Architecture and technical approach for DT-based AI-training: state of the art

[WP3; T3.2; Deliverable: D3.2 1.0]

public

AI training using Simulated Instruments for Machine Optimization and Verification

Document Information

Project	ASIMOV
Grant Agreement No.	20216 ASIMOV - ITEA
Deliverable No.	D3.2
Deliverable No. in WP	WP3; T3.2
Deliverable Title	Architecture and technical approach for DT-based AI-training: state of the art
Dissemination Level	public
Document Version	1.0
Date	2022.03.30
Contact	Richard Doornbos
Organization	TNO
E-Mail	richard.doornbos@tno.nl

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

Task Team (Contributors to this deliverable)

Name	Partner	E-Mail
Iftikhar Ahmad	TietoEVRY	iftikhar.ahmad@tietoevry.com
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
Ville Lämsä	VTT	ville.s.lamsa@vtt.fi
Lukas Schmidt	NorCom	lukas.schmidt@norcom.de

Formal Reviewers

Version	Date	Reviewer
0.4	2022.03.15	Tabea Henning (DLR)
0.4	2022.03.15	Remco Schoenmakers (TFS)

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	2022.03.30	Updated after formal review. Ready for publication.

Abstract

This report describes the state of the art in reinforcement learning and digital twin-based learning, and the first ideas on their application in the ASIMOV use cases. It will be the foundation for further work on researching these techniques in the ASIMOV use cases, which will enable expansion of the knowledge in these fields for general application in the high-tech industry.
A low-threshold introduction to reinforcement learning and Q-learning is followed by an extensive and well-structured literature overview. Finally, the initial ideas about which techniques and approaches to use and how to apply them in the use cases are described in the last part of this report.

Contents

Table of Figures

Table of Tables

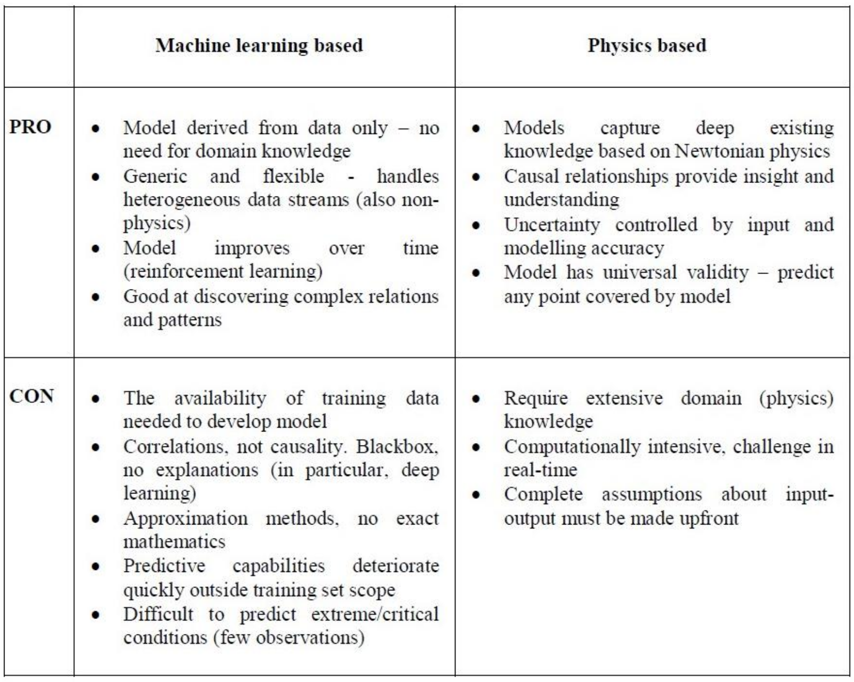
1 Introduction

Performing tests and optimizations on physical systems is often time consuming and can 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 on 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 ensure extensive system-level trials, model tunings, and adaptions. With a DT, a very large number of repetitions and scenarios can be gone through effectively in virtual environment. Next to that it is possible to exploit the higher speed, the repeatability, and furthermore there is 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 DTs and RL 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 focusses on the design and creation of a technical approach as well as a reference architecture for the training of 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. At 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 into 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 the 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 DT 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 versus driven by domain knowledge. This partially coincides with the columns of Table 1 but not exactly:

- Physics-based is an example of using domain knowledge; modelling a supply chain context is another example. Also, in some applications the physics knowledge may be as simple as a mathematical equation, or otherwise be computationally un-intensive.
- Data may be generated in different ways; an important distinction is *observational* and *experimental*. The latter allows for a combined approach of using domain knowledge and then switch 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 a 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.

As a general good practice, it can be beneficial to include domain knowledge in modelling / DT building whenever possible. There can be 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] a number of 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 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 solves a particular type of problem, where decision making is sequential and the final goal is typically long term. Examples of such problems include 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 explicit model at all, to learn from simple feedback alone.

