnitude parameter.
This last option comes at the cost of taking seemingly superfluous measurements but is technically simpler to implement. The first pixel-based RL algorithm for ASIMOV uses it.

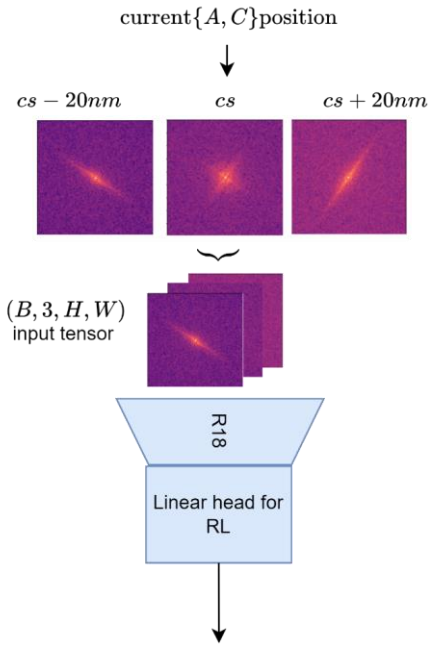


Figure 57: Base DQN architecture with a Resnet 18 encoder and a linear head.The latter is dependent on the number of actions that is available.

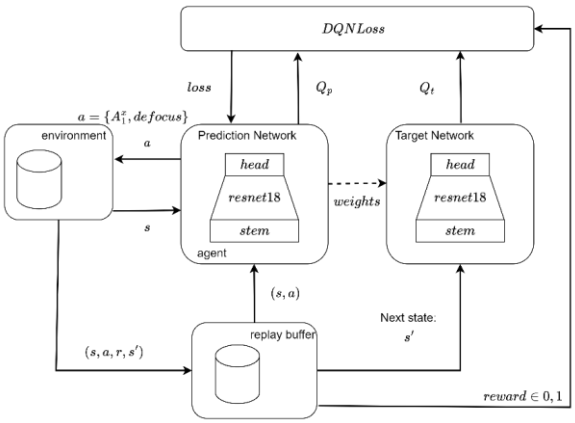


Figure 58 General DQN learning architecture adapted from [1].

5.12.2 Initial Results

Figure 59 shows the initial results of a synthetic trained agent that was deployed and test on actual real microscope data. For this particular style of evaluation, we are not interested in the accumulated rewards but rather if an agents manages to reach the goal state (at all) and if so, how far above par it was in terms of the number of steps taken vs the theoretical minimum shortest path.
A red square indicates that an agent that started from that grid point, where each grid point corresponds to a set of defocus and A1 astigmatism settings, reached the goal. Figures 60 showcases an identical agent with and without the use of focus stacking. Using multiple images increases performance from 36.3 percent to 85.3 %.

5.12.3 Over-training

In this design the only measure that is taken to close the domain gap between synthetic and real data is the use of the Fourier transform. The previously used low pass filter has been dropped. This means that over-training on the synthetic dataset is a significant problem that must be guarded against. This is illustrated in Figures 60 and 61. The left shows generalization results after a 150'000 training iterations resulting in 85.3% performance, the right model has been trained 300'000 iterations and only reaches 50.1%.

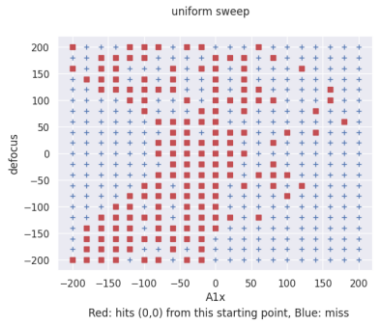


Figure 59: Basic DQN with Resnet18.

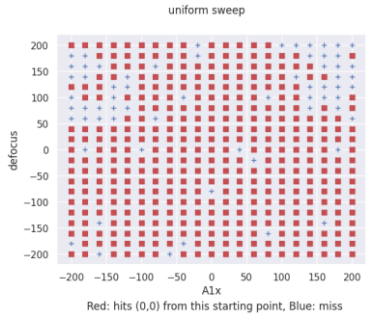


*Figure 60 DQN with Resnet18 with focus-stacking. Evaluation to real data after **150k** synthetic training iterations.*

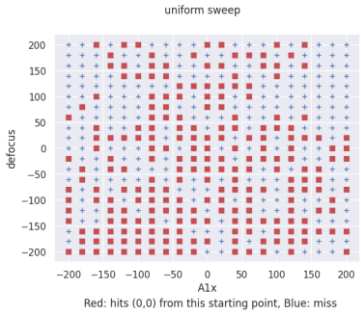


*Figure 61 DQN with Resnet18 with focus-stacking. Evaluation to real data after **300k** synthetic training iterations.*

Commented [v(88]: Top right and bottom left images seem to be the same. Is that true?

Commented [MD89R88]: True, fixed

5.12.4 Determinism

Analyzing the paths of agents that miss their goal, there are a couple of things evident. Outer left corners corresponding to (extreme) A1 astigmatism and defocus values are difficult to deal with resulting in 'stuck' behavior. Secondly, the agent has significant difficulty around the goal state where real images tend to be more similar. This behavior is demonstrated in Figures 60, 61 and 68. In the DQN algorithm, when performing inference, the agent can operate in two ways. Either by taking the action adhering to the maximum Q-value as by policy default, or by treating the Q-values as probability mass function from which one draw samples. In areas where the agent is confident and robust a less optimal action may be corrected along the path. Around the goal or in a corner, it can free an agent from a zone of ambiguity.

5.12.5 Q and V values visualization

Let formally:

$$V_\pi(S) = \mathbb{E}_\pi[\, G_t \mid S_t = s \,],$$
$$V_\pi(S) = \sum_{a \in \mathcal{A}} Q_\pi(s,a)\pi(a \mid s).$$

Commented [K(90]: Missing

Commented [D(91R90]: The equation editor doesn't work in the online web interface with firefox for me either, but if I open the doc normally, it all works.

And:

$$Q_\pi(s,a) = \mathbb{E}_\pi[\, G_t \mid S_t = s, A_t = a \,].$$

We can get some insight in what the agent deems a good state to be in, by looking at a pseudo value function defined as:

$$\hat{V}_\pi(S) \sim \sum_{a \in \mathcal{A}} Q_\pi(s,a).$$

This is illustrated in Figures 62 and 63, where for Q-values the variance is shown over the available actions. From the pseudo V values its immediately clear that the agents learns the distance from a Ronchigram state to the origin effectively. There are a couple of noteworthy things:

- These maps are (near) convex.
- The pseudo V-maps from real and synthetic data show great similarity.
- Good state information, bare an exception, is not sufficient in itself for an agent to act upon.
- Being in a bad state and recognizing this is not disadvantageous, if you know confidently what to do. This is hidden from view in the pseudo V-maps.
- These maps may allow for a decoupling were assigning a value to a Ronchigram state is done by RL and the behavior for the next move is governed by a different algorithm altogether. Such as Bayes (active) inference.

The default Q-learning policy is to choose an action that pertains to the maximum Q value for all actions available in that state. Ideally you would like the entropy to be low over this range. The variance maps in Figures 64 and 65 show a slightly more complicated picture. Clearly around the target there are many zones where the agent doesn't exhibit any strong beliefs.

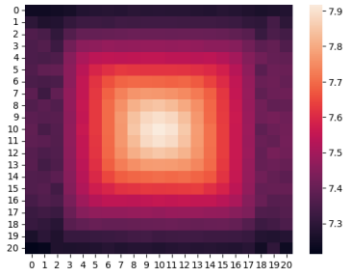


Figure 62: Pseudo v map for digital twin data and the Vivit architecture.

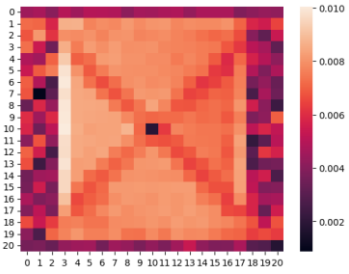


Figure 63: Variance of the Q values for digital twin data and the Vivit architecture. Note the noisy border effects due to the focus-stack not having valid neighbors in its range.

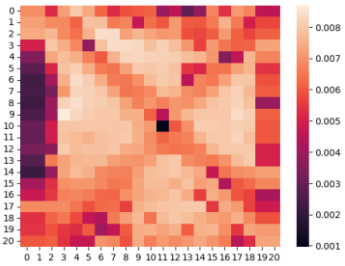


Figure 64 Pseudo V values for the real microscope.

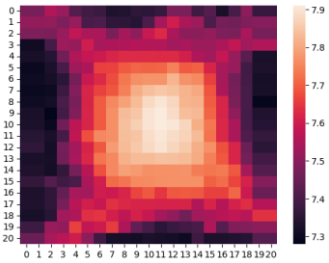


Figure 65 Standard deviation of the Q values per state-action pair for real data.

5.12.6 Architecture variants

A number of different architectures have been tested:

- The MobileNetV2 backbone [BB13]. This backbone is used extensively both in 2D and 3D computer vision problems. For an input image of 224 by 224 it has 3.4 million parameters, which is roughly 1/3 of Resnet18 for a similar image size.
- A Resnet18 based backbone with weight sharing. Specifically, each image from the focus-stack is processed separately through the backbone after which positional encoding is explicitly added to the feature output vector. This positional information may both be fixed or learned. This architecture may be seen in Figure 66.
- Vivit. [BB14] is a vision transformer [BB15] adaptation for video. Here we adapt the architecture to use the channels in the focus-stack as imaginary time component and keep the division into 16 patches per slice.
- MobileNet V2 backbones with weight sharing and a GRU (Gated Recurrent Unit) [BB16] layer on top that explicitly learns from the sequence of latent embedded vectors from left to center to right.

