Koutas *et al. J Infrastruct Preserv Resil* (2024) 5:6
https://doi.org/10.1186/s43065-024-00098-9

Journal of Infrastructure Preservation and Resilience

RESEARCH Open Access

An investigation of belief-free DRL and MCTS for inspection and maintenance planning

Daniel Koutas[1*], Elizabeth Bismut[1] and Daniel Straub[1]

Abstract

We propose a novel Deep Reinforcement Learning (DRL) architecture for sequential decision processes under uncertainty, as encountered in inspection and maintenance (I &M) planning. Unlike other DRL algorithms for (I &M) planning, the proposed +RQN architecture dispenses with computing the belief state and directly handles erroneous observations instead. We apply the algorithm to a basic I &M planning problem for a one-component system subject to deterioration. In addition, we investigate the performance of Monte Carlo tree search for the I &M problem and compare it to the +RQN. The comparison includes a statistical analysis of the two methods' resulting policies, as well as their visualization in the belief space.

Keywords One-component deteriorating system, Maintenance planning, Partially observable MDP, Deep reinforcement learning, Neural networks, Monte Carlo tree search

Introduction

Reliable civil infrastructure, such as power, water and gas distribution systems or transportation networks, is essential for society. Large efforts are therefore spent on properly maintaining these systems. However, at present such maintenance is based mainly on simple legacy rules, such as fixed inspection intervals, combined with expert judgement. There is a significant potential for optimal inspection and maintenance (I &M) planning that makes best use of the information at hand to ensure safe and reliable infrastructure while being sustainable and cost-efficient [1–3].

I &M planning is a sequential decision making problem under uncertainty. One challenge in deriving optimal I &M decisions is the presence of large epistemic and aleatoric uncertainties associated with the system properties, load, representation model, and measurements [4–7]. Another major challenge is the exponential increase in possible I &M strategies with the number of components and the considered time horizon [4, 8]. Standard practice for dealing with these challenges is the use of established decision heuristics, e.g., safety factors during design, predetermined scheduled inspections, and threshold- or failure-based replacement of components [9–11]. The parameters of these heuristics can then be optimized to find good I &M strategies [4, 8]. However, heuristics can be suboptimal and finding good heuristics is challenging.

Another approach to embed uncertainty into the inherently sequential nature of inspection and maintenance problems, is to integrate probabilistic models into decision process models [12–14]. Under certain conditions, these sequential decision problems under uncertainty can be modeled as Partially Observable Markov Decision Processes (POMDPs), which provide an efficient framework for optimal decision making, and can additionally account for measurement errors [15–17]. The POMDP is in general intractable [18]. Many approaches for solving the POMDP use the belief state representation, which incorporates the entire information, i.e., actions and observations up to the current point [15, 19–22]. However, these methods require an explicit probabilistic model of the environment to calculate the transition

*Correspondence:
Daniel Koutas
daniel.koutas@tum.de
[1] Engineering Risk Analysis Group, Technische Universität München, München, Germany

probabilities between states as well as the belief states, which is not always available. In addition, they typically are not computationally efficient beyond small state and action spaces [19]. This hinders their application to I &M planning of infrastructure systems, where the investigated systems are usually consisting of a larger number of components.

Reinforcement learning approaches to solve POMDPs have gained in popularity, including Deep Reinforcement Learning (DRL) with neural networks (NNs), and Monte Carlo Tree Search (MCTS). There exist numerous variants of NNs for discrete [23–25] and continuous [26–28] action space control, employing for example Deep Q-networks (DQNs) [29, 30], Double DQNs (DDQNs) [31, 32] or actor-critic architectures [33]. Although MCTS was originally formulated for fully observable domains with great success [34, 35], it has also been applied to POMDPs [36, 37].

Both NNs and MCTS have been heavily researched in the field of computer games, which provide a safe (i.e., no real-life consequences) and controllable environment with a variety of complex problems to solve (2D, 3D, single-agent, multi-agent, etc.) with an infinite supply of useful data that is much faster than real-time [38]. The success of these methods in this application has motivated researchers to apply them to I &M planning (e.g., [16, 20, 39, 40]). However, this problem's specific characteristics e.g., sparse rewards due to low probability of failure, can pose a challenge to DRL methods, the efficiency of which remains to be systematically assessed.

The literature on solving POMDPs with DRL in the context of I &M is fairly limited. Most studies have focused on fully observable MDPs, for instance coupling Bayesian particle filters and a DQN for real-time maintenance policies [41], employing a DDQN for preventive maintenance of a serial production line [42], coupling a pre-trained NN for reward estimation with a DDQN for maintenance of multi-component systems [43], and adopting a DDQN for rail renewal and maintenance planning [44]. Concerning POMDPs, Andriotis and Papakonstantinou [20] developed the Deep Centralized Multi-agent Actor Critic (DCMAC) architecture for multi-component systems operating in high-dimensional spaces, with extended applications for roadway network maintenance [39]. The corresponding decentralized version (DDMAC), where each agent has a separate policy network [16], has been applied to life cycle bridge assessment [40] and 9-out-of-10 systems [45]. However, both DCMAC and DDMAC take the belief state of the system as an input, which is in general computationally expensive to obtain for a system with many components and arbitrary state evolution processes. Thus, newer studies (e.g., [46, 47]) have shifted the focus to observation-based DRL. However, a problem setting concerning continuous state and continuous erroneous observations has not been considered, yet.

In a similarly limited manner, MCTS has been applied to maintenance planning problems modeled as MDPs. Examples with MCTS include, for instance, finding stochastic schedules in active distribution networks [48], in combination with genetic algorithms for condition-based maintenance [49], or combined with NNs for wind turbine maintenance [50]. To the best of our knowledge, MCTS has not been applied to POMDPs in the context of I &M.