The RL concepts and terms are formalized as follows:

- 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 game state.
- The principal goal of an agent is to 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 loss, 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)

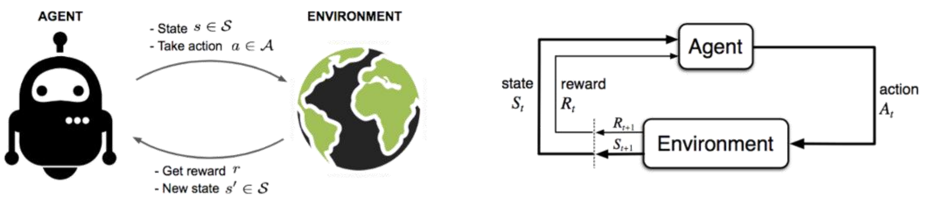


Figure 1 The basic elements of a reinforcement learning setup. Adapted from [99]

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

Version	Status	Date	Page
1.0	public	2022.03.30	11/40

Selecting the move, taking the current best is exploitation, whereas trying new things is exploration. This dilemma is captured by what is known as “Bayesian Bandits” [100].

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 larger 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”.

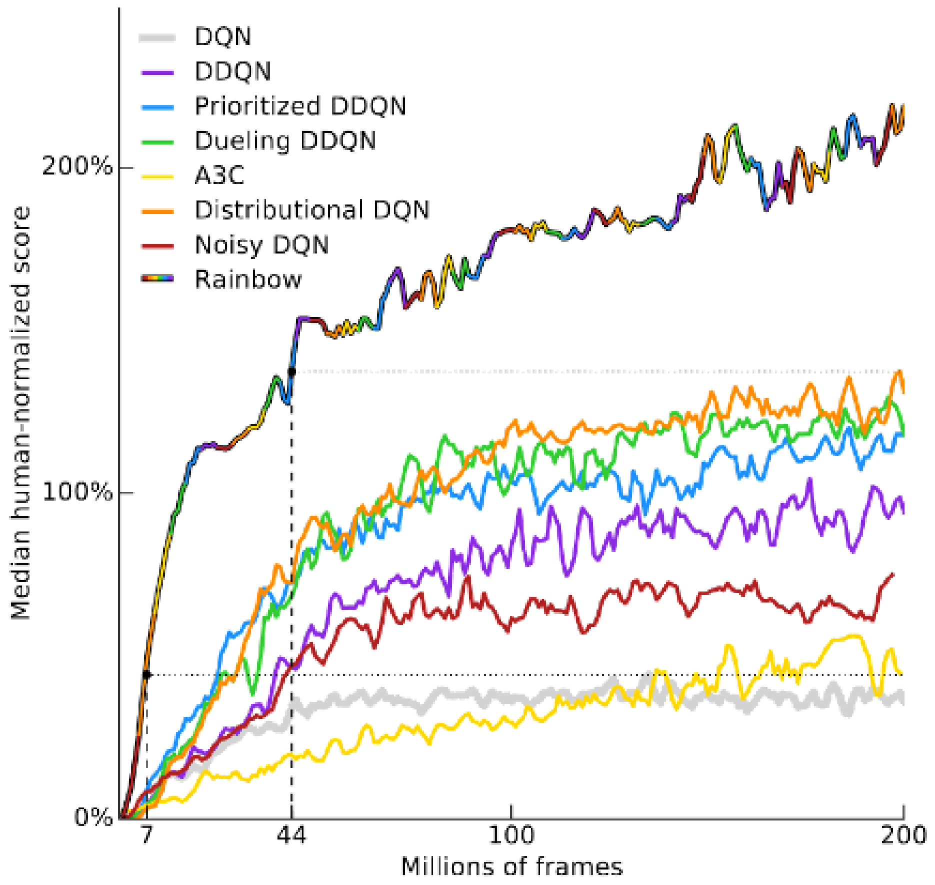


Figure 2 Data requirements for various RL algorithms [88]. The 100 percent mark on the y-axis denotes the human performance level. The x-axis shows the number of required training frames.

1.1.2 Data inefficiency

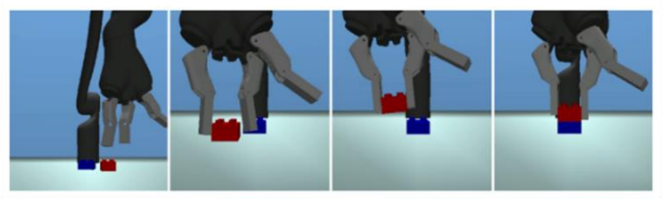
RL algorithms such as Q-learning, match and surpass human performance for Atari computer games like Pong. This is a landmark achievement for an algorithm that has no baked-in knowledge on what Pong is, or how to play it. From simple feedback it learns, and learns to play it well, discovering typical 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 game play, which 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. [101]

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



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

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 does not 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 supervised learning in those domains do not 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.

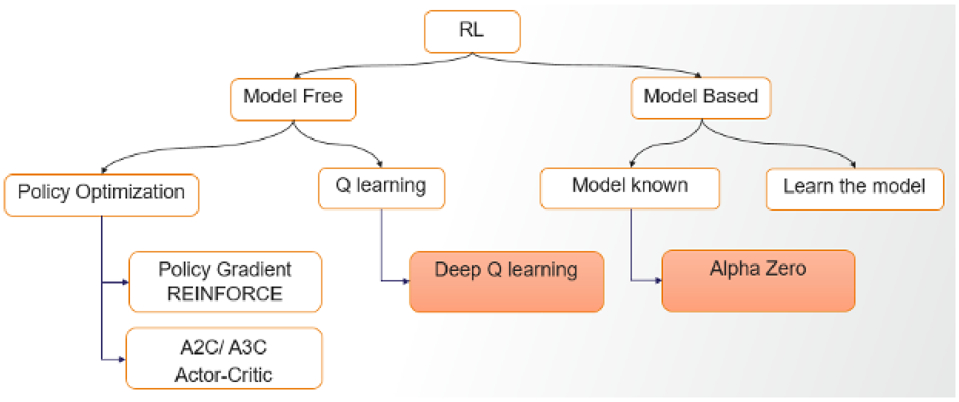


Figure 4 Taxonomy of reinforcement learning strategies.