Generally speaking, there isn't a single architecture that wildly outperforms the others. The weakest performance comes from MobileNetV2 but it is also the least heavy by far. The latter specifically suffers in all states approximately 40nm from the goal state. Otherwise, performance is on comparable. All other architectures manage to achieve over 85% performance on real data. We will revisit this observation extensively in work package 1.2.

5.12.7 Focus-stack variants

We have experimented with a number of variants of the focus stacking method:

- Varying the distance of the left and right neighbor from 20nm upwards to 60nm.
- Setting the distance to a maximum left and right and then uniformly randomly sampling a slice from between the bounds for improved robustness, i.e '*the hopper*'.
- Taking the difference between the left and right image and dropping the central state ronchigram from consideration.

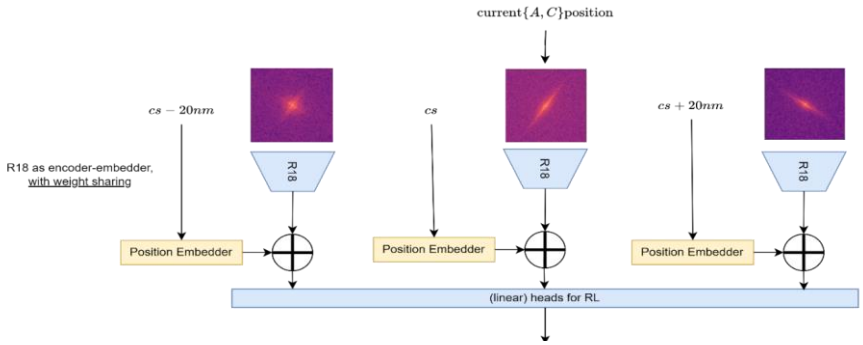


Figure 66: Weight shared Resnet18 with added positional encoding.

In general, the algorithm is completely dependent on the feature differences between left and right to determine its unique position in microscope lens parameter space. Training on the digital twin allows you to set this distance exact to anything exactly. However, on the real microscope these distances in nanometer are approximations. The process is a lot more noisy and as a result, generalization suffers. As a rule of thumb, we found it beneficial to have the left and rights neighbors at least 40nm away from the center state.

5.12.8 Sequence models

Another approach to deal with partial observable markov decision processes, is incorporating sequences of (previous) states into the model [BB89, BB90]. It is possible to use a recurrent model together with a value iterator, if one modifies the Replaybuffer to capture the hidden states of the LSTM or RNN needed for training. We, however, opted to switch to a policy gradient method and deployed PPO-LSTM [BB90] without frame stacking. This method too, comes with significant downsides: it is famously hard to train and requires careful hyper parameter tuning. [BB91]
We managed to get a proof of concept implementation running, be it at a computational cost, but its generalization performance remained severely lacking at around 62% percent, in comparison to methods that use multiple defocus frames.

5.12.9 Conclusions and Future work

Learning from end-to-end from raw pixels is a computationally intensive task and requires a fair bit of hyper parameter tuning. However, the performance of successful models is vastly superior to their manually feature engineered cousins.
The single most significant factor contributing to the current performance is the use of multiple images with identical A1 aberrations but different defocus values. Doing more of such measurements ensures that the resulting image set is unique for the state, and therefor satisfies the Markov property which allows our problem to be formulated as a Markov Decision Process (MDP).

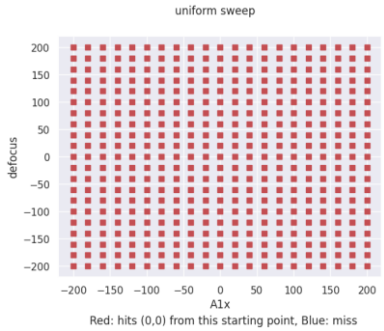


Figure 67: Generalization results for weight shared Resnet 18 with positional encoding.

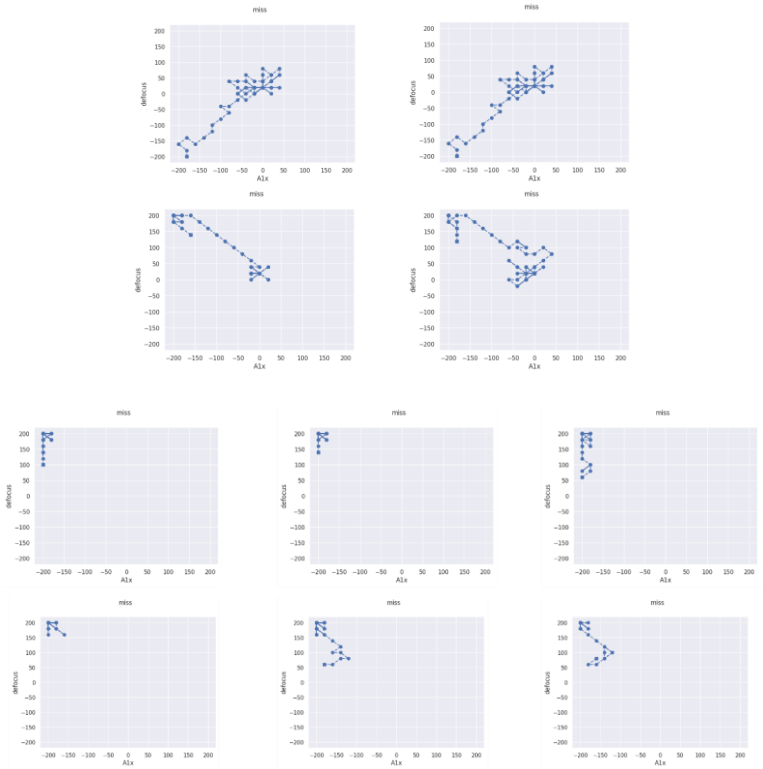


Figure 68: Agent stuck in the corner or lost around the goal state.

6 AI Architecture

6.1 Introduction

A reference architecture for the artificial intelligence will be developed in this part of the document, It should be used as a guideline for creating the architecture of a specific artificial intelligence subsystem. It contains the generic elements, aspects, and best practices. It is aligned in content and layout with the architecture of the digital twin [89]. The digital twin subsystem and the artificial intelligence subsystem are closely interacting, and both systems need to be performing in order for the whole system to work. In this document we intend to lay the basis for the exploration of the architecture of the full system in work package 4.

The reference architecture will comprise 4 main blocks generally following the approach of viewpoints of Software Plaform Embedded System (SPES) framework [91]. In the task on the architecture of DTs (T2.3) this has been found to strike the balance of level of detail and adequacy for describing a reference architecture being as use case agnostic as possible. The views considered are “requirements view”, “functional view”, “logical view” and “technical view”. However, we found that the implementation described in the technical view cannot be disconnected anymore from specific algorithms. No insights on an use case agnostic level can be gained here. Therefore, the technical view is omitted.

6.2 Requirements view

Most requirements stated in [90] (ASIMOV reference architecture) refer to requirements for the AI subsystem focused on the operational phase; that is also reflected in [89]. In this document the requirements for the AI in the training phase will be studied. Should all the requirements in the training phase and the operational phase be the same? Are there consequences to not having aligned requirements? Or are the alignments in the training phase not needed at all? These question and more will be explored in this chapter.

The AI and the DT subsystems are closely related, and the feedback loop between both is key as a prerequisite. Prerequisites can also be understood as requirements, so this assumption is the one that will be used to build the requirements of the AI subsystem during training.

6.2.1 AI subsystem – training requirements.

6.2.1.1 Functional

The AI subsystem will require the data usage as mandated by the goal of the full system, and the tools used therefore. Some of the tool functional requirements are specified in Document 3.1. Examples include constant size of the image (in the EM case 224x224) and format of the image (eg. .png). Other specification indicate what are prerequisites for all the parts to work together, related to infrastructure and software (e.g. GPUs to run RL or containerized environments).

This specification will be the same in the training, operational and finetuning phases, and will accept small variations in the process.

In the UUV Use Case the training process is realized by subcomponents which are well separated, in the sense that they are deployed in a container with own computational environments. Further, the components loosely communicate via interfaces. In order that the training of the RL-Model works, interfaces between the components must be defined carefully. Specific requirements in this hindsight are that the input-output and the interaction of the components are well-defined. A rough illustration of such system is given in Figure 69.

Commented [v(92]: Would it be nice to include the figure we discussed during the last GA here? The one which considered the table with the different levels of abstraction for the different subsystems (AI, DT)

Commented [v(93]: Is this one included in the chapter?

Commented [v(94]: Does this enumeration continue with

2. Functional view
3. Logical view
4. Technical view

below? Maybe some heading structure would clarify this somewhat (since the 1st main block is quite elaborate).

Commented [A(95R94]: We will be working on that next 2 weeks. We opted to leave out the technical view.

The cotents for 1/2/3 are ready we now need to put them in.

Commented [D(96]: I'm not sure what 'alignment' of requirements means. There are many different types of requirements (speed, accuracy, footprint, understandability, etc.). They can be directed to different parts of the AI component, or different phases of its use.

Commented [A(97R96]: ok

Commented [D(98]: Can this be made to apply to the AVL case? Maybe use the word 'data'?

Commented [A(99R98]: Yes good point I will review this content when we finish adding all to points of the architecture.

Commented [A(100]: Expand a little bit, even if some things of 3.1 are repeated.

Commented [SL(vd101R100]: Very valuable to have examples here IMHO

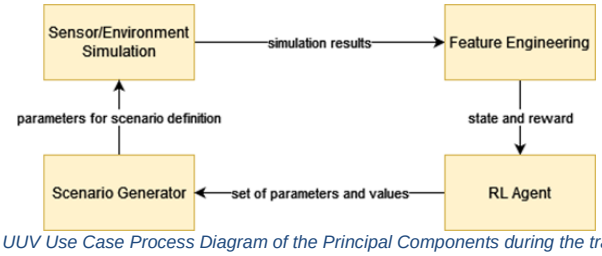


Figure 69 UUV Use Case Process Diagram of the Principal Components during the training loop.