The purpose of this paper is twofold. Firstly, we propose a DRL architecture for POMDP and I &M planning, which does not require the computation of the belief state. The proposed NN combines the features of the Action-specific Deep Recurrent Q-Network [25] and the dueling architecture [51]. The resulting +RQN architecture is able to deal directly with erroneous observations over the whole life cycle of the system.

Secondly, we investigate the performance of MCTS when applied to I &M planning. In this context, we perform a systematic comparison of the proposed +RQN and MCTS. The investigated problem is a one-component system subject to deterioration and is formulated as a POMDP, for which an exact solution is available, because of linear Gaussian assumptions for the model dynamics. Component deterioration models are often used for investigations in infrastructure I &M planning (e.g., [21, 52, 53]) and are applied for I &M planning in practice (e.g., [54, 55]). The analysis includes a comparison of performance, i.e., the achieved optimized expected life cycle costs (LCC) and the computation time. It is carried out for different measurement errors. We also review the information carried by two metrics to compare the resulting policies of the two methods, namely via a statistical analysis and a visualization in the belief space. The solutions from both methods are compared to the exact POMDP solution.

The structure of the paper is as follows. Basic maintenance problem section introduces the investigated problem as well as sequential decision making along with the key definitions and metrics needed for the employed RL methods. Neural networks section explains the workings of the NN architecture used herein, and MCTS section illustrates how the MCTS method has been adapted for solving the proposed problem. Metrics for comparison section is dedicated to the metrics we employ to compare the NN and MCTS solutions, and Computation time, Performance, and Policy comparison sections contain the respective results. Discussion section discusses the obtained solutions and policies, and gives insight into the advantages and disadvantages of the two approaches.

Basic maintenance problem

Investigated system

For the numerical investigations in this paper, we study a one-component system subject to deterioration, taken from [56]. It is modeled with two random variables (RVs): D representing the *deterioration state* and K representing the *deterioration rate*. The subscript t indicates timesteps, where $t = 0,\ 1,\ 2,\ ...,\ T_{\text{end}}$, with finite time horizon T_{end}. The generic deterioration model is given as

$$D_t = D_0 + t \cdot K_0 \quad \Longleftrightarrow \quad \begin{cases} D_t = D_{t-1} + K_{t-1} \\ K_t = K_{t-1} \end{cases}, \tag{1}$$

where D_0 and K_0 are normally distributed and independent. Equation (1) shows that the deterioration process is modeled as a Markov process through state space augmentation. The deterioration D_t is observable with a Gaussian measurement noise, through the measurement random variable O_t, i.e., $O_t \sim \mathcal{N}(D_t, \sigma_E)$.

Four actions $a_0 - a_3$ are available for counteracting the deterioration and ultimately the failure of the structure. The action A_t is taken after observation O_t and affects D_{t+1} and/or K_{t+1} (see Appendix 1). The effects of the actions on the system are detailed in Appendix 1: Table 2. The structure fails when the deterioration exceeds the critical deterioration d_{cr}. In the failed state, an annual failure cost is incurred until the system is either repaired or replaced (no automatic setback of the system to the initial state). In addition, each action a_i has a specific cost c_{a_i} incurred at time t.

Figure 1 depicts the generic influence diagram of the corresponding POMDP.

This case study is set up such that linearity, and hence also the normality of any set of RVs, is conserved (see Appendix 1: Table 2). As a result, the belief state and all transitions of the belief-MDP can be computed analytically.

Moreover, in our case, the covariance matrix does not depend on the observations and the actions taken, and can hence be pre-computed for all timesteps. Thus, the actions and observations only influence the prior and posterior means of D_t and K_t, respectively (see Appendix 1).

The model assumption allows for the system to regenerate if K_t is negative. However, 1) we set up the numerical values so that we limit this effect, 2) it is a useful assumption for obtaining a reference solution and 3) the solution methods introduced hereafter do not require it.

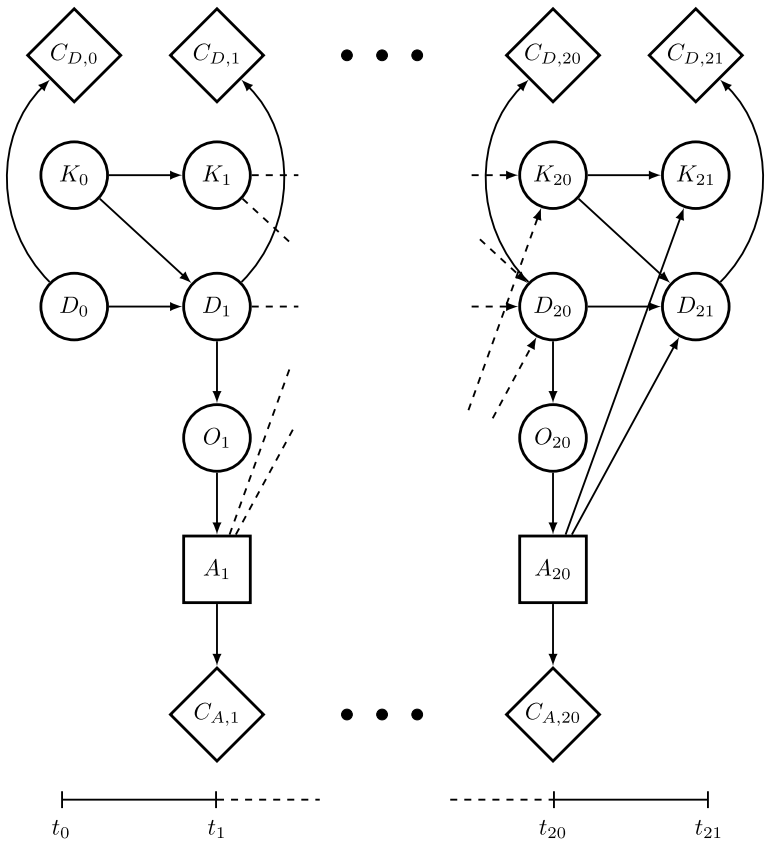

Fig. 1 Complete influence diagram of the model, especially depicting the starting and end operations, where $T_{end} = 21$. The first action is taken at $t = t_1$