1.2 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 slightly more formal.
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:

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

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 expected value given a certain policy. Again informally: the state-value is the expected final return given the fact that the agent is in state s at time t.

The Q-function Q(s, a):

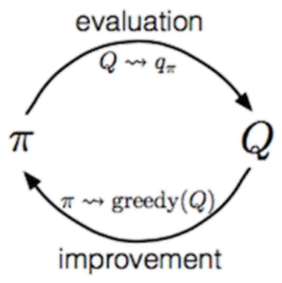
$$Q_\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a]$$

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:

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

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 it 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:

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


evaluation
$Q \rightsquigarrow q_\pi$
π
Q
$\pi \rightsquigarrow \text{greedy}(Q)$
improvement

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

Figure 5 Monte Carlo epsilon greedy Q-learning [99]

After an episode is complete, we 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 (epsilon) 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 highly inefficient. We will therefore introduce two more concepts that are central in RL: Temporal Difference Learning and the Replay-Buffer.

1.2.1 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 from 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

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

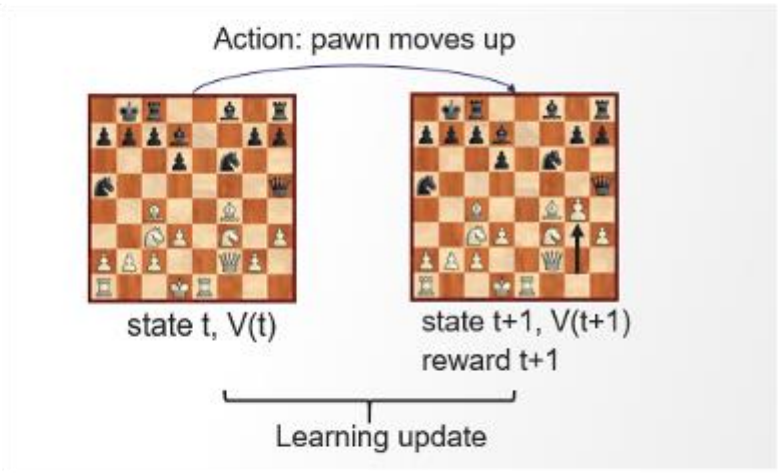


Figure 6 Learning strategy using only two states.

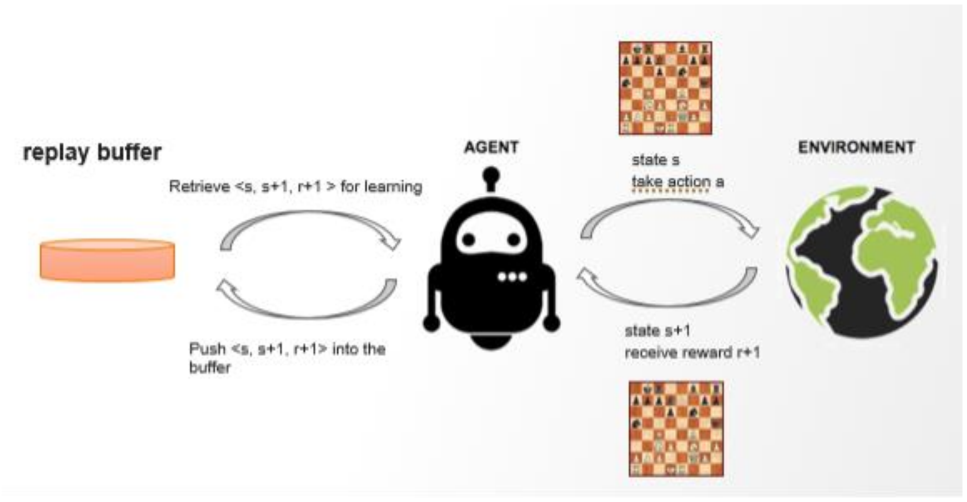


Figure 7 Setup using a replay buffer. Adapted from [99]

1.2.2 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, it does not only learn from complete games. It learns from collected game moves, that come from different games and different moments in the game all together.

This brings us the replay-buffer. The replay buffer is simply a place to store collected (state, action, reward, next-state) tuples 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 has 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, 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 vehicles it is very hard even for a human expert to assess 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. RL 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.