6.2.1.2 Qualities

When taking into account the qualities of the AI subsystem (vs the functional requirements mandated by goal and tooling), the following will be addressed:

- Accuracy:

The accuracy required by the solution will be determined by a few numbers, for example reaching goal within 20nm. The accuracy metrics need to be a guiding force towards the development in the solution even in training phase.
During the training phase the accuracy is a guiding factor and not yet a must.

In the TEM use case the important requirement is to always reach the goal within 20 nm of the goal and then stop when in that position.

In the training phase this means that the training and the testing data need to reach roughly the same results. Not being able to reproduce training results for testing data results in a failure.

In UUV Use Case 1, the objective is to automatically create new and challenging scenarios to fine-tune the parameters of unmanned vehicles. The novelty and criticality of these generated scenarios are assessed using various Key Performance Indicators (KPIs). Unlike the TEM use case, there is no predefined optimal goal in this scenario, and any enhancement in KPIs represents a favourable solution for the use case. Additionally, one can involve experts' opinion to review the performance of the RL-solution.

In UUV Use Case 2, since the objective is to improve the performance of vehicle sensors based on their observations in scenarios generated by UUV Use Case 1, we have access to ground truth data in the form of an image stream from the environment. The accuracy requirement may be quantified, for instance, by achieving a high percentage of correctly identified objects and minimizing discrepancies between the sensor input and the actual image stream.

The information resulting from failures and successes determines quality standards for the DT, in the case of the DT requirements, it is hard to have them all upfront well-established and it is an iterative process. Of course, failing to produce qualitatively good data will result in failure of the AI, but the accuracy of the AI can be determined independently upfront, even when this dependency is known.

- Robustness

The AI solution will work under a certain specified condition of the TEM. Outside those conditions the behaviour cannot be guaranteed.
The factory conditions are difficult to specify upfront given a complex system with many variables, so often just by testing the solution in the expected desired conditions it can be determined what the working conditions are, and then it can be determined if it meets the needs. Finetuning on the DT side to adapt the data is then often required to meet desired working conditions. For different use cases different fine tunings can be achieved from a general trained AI.

For example, an AI can be trained with different sample thicknesses, convergence angles, electrons per pixels in acquisition etc. This functioning AI can then be retrained to meet working conditions, for example one specific low convergence angle (e.g. 14 mrad), standard acquisition dose (0.5 electrons per pixel).

Commented [D(102]: I'm not entirely sure what is understood by 'accuracy' here. Is it how well the procedure in the end approaches the goal?

Commented [D(103]: Isn't this reliability?

Commented [SL(vd104R103]: I agree. Accuracy links more to when the "goal" is met

Commented [A(105R103]: Changed

Commented [D(106]: Is there a strict criterium?

Commented [A(107R106]: I do not think that the factory requirements are sharp today. THey are still working on it.

Commented [A(108R106]: this was prefactory, but for the document and the sake of the document we do not need to distinguish

Commented [D(109]: test?

Commented [T(110R109]: Deleted the Train.

Commented [SL(vd111]: But instead a requirement to improve upon the current solution?

Commented [T(112R111]: Improvement yields if the KPIs improves or experts' validated the scenarios.

Commented [D(113]: for the UUV2 case

Commented [T(114R113]: Thanks. Added.

Commented [SL(vd115]: Can you make this more specific? I wouldn't know what you are referring to here concretely.

Commented [A(116R115]: I will re read it and rework it

Commented [SL(vd117]: Start this section with "TEM use case" in the sentence?

Those conditions do not need to be part of the first set of conditions, and in general if the sample solution worked, the specific one will be able to be trained too, if the rest of the goals of the AI stay equal. In UUV Use Case, Robustness can be achieved by giving slight variations in the unmanned vehicle response to actions (environment) provided by the RL agent (D2.2, Section 5.3)

Concluding, the AI solution is robust if it works within the required working conditions. It is in general hard to define these conditions up front, but they can be set experimentally by validating whether the AI provides an acceptable solution under certain conditions

- Reliability and reproducibility

Very related to accuracy, how reliable a system is and how reproducible its results are, is defined by how often and how well it reaches its goal. It can be stated that the solution should reach its goal above 90% of the time.
Ideally an (AI) system should be reliable all time under working descriptions as described above; but it is possible that in expected or unforeseen circumstance's reliability goes down. Having tools in place to make the AI solution as reliable and reproducible as possible is a good practice, all kind of software and logic can be used to that end.

In the TEM use-case the solutions should be reliable and autonomous, so in case of faced with unforeseen conditions or states, or if the model fails under expected conditions, a restart after 'x' seconds could be set in, or another model can be loaded to deal with the problem (if parameters logged allow for this detection), and even as a last resort a message to an operator needs to be sent to restart the physical system.

Only by testing under expected conditions numbers can be given about the reliability and reproducibility of the system, and during training the boundaries are rather soft, while this becomes a hard requirement in operational phase. But still, it is important to have a gross idea and expectation of what this number can and will be before moving to the operational phase.

To enhance trust in the automatic solutions generated by RL algorithms and to mitigate potential modelling errors, it is advisable to involve experts in the evaluation process, preferably on a sample-by-sample basis. This kind of testing is particularly valuable in RL applications where ground truth data is unavailable, as is often the case in scenarios like UUV1.

To facilitate human evaluation, as is the case with expert assessments, and to ensure reproducibility, it is crucial to employ tools for tasks such as data labeling and process labeling, documenting the training and operational processes, and version control of the RL model. The goal here is to have traceability to link experimental results back to algorithm, data set, and settings, and vice versa. In the UUV Use Case, we specifically make use of the ML-Flow framework for such purpose (see Figure 69).

Further, it is advantageous to complement the tools by a data analytics and management platform that enables thorough analysis of the resulting information. Ideally such a platform should allow interfaces between data, model and users allowing rapid development and usage of further analysis tools for, inter alia, the improvement of model training. Such an interface might be in form of Jupyter Notebook or no-code applications. Further, it is advantageous that such platform can assign hierarchical labels to data This hierarchical labelling system facilitates structured organization and categorization of data, making it easier to access and utilize effectively. Such labels can be seen as facets of the data, e.g., the stage of the data (e.g., cleaned or not cleaned). Furthermore, this label can be also used for realizing the component data interface replacing folders allowing flexibility in handling. An illustration of such a platform is given in Figure 70.

Commented [SL(vd118]: Sample?

Commented [SL(vd119]: Suggestion to add: "and have traceability to link experimental results back to the algorithm, data set, and settings."

Commented [T(120R119]: Thanks. Added the passage

Commented [SL(vd121]: Do you have a better quality image that is readable?

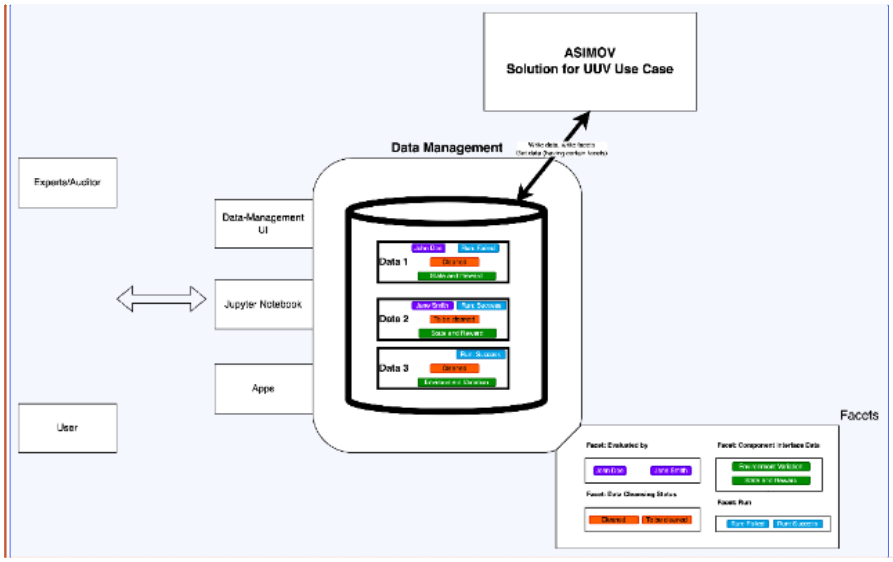


Figure 70 Data Management and ASIMOV Solution.

- Time to result

In training the time to result would be defined as the cycle of training and testing. In general it is desired that the trainings converge fast so that they can be tested as fast as possible and so make many iterations in the development to move on and have feedback on the achieved performance.
But the time to result defined like this depends on many factors, for example, in multidimensional space the trainings last longer than in univariate space. When having images as an input the training takes longer than when working with features; even the size of the image can greatly make the training time differ. The Neural Network used at the backend will also influence the training time needed. The size of the state space will influence the length of the training, if it is bigger, it will take longer.
Of course the computation resources and the types of algorithms run will also influence how long a training will last. If GPUs are available and the algorithm used supports multithreading, the process will be accelerated.

Time to result can also be understood as a requirement for testing: the AI has to be faster than 'x' seconds to reach the goal. To avoid misunderstandings, this definition would be used as an accuracy requirement in the training phase.

For the TEM use case, there is not much that can be influenced in the setup to fasten the training. Given that 3 images of size 224x224 are used as an input to a ResNet18 backbone, and given that DQN of baselines3 is used as an algorithm that can be run on a single GPU, and given that the dimensionality of the state space varies with the stage of development in the research and that the dimensions are given by the problem at hand, the interval of play to fasten this up is limited.

In this case it is more important to have an effective communication process and loop between the DT side and the AI side, and to avoid mistakes to be able to move forwards.

For example, if the data used for training does not converge in training it is important to communicate the kind of failure, to understand the cause and to be able to make another iteration. Making false assumptions or conclusion on the root cause is often more detrimental to the development on the AI side than anything else mentioned above. Speeding up any factor in the AI side with all the givens is less of a priority than having an efficient communication loop with the AI side.