Sequential decision making

At every timestep, the operator has to decide which action to choose based on the history of observations and actions; hence they try to solve a sequential decision making problem. Specifically, as the deterioration state D_t is only observable through erroneous measurements O_t, and the deterioration rate K_t is not observable at all, the investigated setup falls under the category of a Partially Observable Markov Decision Processes (POMDP) [15]. One can transform a POMDP into a belief MDP by replacing the states with the belief (vector) as the variable of interest, and then employ conventional methods for solving MDPs, such as value iteration (VI) or policy iteration [57]. We utilize this belief state representation to obtain a reference solution for the numerical investigations (see POMDP reference solution and Results sections). However, the focus of this paper is specifically on reinforcement learning (RL) techniques that can directly deal with observation-action sequences and hence do not need the belief state representation.

The goal is to find a sequence of actions that minimizes the expected life cycle cost (LCC), which is defined as the sum of discounted expected action and failure costs:

$$\overline{\mathrm{LCC}} = \mathbb{E}[\mathrm{LCC}] = \sum_{t=0}^{T_{end}} \gamma^t \cdot \mathbb{E}[C(A_t) + C(F_t)]. \tag{2}$$

In standard literature, the two costs associated with action and failure are summarized in a single cost $C(s, a)$, which is the immediate cost resulting from executing action a in state s of the system. Hence, will adopt this notation in the following.

The decision-making rule, which determines the action to take in function of the available information, is called the *policy* π. In general, the policy is time- and history-dependent [15, 58]. There exists a mapping from the current observation-action history $h_t = (o_{1:t}, a_{1:t-1})$ to the time-agnostic belief over the set of system states $b(s_t) = p(s_t|o_{1:t}, a_{1:t-1})$, where $b(s)$ represents the probability of the system being in state s, when the agent's belief state is b [59]. Hence, the policy as well as other functions can be expressed in terms of both:

$$\pi = \pi_t(h_t) = \pi_t(b). \tag{3}$$

Accordingly, the ideal policy π^* determines the ideal action to take to reach the set goal. For finite-horizon problems (as for our case study), π^* is generally time-dependent. In our case, the set of ideal policies $\{\pi_t^*,\ t = 1,\ 2,\ ...,\ T_{\mathrm{end}} - 1\}$ is the one that minimizes $\overline{\mathrm{LCC}}$. To find an expression for π_t^*, we substitute the global LCC measure (Eq. 2) with recursively defined value functions.

A *state value function* assigns a value to a particular (belief) state at a specific point in time. We denote with $V_t^\pi(b)$ the sum of expected discounted costs when following policy π starting from belief b at time t [60]. The optimal value function is then defined as [59]:

$$\begin{aligned} V_t^*(b) &= \min_\pi V_t^\pi(b) \\ &= \min_{a\in\mathcal{A}} \left[\sum_{s\in\mathcal{S}} C(s,a)\cdot b(s) + \gamma \cdot \sum_{o\in\mathcal{O}} \mathrm{P(o|b,a)} \cdot \mathrm{V_{t+1}^*(b_o^a)} \right], \end{aligned} \tag{4}$$

where b_o^a is the belief that results from b after executing action a and observing o, and can be obtained from the POMDP model and Bayesian updating (e.g., demonstrated in [57]). Note that $\mathrm{P(o|b,a)}$ can be expressed as a function of the belief transition probability $\mathrm{P(b_o^a|b,a)}$, and the sum over o can be transformed into a sum over b (see POMDP reference solution section).

One can also define an *action-value function* $Q_t^\pi(b_t, a)$, which denotes the value of action a at belief state b under policy π at time t and continuing optimally for the remaining timesteps until the end of the system lifetime [57]. The optimal value function V^* can be expressed as a minimization over the action-value function Q, and the optimal action-value function Q^* satisfies the *Bellman equation* [15, 57, 61]:

$$Q_t^*(b,a) = \sum_{s\in\mathcal{S}} C(s,a)\cdot b(s) + \gamma \cdot \sum_{o\in\mathcal{O}} \mathrm{P(o|b,a)} \cdot \min_{\mathrm{a'}\in\mathcal{A}} \mathrm{Q_{t+1}^*(b_o^{a'}, a')}, \tag{5}$$

Lastly, the *advantage function* $A_t^\pi(b, a)$ is a measure of the relative importance of each action [51]:

$$A_t^\pi(b,a) = Q_t^\pi(b,a) - V_t^\pi(b), \tag{6}$$

where the advantage of the optimal action a^* is 0 [51]:

$$Q_t^\pi(b,a^*) = V_t^\pi(b) \quad \Longrightarrow \quad A_t^\pi(b,a^*) = 0. \tag{7}$$

The optimal policy at every timestep can be easily extracted by performing a greedy selection over the optimal Q-value [57]:

$$\pi_t^*(b) = \arg\min_{a\in\mathcal{A}} Q_t^*(b,a), \tag{8}$$

which is also the value- and advantage-minimizing action from Eqs. (4) and (6), respectively.

The solution methods presented in Neural networks and MCTS sections have the goal of approximating V, Q, or A, from which the optimal policy can be extracted.