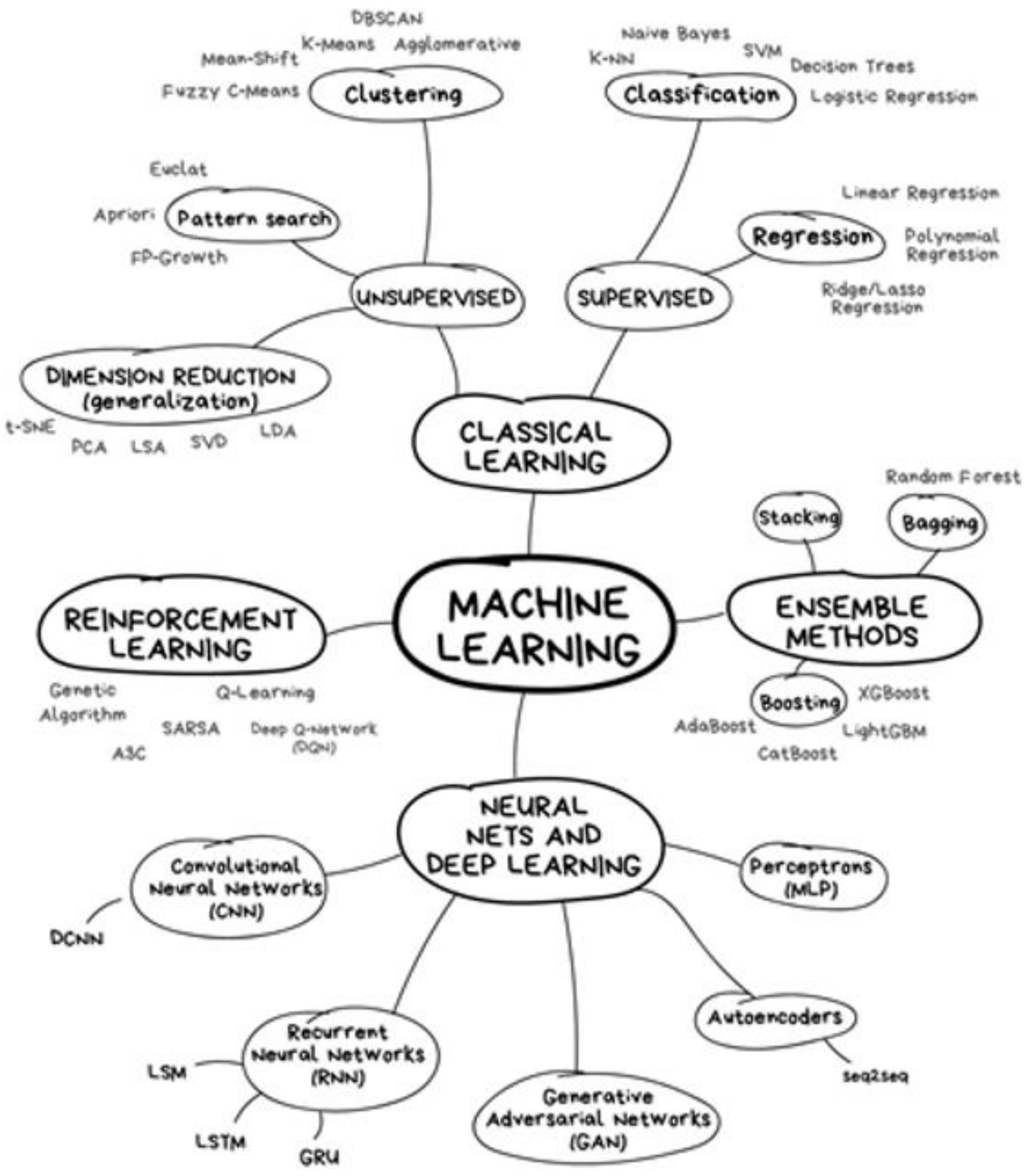


Figure 8 Machine learning taxonomy. Reproduced from [87].

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 known form of RL, where one can include algorithms such as Q-Learning, Temporal Difference, Actor Critic, Policy Gradients among many others. An introduction to RL where these concepts are detailed can be found in Section 1.1. 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 on data or just on models, which will be surveyed next. As a simple approach one can replace the real system by 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 [50]. In fact, a DT is a multi-physics and multi-scale 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.

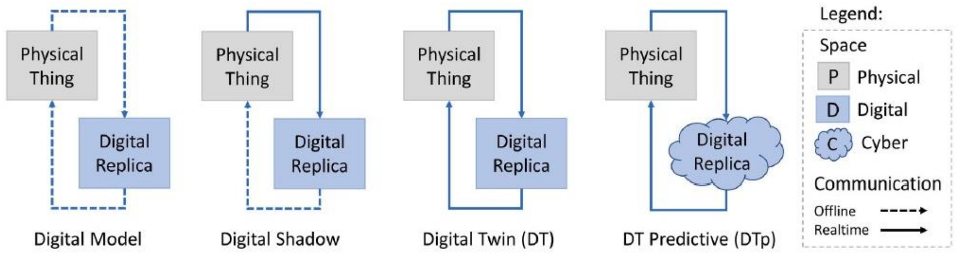


Figure 9 Illustration of the differences of a digital model, a digital shadow and a digital twin. Reproduced from [50].

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 [1] 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.

Here, mostly RL approaches that rely on (static) models will be surveyed, but some works relying on ('living' or online) digital twins will also be surveyed. The reasons for this are twofold. First, the limited number of RL 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, incorporating its extended features. For instance, if the process being replicated by the digital twin changes slowly (when compared to the time constants of the system dynamics) one can rely on RL techniques assuming static models/simulators, and, on top of this, create an adaptation loop to account for the slow process changes.