Commented [D(122]: Can this be made bigger/readable?

Commented [v(123]: From reviewers: Not readable

It is hard how to establish definite requirements for the described process.

In the context of the UUV use case, the execution time within the unmanned underwater vehicle (UUV) is not of primary concern. This is because the primary objective is to generate critical test cases for optimizing the behaviour of the UUV. What holds greater significance in this UUV use case is the RL algorithm's ability to produce potential critical scenarios. These scenarios serve as valuable test cases that will be evaluated in a real test bed environment. The availability of effective critical scenarios generated by RL can lead to a reduction in the costly usage of the test bed.

- Scalability

The scalability as referred to in 2.3 is a kind of robustness across machines. In training the only robustness that will be important is as defined in 'Robustness'.

Commented [SL(vd124]: I would expect here something about scaling to different configurations of microscopes for the TEM case. When does the solution generalize to a broader family of systems?

Commented [D(125R124]: Define 'configuration' but the current model only works for a very specific type of probe corrected STEM. Other machines have other lenses and/or other ways to formate an image.

Another aspect of scalability, as seen in the UUV use case, pertains to the adaptability of the RL solution, making it straightforward to expand for larger use cases. This adaptability can be accomplished by establishing manageable and isolated micro-service structures for each component involved in RL training and operations. Such a structure offers flexibility in terms of combining and extending these components while also granting freedom in determining how they interact with each other.

- Explainability

Explainability is an important factor in training, as failing to explain behaviour will result in not understanding how to improve the AI step by step in a controlled manner. Without an understanding there is no way in developing of the solution towards a desired behaviour, and in AI many iterations will be needed to build the solution in complexity and towards the desired requirements.

As importantly towards building the solution, knowing why the AI works and why it fails is needed to build the human trust to build in the software into a system, to convince stakeholders and people involved.

Some tools and metrics can be used to tracked the behavior, depending on the type of AI this will vary.
In the TEM use case, we use DQN, a value-function based RL with Neural Networks for function approximation. In this specific case, using the q_values and sum(q_values), together with analyzing the action maps taken gives almost full explainability of the policy. Knowledge and interpretation are needed to make the just conclusions, as many factors can influence how the q_values and the actions taken look like.

For example, it can be factors in the data, in the hyperparameters used during training, in the design of the problem, on the reward function definition in the RL.

To enhance trust in the automatic solutions generated by RL algorithms and to mitigate potential modeling errors, it is advisable to involve experts in the evaluation process, preferably on a sample-by-sample basis. This kind of testing is particularly valuable in RL applications where ground truth data is unavailable, as is often the case in scenarios like UUV1.

Similar as in discussed in the reliability and reproducibility, it is crucial to employ tools for tasks such as data labeling and process labeling, documenting the training and operational processes, and version control of the RL model. These tools should be complemented by a data analytics and management platform that enables thorough analysis of the resulting information by an expert.

6.2.2 AI subsystem – operational requirements.

In the operational phase, as an extension to the training phase when successful, all the requirements that needed to be approximate first, need now to be more definite.

For deployment upon a real system, the hard requirements need to be specified and the solution needs to be finetuned in order to function as described in all aspects of the requirements.

The solution in training phase and operational phase should be the same, as varying the conditions may result in needing another iteration on the training phase. The AI software is known to work very well for a very specific range of conditions, so varying them may result in another iteration. The extent of the changes in data or requirements will determine the extent of the rework in the training phase. In general, if the changes are small, the method might be correct, but the models will need to be retrained, possibly needing other processing, hyperparameters or length of training. What makes a small or big change is up to the experts to decide, when the changes of requirements are too large not even the approach can be assumed to be right anymore. So, having changes in the requirements is especially critical for AI systems, that need a good definition of the problem beforehand. In training those requirements need be less accurate, but in big lines, correct. Assuring this for a smooth stepping to deployment is the key.

For the electron microscope the first months there was no specific business-case (but a general problem statement) and a more generic approach to the problem was taken. When the business case was ready the approach was still holding even though the hyperparameter tuning was the key to solve the problem.

The same also holds for UUV Use Case. In the operational phase, it is important to monitor the performance of the RL-Model in hindsight whether it achieves the intended goal of creating critical scenarios. Here, fully documentation of the processes and the possibility of user interacting with the resulting data is important for the users to evaluate and adjust the model for possible changes. Both the model-logging infrastructure (Figure 69) and the data management (Figure 70) required in the training of the model is helpful to fulfill those tasks.

6.3 Functional view

In the case of the AI subsystem, we decompose the functional view into the training phase and the operational or deployment phase.

For ASIMOV the training phase comprises the typical RL blocks, which means that the agent interacts with the own defined environment. That environment is based on the gym-framework, which enables the agent to go to through the sequential steps of the problem. The steps followed are defined by us following the logic of the problem. The RL algorithm used will update its policy and learn a (desired) behaviour in the training process. Depending on the RL algorithm used, the method of learning and updating the policy will differ. In the next section the differences when updating the policy will be highlighted. In this section, through the graph, the learning process can be visualized in Figure 71.

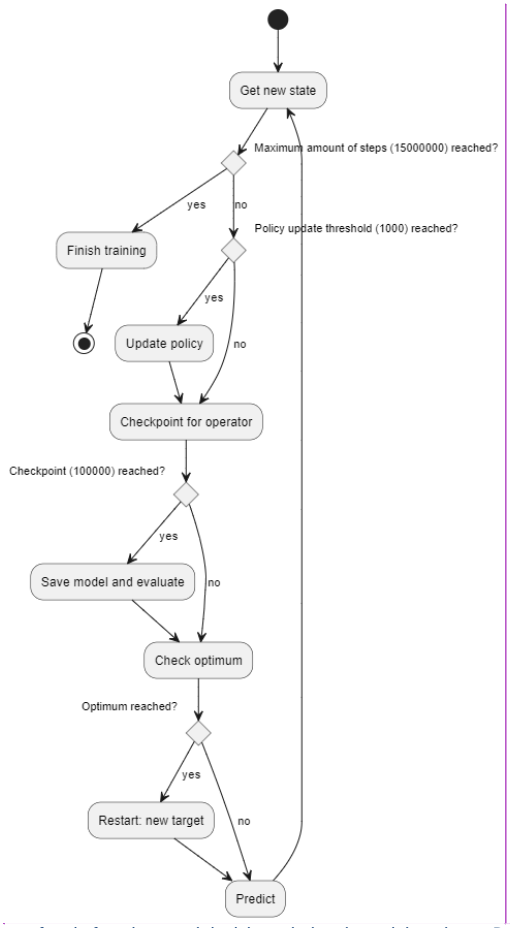


Figure 71 Flow diagram of main functions and decisions during the training phase. Decision points contain exemplary numbers.

The training phase generally consists of three nested loops. The lowest loop is the obtaining a state. predicting the next action, obtaining the next state and so on. This loop is terminated when an optimal state is reached. The sequence of steps is often referred to as an episode. The superordinate loop iterates through episodes. After a certain number of episodes or states across the performed episodes, the policy is updated. The highest order loop then assesses the state of the training which consists of iteratively updating the policy (decision: “Policy update threshold?”). It is generally stopped after a predefined number of policy updates or the evaluation of the training progress on a validation test set.
In the operational phase no more learning is done, the learned behaviour is the one that will be deployed on the machine. For this, the trained policy encoded into e.g., a neural net, is used as is without any further updating. To do so, one either can decouple the artefact from the training algorithm (see Figure 72 or fix the exploration of the algorithm. For value-function based algorithms like DQN, if the exploration rate is 0 the same behaviour will always be observed for a given (estimated) state: the behaviour is deterministic. If the exploration rate is 0.1, one in the ten moves the action might differ from the mandated policy. This behaviour will be chosen from the selected model in training.

Commented [M(126]: Is this the latest version? This seems to have fewer loops in it than previous versions.

Commented [M(127]: What is the "policy update threshold"?

Commented [M(128]: I believe this is not the case for actor-critic. You don't set an exploration rate in this case.

Commented [A(129R128]: You are right, how you do this for actor-critic then? I guess there is exploration/exploitation in place but somehow implicitly encoded?

In the EM use-case the model will be interacting with the calibration software of the microscope in an automated way until it reaches the desired position, disengaging at that point.

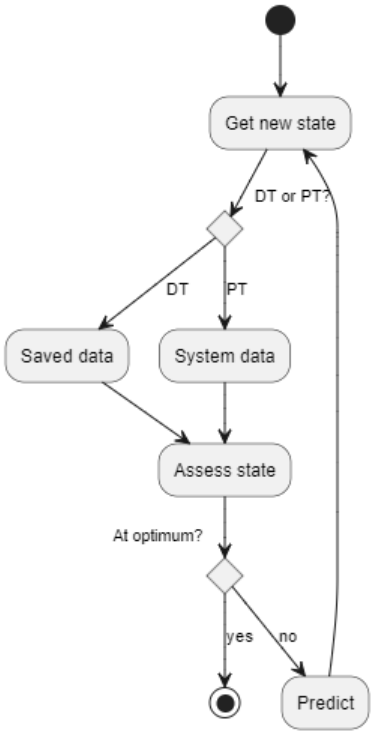


Figure 72 Flow diagram of main functions and decisions during the operational phase. This represents mainly the evaluation flow.

6.4 Logical view

This is the last view that will be developed, given that the technical view would refer to the connection to the embedded software or hardware of the physical devices itself, that is out of scope for this document.

In the logical view the detail of the chosen implementation and how the policies are updated per RL algorithm family will be highlighted. The two families are namely value-based and actor-critic. At this level, there are implementation specific choices that are unavoidable at this level of depth.

Commented [M(130]: Is this true?

Commented [A(131R130]: Should be value-based instead of DQN, I fixed it

From the AI perspective, it is interesting to understand what we called the 'Predict' block and to understand how the policy is updated in RL, what we called the 'Update policy' block. Those are the AI specific elements of the flows.