POMDP reference solution

To evaluate the performance of approximate solutions, we also provide a reference solution for the POMDP model of Investigated system section. It is computed with standard value iteration applied to a discretized belief MDP. The belief, in our case, is a vector comprising the posterior mean values of D and K from Eqs. (27) and (28):

$$\boldsymbol{b} = \begin{bmatrix} \mu''_{D,t} \\ \mu''_{K,t} \end{bmatrix} \tag{9}$$

The adapted version of Eq. (4) for discretized beliefs is then [56, 59]:

$$V_t^*(\boldsymbol{b}) = \min_{a \in \mathcal{A}} \left[C(\boldsymbol{b}, a) + \gamma \cdot \sum_{\boldsymbol{b}' \in \mathcal{B}} \mathrm{P}\left(\boldsymbol{b}' | \boldsymbol{b}, \mathrm{a}\right) \cdot \mathrm{V}_{\mathrm{t+1}}^*(\boldsymbol{b}'), \right] \tag{10}$$

where $C(\boldsymbol{b}, a) = \sum_{s \in \mathcal{S}} C(\boldsymbol{s}, a) \cdot \boldsymbol{b}(\boldsymbol{s})$ (see Eqs. (4) and (5).

Due to the linear Gaussian transition dynamics of this case study, the transition probabilities $\mathrm{P}\left(\boldsymbol{b}' | \boldsymbol{b}, \mathrm{a}\right)$ can be calculated analytically. The discretization of the belief and the computation of the probability tables is done according to [62]. Equation (10) is solved by backward induction for each discrete belief state. The resulting $\overline{\mathrm{LCC}}$ is verified by MCS. The discretization scheme is chosen such that 1) the value function of Eq. (4) is estimated with a small error (compared to MCS on continuous belief space) and 2) such that the resulting policy is quasi-optimal (it performs better than every other solution found).

Note that in the general case, obtaining a reference solution with dynamic programming, e.g., via value iteration, is not feasible due to the super-exponential growth in the value function complexity [63]. Hence, the problem investigated in this work represents a special case.

Neural networks

Architecture

Our aim is an NN approach that is able to handle imperfect observations without the need for computing the belief. The NN needs to account for the time dependence of the value function for the finite horizon problem. This can be achieved by a network architecture that is able to handle sequential data, i.e., the observation-action history. For that, we adopt the basic structure of the action-specific deep recurrent Q-network [23].

The final NN architecture proposed in this work is depicted in Fig. 2, which we name Action-specific Deep Dueling Recurrent Q-network (+RQN). At each timestep t, the two inputs of the network are the one-hot encoded action [64] taken at $t-1$ and the scalar observation obtained at t. The outputs of the network are the estimated Q-values for each action at t. The inputs are fed through two fully connected (FC) layers for feature extraction. The core of the network is formed by the Long Short-term Memory (LSTM) layer, which can resolve short as well as long-term dependencies through the hidden and cell states, respectively [65]. Depending on the observation-action history, these states take different values. Hence, the LSTM layer can be interpreted as a high-dimensional embedding of the history or a high-dimensional approximator of the belief state. The LSTM output is then fed through another FC layer for further feature extraction. To estimate the Q-values, the value and the advantage functions are first estimated separately and combined using a modified version of Eq. (6), which is discussed in the section below. Wang et al. [51] report that this configuration has superior performance compared to standard DQNs.

Q-values, loss, cost and weight updates

Instead of directly using Eq. (6), Wang et al. [51] propose to introduce the mean over the advantages as a correction term, which improves the stability of the optimization of the network parameters. Let $\boldsymbol{\theta}_t^j$ denote the parameters of all layers prior to the value-advantage split, $\boldsymbol{\upsilon}^j$ denote the parameters of the value stream, and $\boldsymbol{\alpha}^j$ the parameters of the advantage stream. The superscript $j = 1, 2, ..., N_e$ refers to the weights at a certain iteration/epoch and hence highlights iterative convergence towards a set of weights that best approximate the true Q-value. Herein, an epoch consists of passing through the whole life cycle of a batch of sample trajectories, after which the weights get updated, and the next epoch starts. Since $\boldsymbol{\theta}$ does also include the hidden and cell states of the LSTM layer, it is dependent on the observation-action history, which is denoted with the subscript t. By contrast, $\boldsymbol{\alpha}$ and $\boldsymbol{\upsilon}$ stay constant for the whole life cycle (epoch). The modified approximation for the Q-values is then [51]:

$$\begin{aligned} \mathbb{Q}_t^j(o_t, a \mid a_{t-1}, \boldsymbol{\theta}_{t-1}^j, \boldsymbol{\alpha}^j, \boldsymbol{\upsilon}^j) = \mathbb{V}_t^j(o_t \mid a_{t-1}, \boldsymbol{\theta}_{t-1}^j, \boldsymbol{\upsilon}^j) + \\ \left(\mathbb{A}_t^j(o_t, a \mid a_{t-1}, \boldsymbol{\theta}_{t-1}^j, \boldsymbol{\alpha}^j) - \frac{1}{|\mathcal{A}|} \sum_{a' \in \mathcal{A}} \mathbb{A}_t^j(o_t, a' \mid a_{t-1}, \boldsymbol{\theta}_{t-1}^j, \boldsymbol{\alpha}^j) \right), \end{aligned} \tag{11}$$

where $\mathbb{Q}_t^j(o_t, a \mid \boldsymbol{\theta}_{t-1}^j, \boldsymbol{\alpha}^j, \boldsymbol{\upsilon}^j)$ is the Q-value estimate for the action a at time t and epoch j after observing o_t and given the previous action a_{t-1}, the weights $\boldsymbol{\theta}_{t-1}^j$ (which embedded $o_{1:t-1}$ and $a_{1:t-2}$ through the hidden and cell states), $\boldsymbol{\alpha}^j$ and $\boldsymbol{\upsilon}^j$. Accordingly, $\mathbb{V}_t^j(o_t \mid a_{t-1}, \boldsymbol{\theta}_{t-1}^j, \boldsymbol{\upsilon}^j)$ does not use the weights of the separate advantage stream $\boldsymbol{\alpha}^j$ and vice versa.