In the remainder of the document a state-of-the-art literature overview of RL approaches and methods is given. The literature about RL 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 be 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], RL 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. *Approximations 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 RL methods, rather than model-free methods.

While [2] sets the groundwork for methods used in RL 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.

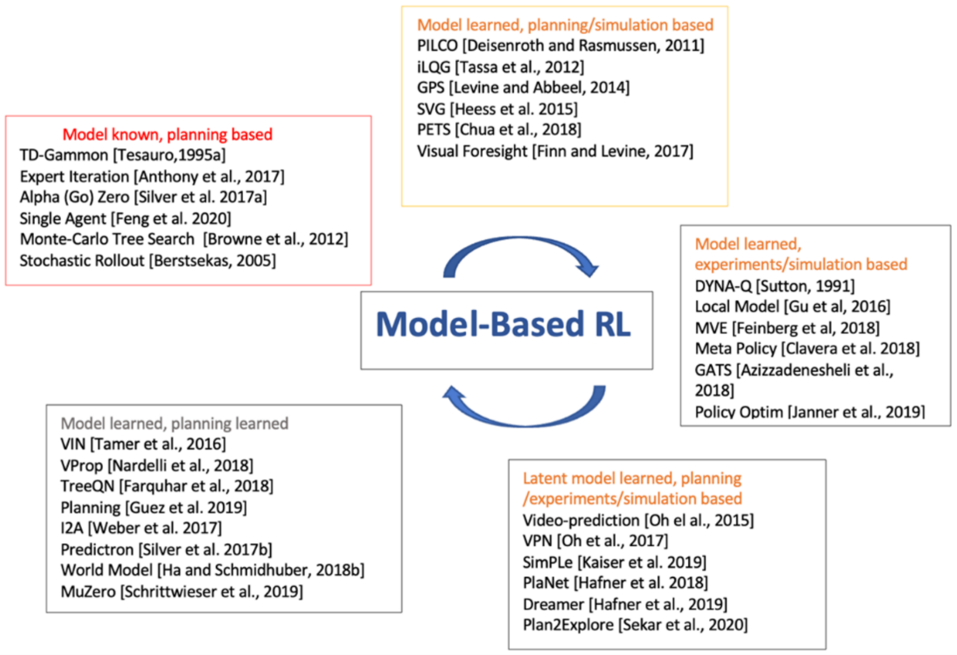

Model known, planning based
TD-Gammon [Tesauro,1995a]
Expert Iteration [Anthony et al., 2017]
Alpha (Go) Zero [Silver et al. 2017a]
Single Agent [Feng et al. 2020]
Monte-Carlo Tree Search [Browne et al., 2012]
Stochastic Rollout [Berstsekas, 2005]
Model learned, planning/simulation based
PILCO [Deisenroth and Rasmussen, 2011]
iLQG [Tassa et al., 2012]
GPS [Levine and Abbeel, 2014]
SVG [Heess et al. 2015]
PETS [Chua et al., 2018]
Visual Foresight [Finn and Levine, 2017]
Model-Based RL
Model learned, experiments/simulation based
DYNA-Q [Sutton, 1991]
Local Model [Gu et al, 2016]
MVE [Feinberg et al, 2018]
Meta Policy [Clavera et al. 2018]
GATS [Azizzadenesheli et al., 2018]
Policy Optim [Janner et al., 2019]
Model learned, planning learned
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]
Latent model learned, planning /experiments/simulation based
Video-prediction [Oh el al., 2015]
VPN [Oh et al., 2017]
SimPLe [Kaiser et al. 2019]
PlaNet [Hafner et al. 2018]
Dreamer [Hafner et al., 2019]
Plan2Explore [Sekar et al., 2020]

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

Model-based Reinforcement Learning methods are divided as follows:

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 [1, 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 spaces. 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-Q [12,13, Sutton, 1990, 1991], see Figure 11. Dyna-Q 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-Q. 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 RL 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].

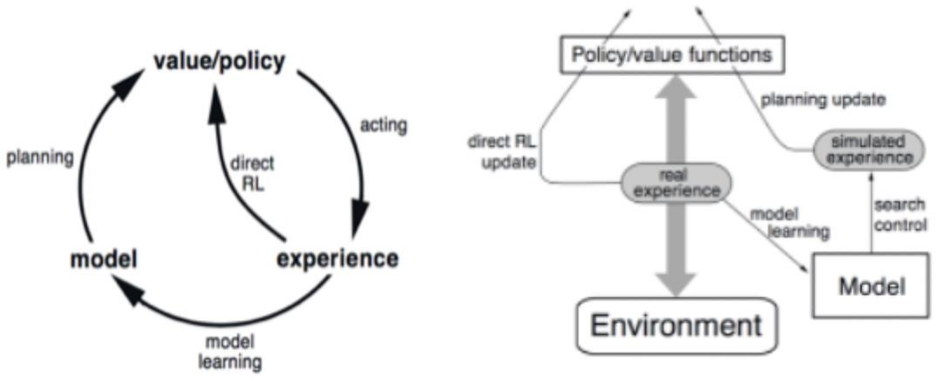


Figure 11 Dyna-Q, is a simple architecture that integrates models and experience [3].

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 RL 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 RL 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 aim 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 categories: 2) Model learned, planning/simulation based (referred to as Explicit Planning on Learned Transitions in [21]; and 3) 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 and an industrial use case is provided. 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 minimize 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.