For the EM usecase, applying a DQN algorithm (value-based), the 'Predict' block is implemented as follows:

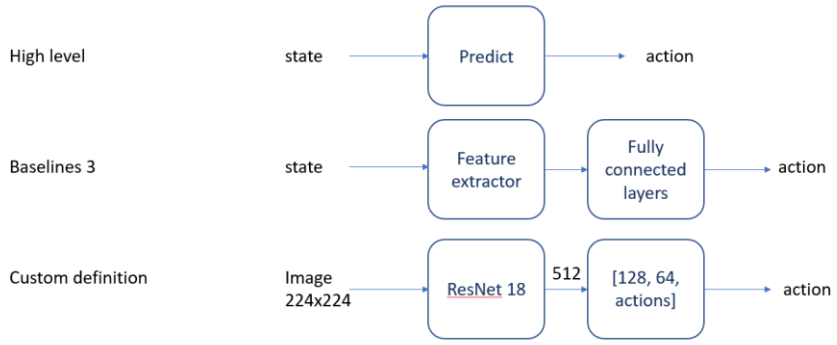


Figure 73

From the perspective of the policy update a vanilla DQN implementation was used, also from the library baselines3. DQN is a value-based model free RL algorithm that works according to working in the graph below.
The Q-network corresponds to the custom-defined network defined above. The Target Q-network in a copy of the network that is only updated every policy update times, as can be seen in graph x in the functional view.

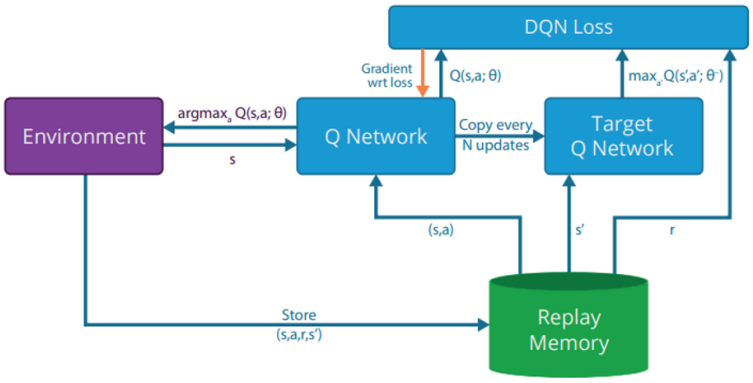


Figure 74

In the UUV case which implements an actor-critic algorithm (see chapter 4), the Predict block contains a neural network called “the actor” with the following dimensions:

(14 states)x128 linear + relu activation layer + 128x128 + relu activation layer + 128x(9 discrete options + 3 discrete options + 6 for continuous action averages + 21 for creating cholesky matrix for continuous actions = 36)

The policy updating consists of multiple steps Figure 73. To estimate the Q-value of a specific action-state-pair the critic is also a neural net with size.

3 layers: (14states + 10 actions)x128 linear + 128x128 linear + 128x1 linear

Afterwards, a gradient-based, two-step process needs to be performed. The E-step is to set the weights for the action, and the M-step is used to fit the improved policy (for more details, consult D1.3.1).

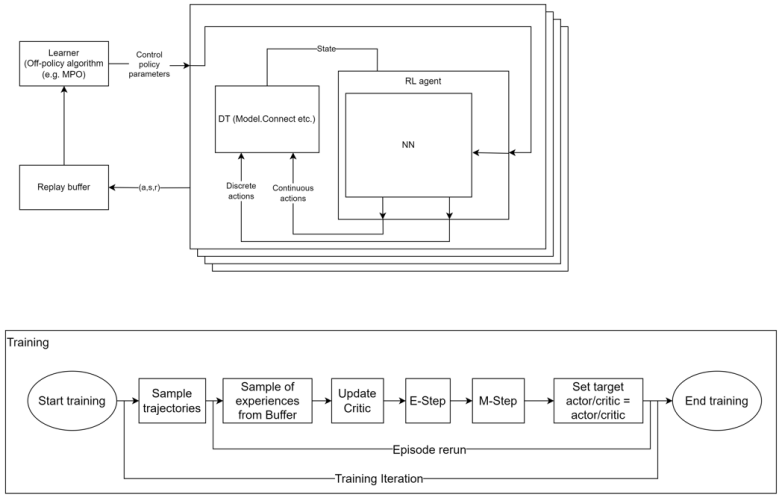


Figure 75 Diagrams of the actor-critic algorithm implemented in the UUV use case consisting of function blocks and steps.

DQN and MPO are both off-policy in this case, using a replay buffer to sample trajectories, even though there are many ways of sampling, and this is also algorithm specific.

7 Conclusion

This report describes the state-of-the-art in reinforcement learning and digital twin-based learning, the first ideas on their application in the ASIMOV use cases, as well as the full AI implementations in the ASIMOV use cases. The final contribution included in this report is a reference architecture for DT-supported AI implementations.

The extensive and well-structured literature overview provides ample information to direct the investigations in applying state-of-the-art techniques in both digital twinning as well as artificial intelligence. Chapters 4 and 5 detail the specific AI approaches and designs applied in the use cases, mostly in chronological order, with results. The chapters present theory behind changes made to off-the-shelf AI tools to ensure a correct fit to the use case. The reference AI architecture in Chapter 6 summarizes the general lessons learned in the ASIMOV project regarding DT-based AI.

The AI applications have obtained varying levels of success, with the main hurdle at the end of the project being the complexity of the cyber-physical system. The most promising results suggest that DT-based AI has great potential as a product to enhance the performance of cyber-physical systems, on the conditions that domain-specific challenges can be overcome, and that the problem fits the AI architecture.

Bibliography

[1] Wright, L., Davidson, S. How to tell the difference between a model and a digital twin. *Adv. Model. and Simul. in Eng. Sci.* **7,** 13 (2020). https://doi.org/10.1186/s40323-020-00147-4

[2] D. Bertsekas '*Reinforcement learning and Optimal control*', 2019.

[3] R.S. Sutton and A.G. Barto, '*Reinforcement learning: an introduction*', 2018

[4] P. Marbach and J. N. Tsitsiklis, "Simulation-based optimization of Markov reward processes," in *IEEE Transactions on Automatic Control*, 2001.

[5] *Silver, D., Huang, A., Maddison, C. et al. Mastering the game of Go with deep neural networks and tree search. Nature 529, 484–489 (2016)

[6] Marc Peter Deisenroth Carl Edward Rasmussen PILCO: A Model-Based and Data-Efficient Approach to Policy Search, 2011

[7] Hyeong Soo Chang, Michael C. Fu, Jiaqiao Hu, and Steven I. Marcus An Adaptive Sampling Algorithm for Solving Markov Decision Processes, Operations Research 2005 53:1, 126-139

[8] Hyeong Soo Chang, Jiaqiao Hu, Michael C. Fu, Steven I. Marcus, Simulation-Based Algorithms for Markov Decision Processes, 2013

[9] C. B. Browne *et al*., "A Survey of Monte Carlo Tree Search Methods," in *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1-43, March 2012, doi: 10.1109/TCIAIG.2012.2186810.

[10] Gerald Tesauro, G. R. Galperin, 1996, On-line Policy Improvement using Monte-Carlo Search, NIPS, Denver, CO

[11] Bertsekas, D. P. 2005, Rollout algorithms for Constrained Dynamic Programming, LIDS report 2646, MIT.

[12] Sutton, R. S. (1990) Integrated architectures for learning, planning and reacting based on approximating dynamic programming, In proceedings of the 7th International Workshop on Machine Learning, pp. 216-224, Morgan Kaufmann.

[13] Sutton, R. S. (1991) Dyna, an integrated architecture for learning, planning and reacting, SIGART Bulletin 2(4):160-163, ACM, New York.

[14] *Francois-Lavet, Vincent; et al. (2018). "An Introduction to Deep Reinforcement Learning". Foundations and Trends in Machine Learning.* ***11****(3–4): 219–354. arXiv:1811.12560.*

[15] Kaishu Xia, Christopher Sacco, Max Kirkpatrick, Clint Saidy, Lam Nguyen, Anil Kircaliali, Ramy Harik, A digital twin to train deep reinforcement learning agent for smart manufacturing plants: Environment, interfaces and intelligence, Journal of Manufacturing Systems, Volume 58, Part B, 2021, Pages 210-230,

[16] Yueyue Dai, Ke Zhang, Sabita Maharjan, Yan Zhang Deep Reinforcement Learning for Stochastic Computation Offloading in Digital Twin Networks, 2020, available at https://arxiv.org/abs/2011.08430

[17] Marius Matulisa, Carlo Harvey, A robot arm digital twin utilising reinforcement learning, Computers & Graphics 95 (2021) 106–114

[18] Huang,Z.;Shen,Y.;Li,J.; Fey, M.; Brecher, C. A Survey on AI-Driven Digital Twins in Industry 4.0: Smart Manufacturing and Advanced Robotics. *Sensors* **2021**, *21*, 6340. https://doi.org/10.3390/s21196340

[19] *W. Kuehn,* Digital twins for Decision making in complex production and logistic enterprises. *Int. J. of Design & Nature and Ecodynamics. Vol. 13, No. 3 (2018) 260–271*

[20] Kosmas Alexopoulos, Nikolaos Nikolakis & George Chryssolouris (2020) Digital twin-driven supervised machine learning for the development of artificial intelligence applications in manufacturing, International Journal of Computer Integrated Manufacturing, 33:5, 429-439,DOI: 10.1080/0951192X.2020.1747642

[21] Aske Plaat and Walter Kosters and Mike Preuss, Deep Model-Based Reinforcement Learning for High-Dimensional Problems, a Survey, 2020, arXiv 2008.05598.

[22] Gerald Tesauro. Temporal difference learning and TD-Gammon. Communica- tions of the ACM, 38(3):58–68, 1995.