To evaluate the performance of the network, i.e., the accuracy of the predicted Q-values, we need a target value for each pair of sample observation and action $o_t^{(i)}$, $a_{t-1}^{(i)}$, which are passed as inputs to the network. To obtain a

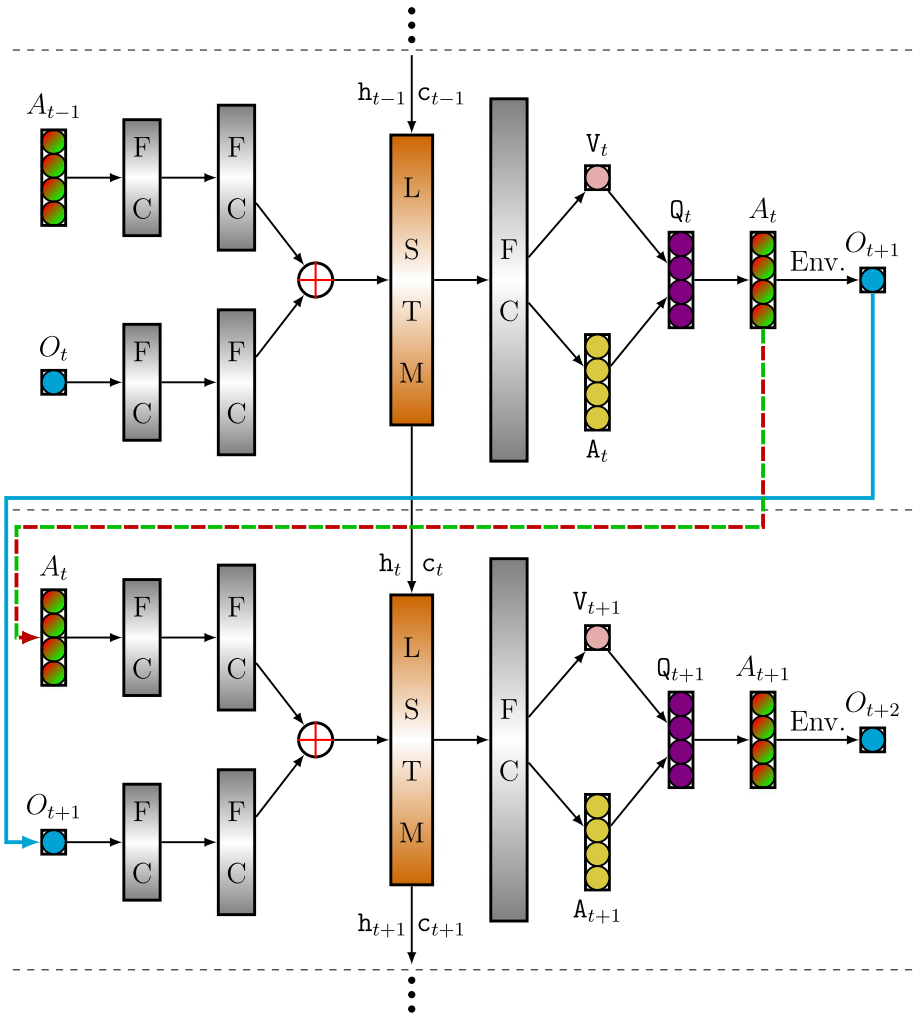

Fig. 2 Snapshot of our NN architecture depicting the information flow through the network from $t \rightarrow t+1$. A is the action taken at the previous timestep in one-hot encoded form, e.g., $a_0 = [1\ 0\ 0\ 0]^T$, O is the current observation, V, A & Q are the Value, Advantages, and action-values, respectively. The "Env." above the arrow denotes an interaction with the environment. The gray layers represent FC layers and the orange layer represents the LSTM layer with its hidden state and cell states h, c. The + represents the concatenation operation. The number of circles inside some layers depicts the fixed number of nodes, and the relative sizes of the layers qualitatively show the number of nodes; adapted and merged from [25, 51]

target value, we use the fact that the optimal Q-values follow the Bellman equation (Eq. 5). Therefore, we define the NN output $y^{(i),j}_{\text{NN,t}}$ and the target value $y^{(i),j}_{\text{Tar,t}}$ for a sample (i) at a specific point in time t and epoch j as [25]:

$$y^{(i),j}_{\text{NN,t}} = \mathcal{Q}^j_t(o^{(i)}_t, a_{sel.} \mid a_{t-1}, \boldsymbol{\theta}^j_{t-1}, \boldsymbol{\alpha}^j, \boldsymbol{\upsilon}^j) \tag{12}$$

$$y^{(i),j}_{\text{Tar,t}} = c^{(i)}_t + \gamma \cdot \min_{a' \in \mathcal{A}} \mathcal{Q}^j_{t+1}(o^{(i)}_{t+1}, a' \mid a_t, \boldsymbol{\theta}^{j,-}_t, \boldsymbol{\alpha}^{j,-}, \boldsymbol{\upsilon}^{j,-}) \tag{13}$$

where $a_{sel.}$ denotes the selected action under the behaviour policy (ϵ-greedy, see Appendix 2: Optimized NN parameters section) at j; $c^{(i)}_t$ is the total cost sample at t which includes the cost of a potential failure at t and the cost of the latest selected action at $t-1$ under the behaviour policy at j. Moreover, $\mathcal{Q}^j_{t+1}(o^{(i)}_{t+1}, a' \mid \boldsymbol{\theta}^{j,-}_t, \boldsymbol{\alpha}^{j,-}, \boldsymbol{\upsilon}^{j,-})$ denotes the Q-value estimate at $t+1$ and epoch j, after an action has been taken under the behaviour policy at t and j which, upon interaction with the environment, resulted in observation $o^{(i)}_{t+1}$. The "$-$" indicates that the parameters $\boldsymbol{\theta}^{j,-}_t, \boldsymbol{\alpha}^{j,-}, \boldsymbol{\upsilon}^{j,-}$ belong to a separate *target network* [25]. More details on the sampling procedure and the target network are provided in Training procedure section.