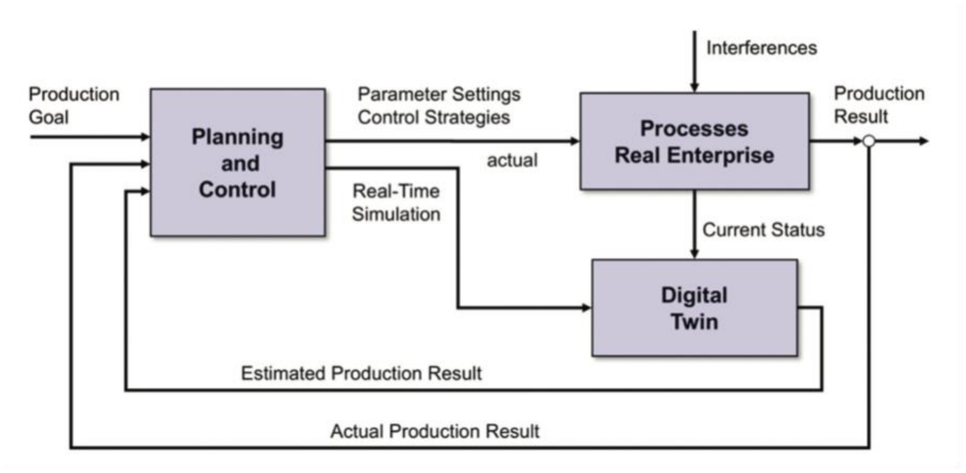


Figure 12 Reproduced from [3] in which this framework is proposed.

An established platform in the field of DTs 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.

Note: relevant new papers [102, 103] are being published at high pace in this rapidly developing field. We will cover the latest papers in the next version of this deliverable.

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 behavior 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.

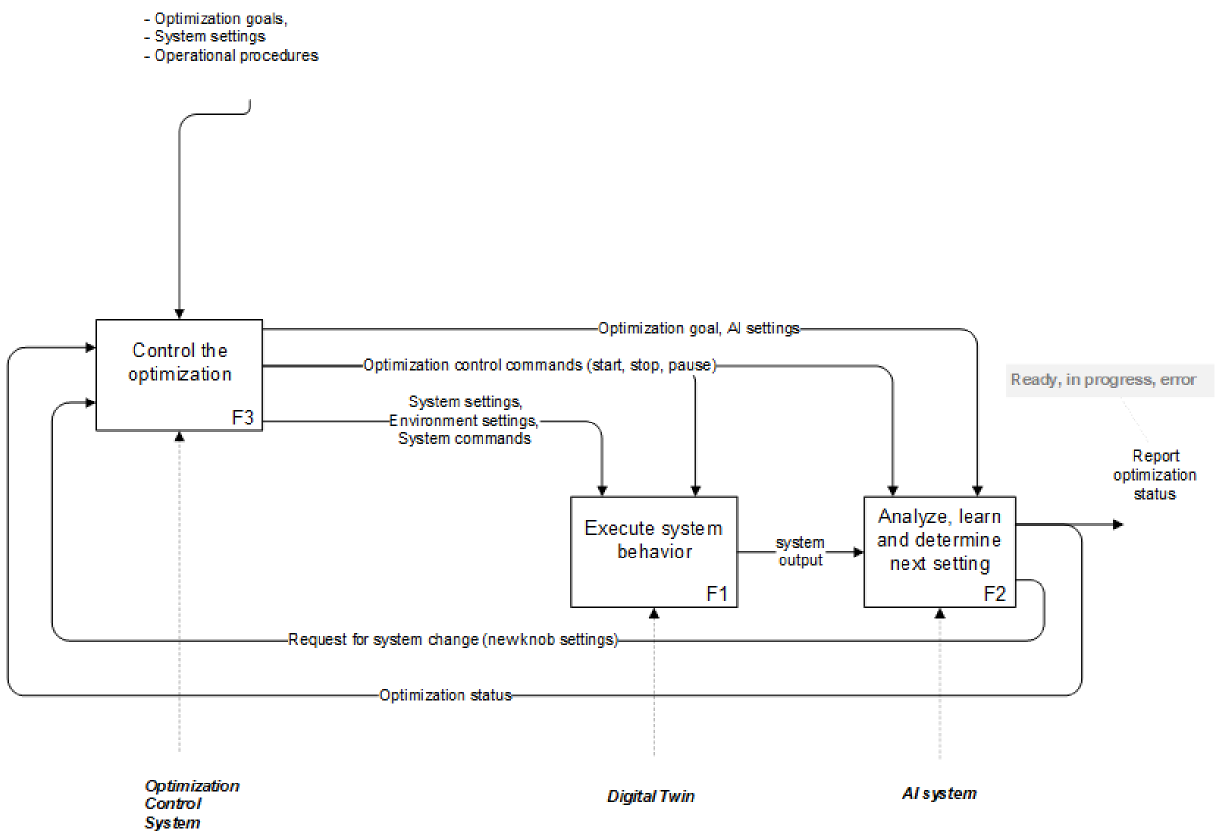

- Optimization goals,
- System settings
- Operational procedures
Control the optimization
F3
Optimization goal, AI settings
Optimization control commands (start, stop, pause)
System settings, Environment settings, System commands
Ready, in progress, error
Report optimization status
Execute system behavior
F1
system output
Analyze, learn and determine next setting
F2
Request for system change (newknob settings)
Optimization status
Optimization Control System
Digital Twin
AI system

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 RL. At high level, RL 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 interacts 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.
A **Policy** defines the learning behaviour of the agent at a given time. It is kind of stimulus and response rules. It can be stochastic i.e., specifying probabilities for each action.
A **Reward** signal defines the goals of the RL 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 the long run. Rewards are primary sense and value is the main goals in long term. Reward is given by the environment, but the value function must be estimated and re-estimated from the sequence of actions taken by agent.