[23] Thomas Anthony, Zheng Tian, and David Barber. Thinking fast and slow with deep learning and tree search. In Advances in Neural Information Processing Systems, pages 5360–5370, 2017

[24] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja, Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of Go without human knowledge. Nature, 550(7676):354, 2017

[25] Dieqiao Feng, Carla P Gomes, and Bart Selman. Solving hard AI planning instances using curriculum-driven deep reinforcement learning. arXiv preprint arXiv:2006.02689, 2020

[26] Yuval Tassa, Tom Erez, and Emanuel Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. In 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 4906–4913, 2012.

[27] Sergey Levine and Pieter Abbeel. Learning neural network policies with guided policy search under unknown dynamics. In Advances in Neural Information Processing Systems, pages 1071–1079, 2014.

[28] Nicolas Heess, Gregory Wayne, David Silver, Timothy Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. In Advances in Neural Information Processing Systems, pages 2944–2952, 2015.

[29] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In Advances in Neural Information Processing Systems, pages 4754–4765, 2018

[30] Chelsea Finn and Sergey Levine. Deep visual foresight for planning robot mo- tion. In 2017 IEEE International Conference on Robotics and Automation (ICRA), pages 2786–2793, 2017

[31] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep Q-learning with model-based acceleration. In International Conference on Machine Learning, pages 2829–2838, 2016

[32] Vladimir Feinberg, Alvin Wan, Ion Stoica, Michael I Jordan, Joseph E Gonzalez, and Sergey Levine. Model-based value estimation for efficient model-free reinforcement learning. arXiv preprint arXiv:1803.00101, 2018

[33] Ignasi Clavera, Jonas Rothfuss, John Schulman, Yasuhiro Fujita, Tamim Asfour, and Pieter Abbeel. Model-based reinforcement learning via meta-policy optimization. arXiv preprint arXiv:1809.05214, 2018

[34] Kamyar Azizzadenesheli, Brandon Yang, Weitang Liu, Emma Brunskill, Zachary C Lipton, and Animashree Anandkumar. Surprising neg- ative results for generative adversarial tree search. arXiv preprint arXiv:1806.05780, 2018

[35] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. In Advances in Neural Information Processing Systems, pages 12498–12509, 2019

[36] Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard L Lewis, and Satinder Singh. Action-conditional video prediction using deep networks in Atari games. In Advances in Neural Information Processing Systems, pages 2863– 2871, 2015.

[37] Junhyuk Oh, Satinder Singh, and Honglak Lee. Value prediction network. In Advances in Neural Information Processing Systems, pages 6118–6128, 2017

[38] Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Koza- kowski, Sergey Levine, et al. Model-based reinforcement learning for Atari.

[39] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. arXiv preprint arXiv:1811.04551, 2018.

[40] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. arXiv preprint arXiv:1912.01603, 2019. arXiv preprint arXiv:1903.00374, 2019.

[41] Ramanan Sekar, Oleh Rybkin, Kostas Daniilidis, Pieter Abbeel, Danijar Hafner, and Deepak Pathak. Planning to explore via self-supervised world models. arXiv preprint arXiv:2005.05960, 2020

[42] Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. In Adv. in Neural Information Processing Systems, pages 2154–2162, 2016

[43] Nantas Nardelli, Gabriel Synnaeve, Zeming Lin, Pushmeet Kohli, Philip HS Torr, and Nicolas Usunier. Value propagation networks. arXiv preprint arXiv:1805.11199, 2018

[44] Gregory Farquhar, Tim Rockta¨schel, Maximilian Igl, and SA Whiteson. TreeQN and ATreeC: Differentiable tree planning for deep reinforcement learning. International Conference on Learning Representations, 2018

[45] Arthur Guez, Mehdi Mirza, Karol Gregor, Rishabh Kabra, Sebastien Racaniere, Theophane Weber, David Raposo, Adam Santoro, Laurent Orseau, Tom Eccles, et al. An investigation of model-free planning. arXiv preprint arXiv:1901.03559, 2019

[46] T. Weber, Sebastien Racanie`re, David Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adria Puigdomenech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, et al. Imagination-augmented agents for deep reinforcement learning. In Advances in Neural Information Processing Systems, pages 5690–5701, 2017

[47] David Silver, Hado van Hasselt, Matteo Hessel, Tom Schaul, Arthur Guez, Tim Harley, Gabriel Dulac-Arnold, David Reichert, Neil Rabinowitz, Andre Barreto, et al. The predictron: End-to-end learning and planning. In Proceedings of the 34th International Conference on Machine Learning, pages 3191–3199, 2017b

[48] David Ha and Jurgen Schmidhuber. World models. arXiv preprint arXiv:1803.10122, 2018b

[49] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering Atari, Go, chess and shogi by planning with a learned model. arXiv preprint arXiv:1911.08265, 2019

[50] Shohin Aheleroff, Xun Xu, Ray Y Zhong, Yuqian Lu, Digital Twin as a Service (DTaaS) in Industry 4.0: An Architecture Reference Model, Advanced Engineering Informatics, 47, 2021

[51] Stein Ove Erikstad, Merging Physics, Big Data Analytics and Simulation for the Next-Generation Digital Twins, Conference paper, HIPER 2017, High-Performance Marine Vehicles, Zevenwacht, South-Africa, 11-13 September 2017.

[52] https://www.kdnuggets.com/2019/09/6-tips-training-data-strategy-machine-learning.html

[53] Laura von Rueden, et. al. Informed Machine Learning – A Taxonomy and Survey of Integrating Prior Knowledge into Learning Systems, IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, 2021, arXiv:1903.12394v3

[54] Abele D., D'Onofrio S. (2020) Artificial Intelligence – The Big Picture. In: Portmann E., D'Onofrio S. (eds) Cognitive Computing. Edition Informatik Spektrum. Springer Vieweg, Wiesbaden. https://doi.org/10.1007/978-3-658-27941-7_2

[55] https://en.wikipedia.org/wiki/IDEF0

[56] Peter Stehouwer, Dick den Hertog, *First ASMO UK / ISSMO Conference on Engineering Design Optimization*, SIMULATION-BASED DESIGN OPTIMISATION: METHODOLOGY AND APPLICATIONS , http://www.asmo-uk.com/1st-asmo-uk/html/menu_page.html or https://cqm.nl/uploads/media/5e414fa60e2eb/simulation-based-design-optmisation-methodology-and-applications-asmo1999.pdf

[57] CQM whitepaper, Erwin Stinstra et al, DESIGN OPTIMISATION: SOME PITFALLS AND THEIR REMEDIES, https://cqm.nl/uploads/media/5e426cf3b1f1c/design-optimisation-some-pitfalls-and-their-remedies.pdf

[58] Conference proceedings, CQM, Structural mass optimization of the engine frame of the Ariane 5 ESC-B, https://cqm.nl/uploads/media/5e41530c71083/ariane-5-e847f0d6d01.pdf

[59] CQM whitepaper, DoCE for an optimal high voltage tube, https://cqm.nl/uploads/media/61eea408d39f4/cqm-doce-for-an-optimal-high-voltage-tube.pdf

[60] Rezaeifar, S., Dadashi, R., Vieillard, N., Hussenot, L., Bachem, O., Pietquin, O., & Geist, M. (2021). Offline Reinforcement Learning as Anti-Exploration. *arXiv preprint arXiv:2106.06431*.

[61] Dadashi, R., Rezaeifar, S., Vieillard, N., Hussenot, L., Pietquin, O., & Geist, M. (2021). Offline Reinforcement Learning with Pseudometric Learning. *arXiv preprint arXiv:2103.01948*.

[62] Lagoudakis, M. G. and Parr, R. Least-squares policy iteration. Journal of Machine Learning Research, 2003

[64] Lange, S., Gabel, T., and Riedmiller, M. Batch reinforcement learning. In Reinforcement learning.

[65] Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., & Anandkumar, A. (2020). Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*.

[66] Thuerey, N., Holl, P., Mueller, M., Schnell, P., Trost, F., & Um, K. (2021). Physics-based Deep Learning. *arXiv preprint arXiv:2109.05237*.

[67] Sanchez-Gonzalez, A., Godwin, J., Pfaff, T., Ying, R., Leskovec, J., & Battaglia, P. (2020, November). Learning to simulate complex physics with graph networks. In *International Conference on Machine Learning* (pp. 8459-8468). PMLR.

[68] Chu, M., & Thuerey, N. (2017). Data-driven synthesis of smoke flows with CNN-based feature descriptors. *ACM Transactions on Graphics (TOG)*, *36*(4), 1-14.

[69] Thuerey, N., Weißenow, K., Prantl, L., & Hu, X. (2020). Deep learning methods for Reynolds-averaged Navier–Stokes simulations of airfoil flows. *AIAA Journal*, *58*(1), 25-36.

[70] Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2017). Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations. *arXiv preprint arXiv:1711.10561*.

[71] Kiwon Um, Robert Brand, Philipp Holl, Raymond Fei, and Nils Thuerey. Solver-in-the-loop: learning from differentiable physics to interact with iterative pde-solvers. *Advances in Neural Information Processing Systems*, 2020

[72] Shrivastava, Ashish, et al. "Learning from simulated and unsupervised images through adversarial training." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017.

[73] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, Image-to-Image Translation with Conditional Adversarial Networks. ArXiv, 2016

[74] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, Generative Adversarial Nets. *Proceedings Neural Information Processing Systems Conference*, 2014

[75] Chen, Ting, et al. "A simple framework for contrastive learning of visual representations." *International conference on machine learning*. PMLR, 2020.

[76] Caron, Mathilde, et al. "Emerging properties in self-supervised vision transformers." *arXiv preprint arXiv:2104.14294* (2021).

[77] Caron, Mathilde, et al. "Emerging properties in self-supervised vision transformers." *arXiv preprint arXiv:2104.14294* (2021).