For training, we use the mean-squared error (MSE) *loss function* [66]:

$$\mathcal{L}_{\mathrm{MSE}}\left(y_{\mathrm{NN}},\, y_{\mathrm{Tar}}\right) := \left(y_{\mathrm{NN}} - y_{\mathrm{Tar}}\right)^2. \tag{14}$$

For *stochastic gradient descent*, a batch of N_b samples is passed through the network to speed up training [67]. On this basis, the MSE cost function accumulated over the whole life cycle is evaluated as

$$\mathcal{C}_{\mathrm{MSE}}^{\mathrm{LCC}}\left(\boldsymbol{y}_{\mathrm{NN}}^{(1:N_b)},\, \boldsymbol{y}_{\mathrm{Tar}}^{(1:N_b)}\right) := \frac{1}{N_b} \sum_{t=1}^{T_{\mathrm{end}}-1} \sum_{i=1}^{N_b} \mathcal{L}_{\mathrm{MSE}}\left(y_{\mathrm{NN,t}}^{(i)},\, y_{\mathrm{Tar,t}}^{(i)}\right). \tag{15}$$

The NN weights are updated based on this cost function. The simplest gradient-based update scheme is [66, 68]:

$$\boldsymbol{\nu}^{j+1} = \boldsymbol{\nu}^{j} - \eta \nabla_{\boldsymbol{\nu}} \mathcal{C}_{\mathrm{MSE}}^{\mathrm{LCC}}, \tag{16}$$

where η is the learning rate. The weights $\boldsymbol{\alpha}^{j+1}$ and $\boldsymbol{\theta}^{j+1}$ are computed accordingly. Alternative update schemes such as RMSProp or Adam are available (e.g., [68]).

Training procedure

Each epoch *j* is composed of a data collection and a training phase. The data collection part consists of simulating a batch of N_b trajectories with the current network with weights $\boldsymbol{\theta}^j, \boldsymbol{\alpha}^j, \boldsymbol{\nu}^j$. We start by drawing initial samples $\boldsymbol{d}_0^{(i)}$ and $\boldsymbol{k}_0^{(i)}$ from their initial distributions and check for resulting failure costs $\boldsymbol{c}_{f,0}^{(i)}$. The actions $\boldsymbol{a}_0^{(i)}$ are fixed to a_0 (with action costs $\boldsymbol{c}_{a,0}^{(i)} = \boldsymbol{0}$), as observation-based action selection starts at $t = 1$. Then, $\boldsymbol{d}_0^{(i)}, \boldsymbol{k}_0^{(i)}, \boldsymbol{a}_0^{(i)}$ are passed to the environment which returns $\boldsymbol{d}_1^{(i)}$, $\boldsymbol{k}_1^{(i)}$ and $\boldsymbol{c}_{f,1}^{(i)}$ according to the dynamics in Appendix 1: Table 2. For $t = 1, ..., T_{\mathrm{end}-1}$, observations $\boldsymbol{o}_t^{(i)}$ are generated from $\mathcal{N}(\boldsymbol{d}_t^{(i)}, \sigma_E)$ and passed with $\boldsymbol{a}_{t-1}^{(i)}$ to the network which outputs the Q-values. The behaviour policy at epoch *j* selects the next action according to the ϵ-greedy scheme, where a random action is selected with probability ϵ (for exploration) and the action with minimal Q-value is selected with probability $1 - \epsilon$ (for exploitation). The chosen action $\boldsymbol{a}_t^{(i)}$ together with $\boldsymbol{d}_t^{(i)}$ and $\boldsymbol{k}_t^{(i)}$ is passed to the environment that simulates the system for one timestep and returns $\boldsymbol{d}_{t+1}^{(i)}$, $\boldsymbol{k}_{t+1}^{(i)}$ and $\boldsymbol{c}_{f,t+1}^{(i)}$. This alternating interaction between network and environment continues until the end of the system lifetime is reached. The samples $\boldsymbol{o}_{1:T_{\mathrm{end}-1}}^{(i)}$, $\boldsymbol{a}_{0:T_{\mathrm{end}-2}}^{(i)}$ and $\boldsymbol{c}_{0:T_{\mathrm{end}}}^{(i)} = \boldsymbol{c}_{f,0:T_{\mathrm{end}}}^{(i)} + \boldsymbol{c}_{a,1:T_{\mathrm{end}-1}}^{(i)}$ are then stored for the training phase.

Once a batch of sample trajectories has been collected, the training phase starts. Herein, the batch is again fed through the network sequentially, and the cost is accumulated over the whole life cycle. For the computation of the individual MSE loss terms, a target network is defined such that the values of the target network weights are clones of the original network weights: $\theta^{j,-} = \theta^j$, $\alpha^{j,-} = \alpha^j$, $\upsilon^{j,-} = \upsilon^j$. At each time *t*, $\boldsymbol{o}_t^{(i)}$ and $\boldsymbol{a}_{t-1}^{(i)}$ are the inputs of the network; $\boldsymbol{o}_{t+1}^{(i)}$ and $\boldsymbol{a}_t^{(i)}$ are the inputs of the target network. The target NN outputs are greedily selected over the respective Q-values (as opposed to the ϵ–greedy behaviour policy used for trajectory sampling, hence this is *off-policy* learning [69]) according to Eqs. (12 and (13). The batch cost at *t* is computed with a batch-averaged version of Eq. (14) and added to the total cumulative cost. This process continues until the end of the life cycle is reached, and the LCC MSE cost has been computed according to Eq. (15). Then, the LSTM is unrolled, the loss is backpropagated through time [65] and the weights are adjusted according to the chosen update scheme (e.g., Eq. (16). After updating, the learning procedure continues with the next epoch until the weights have converged. The weights of the target network are updated periodically every *p* epochs to ensure stable optimization [30].