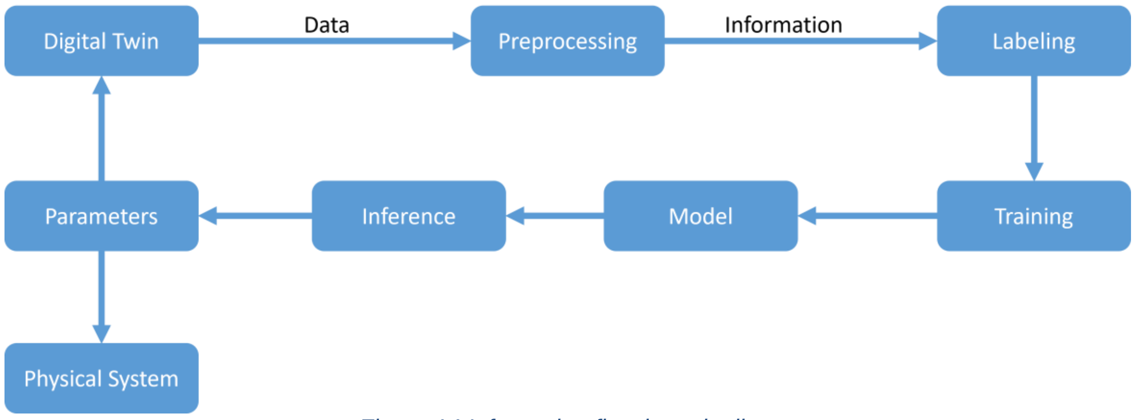


Figure 14 Information flow in a pipeline.

The 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 there are many strategies for it either online training or incremental training can be employed. In the second case, retraining is usually 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 RL 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 will be examined. 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.

3 The ASIMOV use cases

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 behavioral 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 analyzed. 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 realize such a system, an AI-learning technique is needed, which can optimize 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 RL 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.

3.2 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 RL 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.

3.3 Electron Microscope

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 RL 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.

3.3.1 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 RL 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 build, training RL algorithms on existing static datasets could be a good first step.

3.3.2 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.

3.3.3 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 STEM 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.

3.4 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 STEM 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 STEM 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 document. Also, Section 4.4.2 mentions for STEM modeling hysteresis, drift, pollution in the DT and dynamic environment based on a limited number of parameters 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; STEM 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 is quite possibly non-linear in the inputs
- Reward system for uses cases STEM 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 and merging

3.5 Suitability of different learning algorithms

3.5.1 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 at the introductory part of the chapter.

The goal of an MDP is to find the best decision, or action, in all states $s \in S$. The goal of RL is to find the optimal policy, which is the function that gives the best action a in all states $s \in S$. 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 RL) or they can be used to find the policy without first building the model (direct or model-free RL).

3.5.2 The learning and the planning problems

3.5.2.1 The learning problem

Capturing the transition function T from state S 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.

In the use case for ASIMOV, the learning part could comprise the T and R functions of the digital twin of the STEM or UUV, or the relevant processes within them. In the literature, a DT 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 DT 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).

3.5.2.2 The planning problem

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 as well.

3.5.3 Which approaches should we try and why?

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 STEM 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:

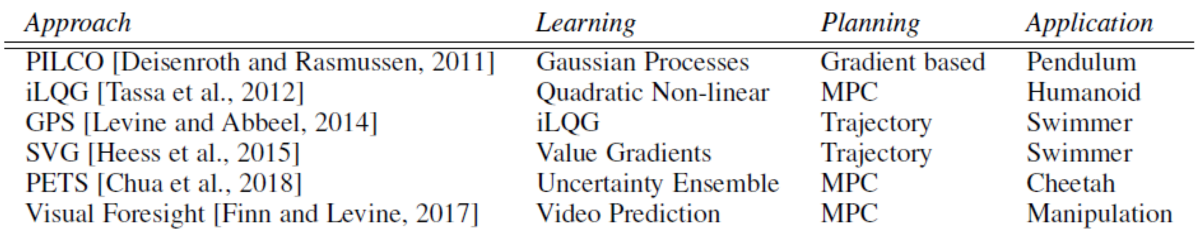
- For stochastic problems, often policy optimization algorithms work best; for example PPO belongs to this family of algorithms.
- For deterministic problems, often value optimization methods work best; for example DQN belongs to this family of algorithms.
- Combinations of both the categories above are also possible. In this category the best of policy optimization and value optimization approaches are combined. SAC would be an example of algorithms in this category.

Table 15 below can provide some visual clarity and examples.