[78] https://www.tietoevry.com/en/blog/2019/12/robust-and-scalable-ml-lifecycle-for-a-high-performing-ai-team/

[79] https://docs.microsoft.com/en-us/learn/paths/build-ai-solutions-with-azure-ml-service/

[80] https://aws.amazon.com/sagemaker

[81] https://cloud.google.com/ai-platform/docs/technical-overview

[82] https://mlflow.org

[83] https://www.tensorflow.org/tfx

[84] ASIMOV-consortium, ASIMOV - Full Project Proposal, 2020.

[85] https://www.bons.ai

[86] ASIMOV deliverable, IR1.1 - ASIMOV_Specifications_and_Commonality_Analysis_V1.pdf, 2021.
[89] Ni, Tianwei, Benjamin Eysenbach, and Ruslan Salakhutdinov. "Recurrent model-free rl can be a strong baseline for many pomdps." *arXiv preprint arXiv:2110.05038* (2021).
[90] Morad, Steven, et al. "Popgym: Benchmarking partially observable reinforcement learning." *arXiv preprint arXiv:2303.01859* (2023).

[91] K. Pohl, M. Broy, H. Daembkes and H. Hönninger, Advanced Model-Based Engineering of Embedded Systems, Springer International Publishing, 2016

[A1] Westhofen, L., Neurohr, C., Koopmann, T., Butz, M., Schütt, B., Utesch, F., ... & Böde, E. (2021). Criticality metrics for automated driving: A review and suitability analysis of the state of the art. arXiv preprint arXiv:2108.02403.
[A2] Degrave, J., Felici, F., Buchli, J., Neunert, M., Tracey, B., Carpanese, F., ... & Riedmiller, M. (2022). Magnetic control of tokamak plasmas through deep reinforcement learning. Nature, 602(7897), 414-419. https://doi.org/10.1038/s41586-021-04301-9.

[A3] Abdolmaleki, A., Springenberg, J. T., Tassa, Y., Munos, R., Heess, N., & Riedmiller, M. (2018). Maximum a posteriori policy optimisation. arXiv preprint arXiv:1806.06920.
[A4] Abdolmaleki, A., Springenberg, J. T., Degrave, J., Bohez, S., Tassa, Y., Belov, D., ... & Riedmiller, M. (2018). Relative entropy regularized policy iteration. arXiv preprint arXiv:1812.02256.
[A5] Neunert, M., Abdolmaleki, A., Wulfmeier, M., Lampe, T., Springenberg, T., Hafner, R., ... & Riedmiller, M. (2020, May). Continuous-discrete reinforcement learning for hybrid control in robotics. In Conference on Robot Learning (pp. 735-751). PMLR.
[A6] Wurman, P. R., Barrett, S., Kawamoto, K., MacGlashan, J., Subramanian, K., Walsh, T. J., ... & Kitano, H. (2022). Outracing champion Gran Turismo drivers with deep reinforcement learning. Nature, 602(7896), 223-228.
[A7] Masson, W., Ranchod, P., & Konidaris, G. (2016, February). Reinforcement learning with parameterized actions. In Thirtieth AAAI Conference on Artificial Intelligence.
[A8] Hausknecht, M., & Stone, P. (2015). Deep reinforcement learning in parameterized action space. arXiv preprint arXiv:1511.04143.
[A9] Hoffman, M., Shahriari, B., Aslanides, J., Barth-Maron, G., Behbahani, F., Norman, T., ... & de Freitas, N. (2020). Acme: A research framework for distributed reinforcement learning. arXiv preprint arXiv:2006.00979.
[A10] Arnold, A., Nallapati, R., & Cohen, W. W. (2007, October). A comparative study of methods for transductive transfer learning. In Seventh IEEE international conference on data mining workshops (ICDMW 2007) (pp. 77-82). IEEE.
[A11] Torrey, L., & Shavlik, J. (2010). Transfer learning. In Handbook of research on machine learning applications and trends: algorithms, methods, and techniques (pp. 242-264). IGI global.
[A12] Neyshabur, B., Sedghi, H. & Zhang, C. (2020). What is being transferred in transfer learning?. In Advances in Neural Information Processing Systems (pp. 512-523).[13] Sutton, R. S., Precup, D., & Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. Artificial intelligence, 112(1-2), 181-211.
[A13] Croonenborghs, T., Driessens, K., & Bruynooghe, M. (2007, June). Learning relational options for inductive transfer in relational reinforcement learning. In International Conference on Inductive Logic Programming (pp. 88-97). Springer, Berlin, Heidelberg.
[A15] Walsh, T. J., Li, L., & Littman, M. L. (2006, June). Transferring state abstractions between MDPs. In ICML Workshop on Structural Knowledge Transfer for Machine Learning.
[A16] Taylor, M. E., & Stone, P. (2007, June). Cross-domain transfer for reinforcement learning. In Proceedings of the 24th international conference on Machine learning (pp. 879-886).
[A17] Konidaris, G., & Barto, A. (2006, June). Autonomous shaping: Knowledge transfer in reinforcement learning. In Proceedings of the 23rd international conference on Machine learning (pp. 489-496).
[A18] Price, B., & Boutilier, C. (1999, June). Implicit imitation in multiagent reinforcement learning. In ICML (pp. 325-334).
[A19] Torrey, L., Shavlik, J., Walker, T., & Maclin, R. (2006). Relational skill transfer via advice taking. In ICML Workshop on Structural Knowledge Transfer for Machine Learning.
[A20] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang and W. Zaremba, OpenAI Gym, 2016.
[A21] MLflow, “MLflow Website,” [Online]. Available: https://mlflow.org/. [Accessed 23 11 2022].
[A22] Z. Chen, C. K. Yeo, B. S. Lee and C. T. Lau, "Autoencoder-based network anomaly detection," 2018 Wireless Telecommunications Symposium (WTS), Phoenix, AZ, USA, 2018, pp. 1-5, doi: 10.1109/WTS.2018.8363930.

[B1] Franken, L. E., Grünewald, K., Boekema, E. J., Stuart, M. C. A., A Technical Introduction to Transmission Electron Microscopy for Soft-Matter: Imaging, Possibilities, Choices, and Technical Developments. *Small* 2020, 16, 1906198. https://doi.org/10.1002/smll.201906198
[B2] Kirkland, Earl J. *Advanced computing in electron microscopy*. Vol. 12. New York: Plenum Press, 1998.
[B3] *Schnitzer, N.; Sung, S.H.; Hovden, R.H. (2019). "Introduction to the Ronchigram and its Calculation with Ronchigram". Microscopy Today. **3**: 12–15.*
[B4] He, Kaiming, et al. "Deep residual learning." *Image Recognition* (2015).
[B5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, ImageNet: A Large-Scale Hierarchical Image Database**.** *IEEE Computer Vision and Pattern Recognition (CVPR), 2009*
[B6] Gidaris, S., Singh, P., & Komodakis, N. (2018). Unsupervised representation learning by predicting image rotations. *arXiv preprint arXiv:1803.07728*.

[B7] Doersch, C., Gupta, A., & Efros, A. A. (2015). Unsupervised visual representation learning by context prediction. In *Proceedings of the IEEE international conference on computer vision* (pp. 1422-1430).
[B8] Ouali, Y., Hudelot, C., & Tami, M. (2020). An overview of deep semi-supervised learning. *arXiv preprint arXiv:2006.05278*.
[B9] Van Engelen, J. E., & Hoos, H. H. (2020). A survey on semi-supervised learning. *Machine Learning*, *109*(2), 373-440.
[B10] Chopra, S., Hadsell, R., & LeCun, Y. (2005, June). Learning a similarity metric discriminatively, with application to face verification. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)* (Vol. 1, pp. 539-546). IEEE.
[B11] Chuang, C. Y., Robinson, J., Lin, Y. C., Torralba, A., & Jegelka, S. (2020). Debiased contrastive learning. *Advances in neural information processing systems*, *33*, 8765-8775.
[B12] Robinson, J., Chuang, C. Y., Sra, S., & Jegelka, S. (2020). Contrastive learning with hard negative samples. *arXiv preprint arXiv:2010.04592*.
[B13] Chen, T., Kornblith, S., Norouzi, M., & Hinton, G. (2020, November). A simple framework for contrastive learning of visual representations. In *International conference on machine learning* (pp. 1597-1607). PMLR.
[B14] Ganin, Y., Ustinova, E., Ajakan, H., Germain, P., Larochelle, H., Laviolette, F., ... & Lempitsky, V. (2016). Domain-adversarial training of neural networks. *The journal of machine learning research*, *17*(1), 2096-2030.
[B15] Grill, J. B., Strub, F., Altché, F., Tallec, C., Richemond, P., Buchatskaya, E., ... & Valko, M. (2020). Bootstrap your own latent-a new approach to self-supervised learning. Advances in Neural Information Processing Systems, 33, 21271-21284.
[B16] Huang, X., & Belongie, S. (2017). Arbitrary style transfer in real-time with adaptive instance normalization. In *Proceedings of the IEEE international conference on computer vision* (pp. 1501-1510).
[B17] https://en.wikipedia.org/wiki/Domain_adaptation
[B18] Tousch, A, C Renaudin, Christophe, (Yet) Another Domain Adaptation library, 2020
[B19] Long, M., Cao, Z., Wang, J., & Jordan, M. I. (2018). Conditional adversarial domain adaptation. *Advances in neural information processing systems*, *31*.
[B20] Song, L., Huang, J., Smola, A., & Fukumizu, K. (2009, June). Hilbert space embeddings of conditional distributions with applications to dynamical systems. In Proceedings of the 26th Annual International Conference on Machine Learning (pp. 961-968).
[B21] *Kantorovich LV (1939). "Mathematical Methods of Organizing and Planning Production". Management Science.* ***6*** *(4): 366–422. doi:10.1287/mnsc.6.4.366. JSTOR 2627082*
[B22] Shen, J., Qu, Y., Zhang, W., & Yu, Y. (2018, April). Wasserstein distance guided representation learning for domain adaptation. In *Thirty-second AAAI conference on artificial intelligence*.
[B23] Gretton, A., Borgwardt, K. M., Rasch, M. J., Schölkopf, B., & Smola, A. (2012). A kernel two-sample test. *The Journal of Machine Learning Research*, *13*(1), 723-773.
[B24] Sun, B., & Saenko, K. (2016, October). Deep coral: Correlation alignment for deep domain adaptation. In *European conference on computer vision* (pp. 443-450). Springer, Cham.
[B25] Long, M., Cao, Y., Wang, J., & Jordan, M. (2015, June). Learning transferable features with deep adaptation networks. In *International conference on machine learning* (pp. 97-105). PMLR.
[B26] Long, M., Zhu, H., Wang, J., & Jordan, M. I. (2017, July). Deep transfer learning with joint adaptation networks. In *International conference on machine learning* (pp. 2208-2217). PMLR.
[B27] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.
[B28] Van Hasselt, H., Guez, A., & Silver, D. (2016, March). Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence* (Vol. 30, No. 1).
[B29] Degrave, J., Felici, F., Buchli, J., Neunert, M., Tracey, B., Carpanese, F., & Riedmiller, M. (2022). Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, *602*(7897), 414-419.
[B30] Lanctot, M., Zambaldi, V., Gruslys, A., Lazaridou, A., Tuyls, K., Pérolat, J., ... & Graepel, T. (2017). A unified game-theoretic approach to multiagent reinforcement learning. *Advances in neural information processing systems*, *30*.
[B31] Sepp Hochreiter, A. Steven Younger, and Peter R. Conwell. "Learning to learn using gradient descent." Intl. Conf. on Artificial Neural Networks. 2001.
[B32] Wang, J. X., Kurth-Nelson, Z., Tirumala, D., Soyer, H., Leibo, J. Z., Munos, R., ... & Botvinick, M. (2016). Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*.
[B33] Duan, Y., Schulman, J., Chen, X., Bartlett, P. L., Sutskever, I., & Abbeel, P. (2016). Rl $^ 2$: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*.