The hyperparameter tuning procedure, either by grid search or by some heuristics, is outlined in Appendix 2.

MCTS

Functionality

Monte Carlo tree search (MCTS) arises from the combination of tree search and Monte Carlo sampling [70]. Classically, games have been modeled with game trees, where the root is the starting position, leaves are possible ending positions, and each edge represents a possible move [71]. To select the best action at a given node (position), one needs to know its consequences. Small games can be solved by constructing the full game tree and using backwards induction [72]. However, for more complex games (e.g., chess, Go), this is practically impossible. Hence, one needs an estimator of the preference for each resulting position. Defining the value of each node as an *expected outcome* given random play opened the door for the use of Monte Carlo, which specifies node values as random variables and characterizes game trees as probabilistic [73]. In [36], MCTS was extended to partially observable environments.

The MCTS algorithm consists of four main steps: selection, expansion, rollout, and backpropagation. In the selection step, the algorithm traverses the tree from the root to a leaf node using a selection policy (see UCT for action selection section). In the expansion step, the algorithm adds a child node to the selected leaf node. In the rollout step, the algorithm performs a simulation from the newly added node until the end of the lifetime by choosing uniformly random actions, i.e., $p(a_i) = \frac{1}{4}$. In the backpropagation step, the algorithm updates the statistics of all nodes along the path from the selected node to the root node based on the simulation outcome [74]. The Q-value of an action *a* for a given observation-action history *h* at time *t* is the updated statistic at an action and is computed as:

$$Q_t(h,a) \approx \frac{1}{N(h,a)} \sum_{i=1}^{N(h,a)} q_t^{(i)}(h,a), \tag{17}$$

where $N(h, a)$ is the total number of samples used for the estimation, or the current visitation counter of the respective action node, and $q_t^{(i)}(h, a)$ are the individual (backpropagated) results at time t.

UCT for action selection

To make use of the exploitation-exploration tradeoff [58], we implement the Upper Confidence Bound for Trees (UCT) algorithm. The UCT selects the next action A_t based on minimizing the estimation of the Q-value for each action (exploitation) minus an exploration term [75]:

$$A_t = \underset{a\in\mathcal{A}}{\arg\min} \mathrm{UCT}_t(h,a) = \underset{a\in\mathcal{A}}{\arg\min} Q_t(h,a) - c\cdot\sqrt{\frac{\ln N(h)}{N(h,a)}}, \tag{18}$$

where c is an adjustable constant that enables a tradeoff between exploration and exploitation, and $N(h)$ is the visitation counter of the parent node such that $N(h) = \sum_a N(h,a)$. A pseudocode for the implementation of MCTS for POMDPs is given in [36].

Each simulation starts by sampling an initial state from the current belief state, which is, for our case study, described in Eq. (19):

$$\begin{bmatrix} D_t \\ K_t \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu''_{D,t} \\ \mu''_{K,t} \end{bmatrix}, \begin{bmatrix} \sigma''^2_{D,t} & \rho''_t\sigma''_{D,t}\sigma''_{K,t} \\ \rho''_t\sigma''_{D,t}\sigma''_{K,t} & \sigma''^2_{K,t} \end{bmatrix} \right). \tag{19}$$

Silver and Veness [36] propose a samples-based approximation of the belief state for the general case when the belief state is not analytically available. We have not implemented this in the case study, hence one should keep in mind that an MCTS without the belief is likely to perform worse.

The tuning of the MCTS parameters is outlined in Appendix 3.

Results

Metrics for comparison

We employ several metrics to assess the performance of the NN and the MCTS approaches and to compare the results to the POMDP reference solution.

Firstly, we evaluate the computation time needed by the methods, including training and testing times.

Secondly, their computational performance is compared through the LCC's expected value and the standard deviation for the identified policies. Thereby, $\overline{\mathrm{LCC}}$ is approximated with Monte Carlo (MC) samples for both methods. The optimal solution curve obtained by evaluating the POMDP with VI (see POMDP reference solution section) serves as a reference. We additionally provide the performance of a benchmark policy that consists of choosing action a_1 in every timestep, irrespective of the observation.

Thirdly, we investigate the policies obtained from each method. The analysis comprises a statistical representation of the actions taken at each timestep to reveal potential tendencies, as well as a depiction in the belief space for policy extraction.

Computation time

All computations are performed on a Fujitsu Celcius R970 PC comprising an NVIDIA GP104GL (Quadro P4000) 8118 MB GPU and Intel Xeon Silver 4114 2.20 GHz: 10 Cores 20 Logical Processor. To accelerate the computation, training and testing of the NNs is conducted on the GPU, whereas MCTS is implemented with CPU parallelization.

With these specifications, the process of training and testing a single NN took 45 seconds (25 seconds of training and 20 seconds of testing 10^6 sample trajectories). In training, we consider different hyperparameter configurations following Appendix 2: Optimized NN parameters section, which leads to a total training time of approx. 150min.

By contrast, with MCTS there is no distinct training phase. Nevertheless, it is necessary to find good MCTS parameters, as described in Appendix 3: Tunable MCTS parameters section. This is a time-consuming process, because testing is expensive with MCTS. With the chosen parameter setting, generating 1000 trajectories for testing takes 20 minutes. For this reason, NN training is ultimately significantly cheaper and more straightforward.

Once the NN is trained or the MCTS setting is fixed, evaluating the policy is efficient. For NN, the computational time is negligible; for MCTS, it is in the order of seconds.