From the model-based perspective, Tables 2 and 3 belonging to [21] below are provided with multiple approaches, from which the most promising should be selected depending on the use case of interest.

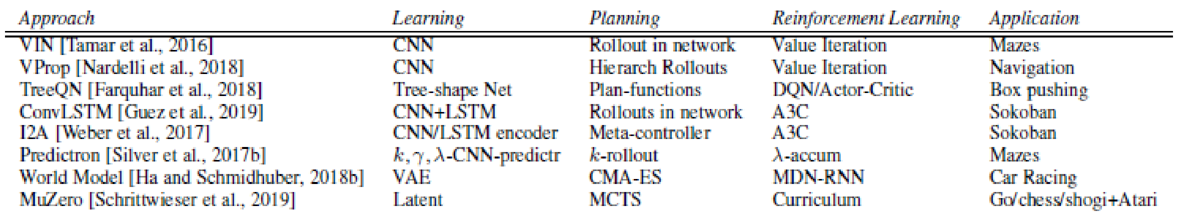
In the STEM use case, given transitions and rewards functions can be considered very unlikely, as far as the knowledge today dictates. So approaches based on learned transitions are considered more likely, and maybe in the future, end-to-end approaches deserve a try, but should surely be seen as less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

Table 2 Learned transition explicit planning approaches.



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 as well.

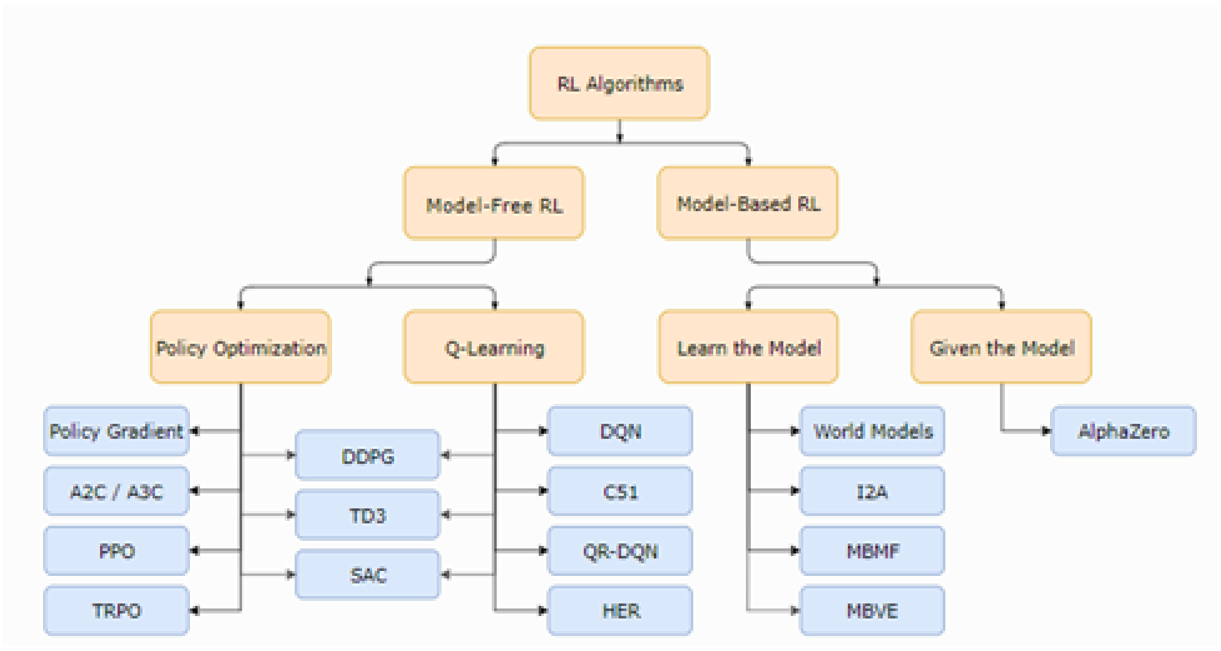


Figure 15 A taxonomy of reinforcement learning algorithms.

4 Conclusion

This report describes the state of the art in reinforcement learning and digital twin based learning, and the first ideas on their application in the ASIMOV use cases.
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. Currently reinforcement learning is considered the most promising approach for all use cases we address in the ASIMOV project. During the execution of this project the initial ideas about which techniques and approaches to use and how to apply them, will be validated and adjusted when needed. These results will be captured and consolidated in a next version of this report.

5 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.

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

[88] Hessel, Matteo, et al. "Rainbow: Combining improvements in deep reinforcement learning." *Thirty-second AAAI conference on artificial intelligence*. 2018..

[99] lilianweng.github.io

[100] Katehakis, Michael N., and Arthur F. Veinott Jr. "The multi-armed bandit problem: decomposition and computation." *Mathematics of Operations Research* 12.2 (1987): 262-268.

[101] Schrittwieser, Julian, et al. "Mastering atari, go, chess and shogi by planning with a learned model." *Nature* 588.7839 (2020): 604-609.

[102] Degrave, Jonas, et.al.;"Magnetic control of tokamak plasmas through deep reinforcement learning", Nature volume 602, pages 414–419, 2022.

[103] Wurman, Peter R. et.al., “Outracing champion Gran Turismo drivers with deep reinforcement learning”, Nature volume 602, pages 223–228, 2022.