[B34] Xu, Z., van Hasselt, H. P., & Silver, D. (2018). Meta-gradient reinforcement learning. *Advances in neural information processing systems*, *31*.
[B35] Rezaeifar, S., Dadashi, R., Vieillard, N., Hussenot, L., Bachem, O., Pietquin, O., & Geist, M. (2021). Offline reinforcement learning as anti-exploration. *arXiv preprint arXiv:2106.06431*.
[B36] Gupta, A., Mendonca, R., Liu, Y., Abbeel, P., & Levine, S. (2018). Meta-reinforcement learning of structured exploration strategies. *Advances in neural information processing systems*, *31*.
[B37] Kingma, D. P., & Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.
[B38] Nousiainen, J., Rajani, C., Kasper, M., & Helin, T. (2021). Adaptive optics control using model-based reinforcement learning. *Optics Express*, *29*(10), 15327-15344.
[B39] Chua, K., Calandra, R., McAllister, R., & Levine, S. (2018). Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *Advances in neural information processing systems*, *31*.
[B40] Zoph, B., & Le, Q. V. (2016). Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*.
[B41] Terence Tao, (2016). How to assign partial credit on an exam of true-false questions?
[B42] Nagabandi, A., Kahn, G., Fearing, R. S., & Levine, S. (2017). Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning. CoRR abs/1708.02596 (2017). *arXiv preprint arXiv:1708.02596*.
[B43] Levine, S., Finn, C., Darrell, T., & Abbeel, P. (2015). End-toend training of deep visuomotor policies. arXiv preprint arXiv: 1504.00702.
[B44] Gu, S., Lillicrap, T., Sutskever, I., & Levine, S. (2016, June). Continuous deep q-learning with model-based acceleration. In *International conference on machine learning* (pp. 2829-2838). PMLR.
[B45] Abdolmaleki, A., Springenberg, J. T., Degrave, J., Bohez, S., Tassa, Y., Belov, D., ... & Riedmiller, M. (2018). Relative entropy regularized policy iteration. *arXiv preprint arXiv:1812.02256*.
[B46] Janner, M., Fu, J., Zhang, M., & Levine, S. (2019). When to trust your model: Model-based policy optimization. *Advances in Neural Information Processing Systems*, *32*.
[B47] Wang, T., Bao, X., Clavera, I., Hoang, J., Wen, Y., Langlois, E., ... & Ba, J. (2019). Benchmarking model-based reinforcement learning. *arXiv preprint arXiv:1907.02057*.
[B48] Åström, K. J. (2012). *Introduction to stochastic control theory*. Courier Corporation.
[B49] Levine, S., & Koltun, V. (2013, May). Guided policy search. In *International conference on machine learning* (pp. 1-9). PMLR.
[B50] Nagabandi, A., Kahn, G., Fearing, R. S., & Levine, S. (2017). Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning. CoRR abs/1708.02596 (2017). *arXiv preprint arXiv:1708.02596*.
[B51] Coulom, R. (2006, May). Efficient selectivity and backup operators in Monte-Carlo tree search. In *International conference on computers and games* (pp. 72-83). Springer, Berlin, Heidelberg.
[B52] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., ... & Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *nature*, *529*(7587), 484-489.
[B53] Gu, S., Lillicrap, T., Sutskever, I., & Levine, S. (2016, June). Continuous deep q-learning with model-based acceleration. In *International conference on machine learning* (pp. 2829-2838). PMLR.
[B54] Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine learning proceedings 1990* (pp. 216-224). Morgan Kaufmann.
[B55] Ha, D., & Schmidhuber, J. (2018). World models. *arXiv preprint arXiv:1803.10122*.
[B56] Kaiser, L., Babaeizadeh, M., Milos, P., Osinski, B., Campbell, R. H., Czechowski, K., ... & Michalewski, H. (2019). Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374*.
[B57] Watter, M., Springenberg, J., Boedecker, J., & Riedmiller, M. (2015). Embed to control: A locally linear latent dynamics model for control from raw images. *Advances in neural information processing systems*, *28*.
[B58] Feinberg, V., Wan, A., Stoica, I., Jordan, M. I., Gonzalez, J. E., & Levine, S. (2018). Model-based value estimation for efficient model-free reinforcement learning. *arXiv preprint arXiv:1803.00101*.
[B59] Tamar, A., Wu, Y., Thomas, G., Levine, S., & Abbeel, P. (2016). Value iteration networks. *Advances in neural information processing systems*, *29*.
[B60] Oh, J., Singh, S., & Lee, H. (2017). Value prediction network. *Advances in neural information processing systems*, *30*.

[B61] Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018, July). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning* (pp. 1861-1870). PMLR.
[B62] Achiam J., Held D., Tamar A.,Abbeel P. Constrained Policy Optimization (2017, May). ICML 2017.
[B63] Stooke et al 2020. Responsive Safety in RL by PID Lagrangian Methods.
[B64] Optimal transport for colour. https://dcoeurjo.github.io/OTColorTransfer/

[BB1] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.
[BB2] Long-Ji Lin. Reinforcement learning for robots using neural networks. Technical report, DTIC Document, 1993.
[BB3] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., ... & Kavukcuoglu, K. (2016, June). Asynchronous methods for deep reinforcement learning. In International conference on machine learning (pp. 1928-1937). PMLR.
[BB4] Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., ... & Kavukcuoglu, K. (2018, July). Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In International conference on machine learning (pp. 1407-1416). PMLR.
[BB5] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347.
[BB6] Kempka, M., Wydmuch, M., Runc, G., Toczek, J., & Jaśkowski, W. (2016, September). Vizdoom: A doom-based ai research platform for visual reinforcement learning. In 2016 IEEE conference on computational intelligence and games (CIG) (pp. 1-8). IEEE.
[BB7] Ha, D., & Schmidhuber, J. (2018). World models. arXiv preprint arXiv:1803.10122.
[BB8] Hafner, D., Lillicrap, T., Ba, J., & Norouzi, M. (2019). Dream to control: Learning behaviors by latent imagination. arXiv preprint arXiv:1912.01603.
[BB9] Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., ... & Silver, D. (2020). Mastering atari, go, chess and shogi by planning with a learned model. Nature, 588(7839), 604-609.
[BB10] Hafner, D., Lillicrap, T., Norouzi, M., & Ba, J. (2020). Mastering atari with discrete world models. arXiv preprint arXiv:2010.02193.
[BB11] Hafner, D., Pasukonis, J., Ba, J., & Lillicrap, T. (2023). Mastering diverse domains through world models. arXiv preprint arXiv:2301.04104.
[BB12] Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.
[BB13] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 4510-4520).
[BB14] Arnab, A., Dehghani, M., Heigold, G., Sun, C., Lučić, M., & Schmid, C. (2021). Vivit: A video vision transformer. In *Proceedings of the IEEE/CVF international conference on computer vision* (pp. 6836-6846).
[BB15] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., ... & Houlsby, N. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.
[BB16] Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.
[BB17] Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.
[BB18] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, *30*.
[BB19] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).
[BB20] Machado, M. C., Bellemare, M. G., Talvitie, E., Veness, J., Hausknecht, M., & Bowling, M. (2018). Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, *61*, 523-562.
[BB21] http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf
[BB22] Koller, D., & Friedman, N. (2009). *Probabilistic graphical models: principles and techniques*. MIT press.

[BB87] https://vas3k.com/blog/machine_learning

[BB88] Hessel, Matteo, et al. "Rainbow: Combining improvements in deep reinforcement learning." *Thirty-second AAAI conference on artificial intelligence*. 2018.
[BB89] ASIMOV Architecture of optimized digital twins for AI-based training, 2022.
[BB90] Schulman, John, et al. "Proximal policy optimization algorithms." *arXiv preprint arXiv:1707.06347* (2017).
[BB91] https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/