Performance

Figure 3 shows the mean LCC achieved by the +RQN, MCTS, VI, and the basic benchmark in function of the observation error. Firstly, all curves have a characteristic shape which consists of two saturation regions $\sigma_E < 0.5$ (essentially corresponding to perfect observations) and $\sigma_E > 10^3$ (uninformative observations) and a smooth transition in-between. Both the NN and MCTS methods perform worse than the optimal solution. However, the NN consistently outperforms the MCTS method, which performs especially poorly under high observation errors.

Figure 4 shows the standard deviation of the resulting LCC in function of the observation error. The standard deviation increases with increasing observation error, which is to be expected. The NN generally leads to a slightly higher LCC standard deviation than the VI reference solution, although with some exceptions. By contrast, the MCTS results in a low LCC standard deviation for small σ_E and in a very large one for large σ_E.

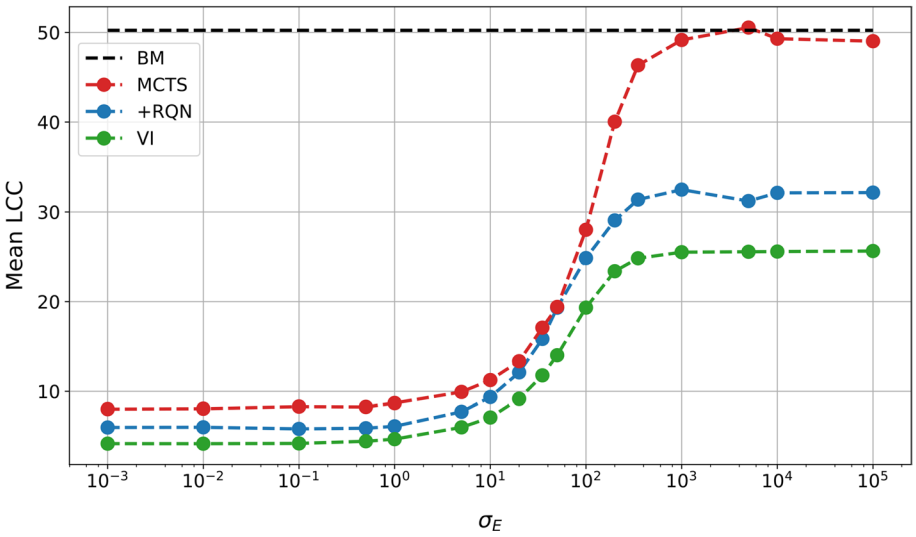

Fig. 3 Comparison of achieved mean LCC of +RQN (blue), MCTS (red), value iteration (green), and our a priori benchmark (black) for different measurement errors, where the policies of the +RQN, MCTS and VI are averaged over 10^6, 2×10^3 and 2×10^6 trajectories, respectively

Policy comparison

Figure 5 depicts the identified strategy profiles for the +RQN, MCTS, and VI in a statistical sense for the selected cases of $\sigma_E = \{0.5, 50\}$.

The reference VI method utilizes mainly a_1 in the first half of the system lifetime and employs a_2 in the second half. More maintenance is performed when the observation error is larger; for $\sigma_E = 50$, action a_1 is implemented early on in all cases, i.e., independent of the observation. Action a_3 is avoided, presumably due to its large cost.

The actions selected by the NN, as shown in panels (c) and (d), differ significantly from those of the reference solution. Note that the policies obtained with the NN vary substantially among repeated training runs, even if they lead to similar $\overline{\text{LCC}}$. The results in Fig. 5 correspond to a single trained NN for each observation error; with other trained NN instances, different proportions of a_0, a_1, a_2 are observed. In all trained NN, we observe that for $\sigma_E < 200$, the NN employs solely a_2 for failure prevention; a_1 is involved only for higher observation errors.

By contrast, MCTS has a similar strategy profile over all observation errors: about 30% use a_1 at every timestep. The only difference observed for larger observation errors is the increased use of a_2 in the second half of the system lifetime with increasing measurement errors. Interestingly,

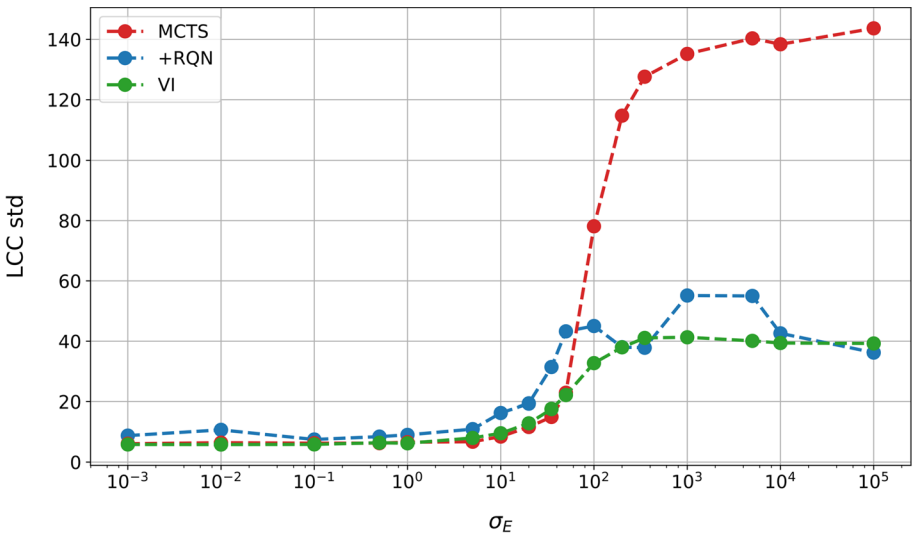

Fig. 4 Comparison of achieved LCC standard deviation of +RQN (blue), MCTS (red), value iteration (green) for different measurement errors, where the policies of the +RQN, MCTS and VI are averaged over 10^6, 2×10^3 and 2×10^6 trajectories, respectively

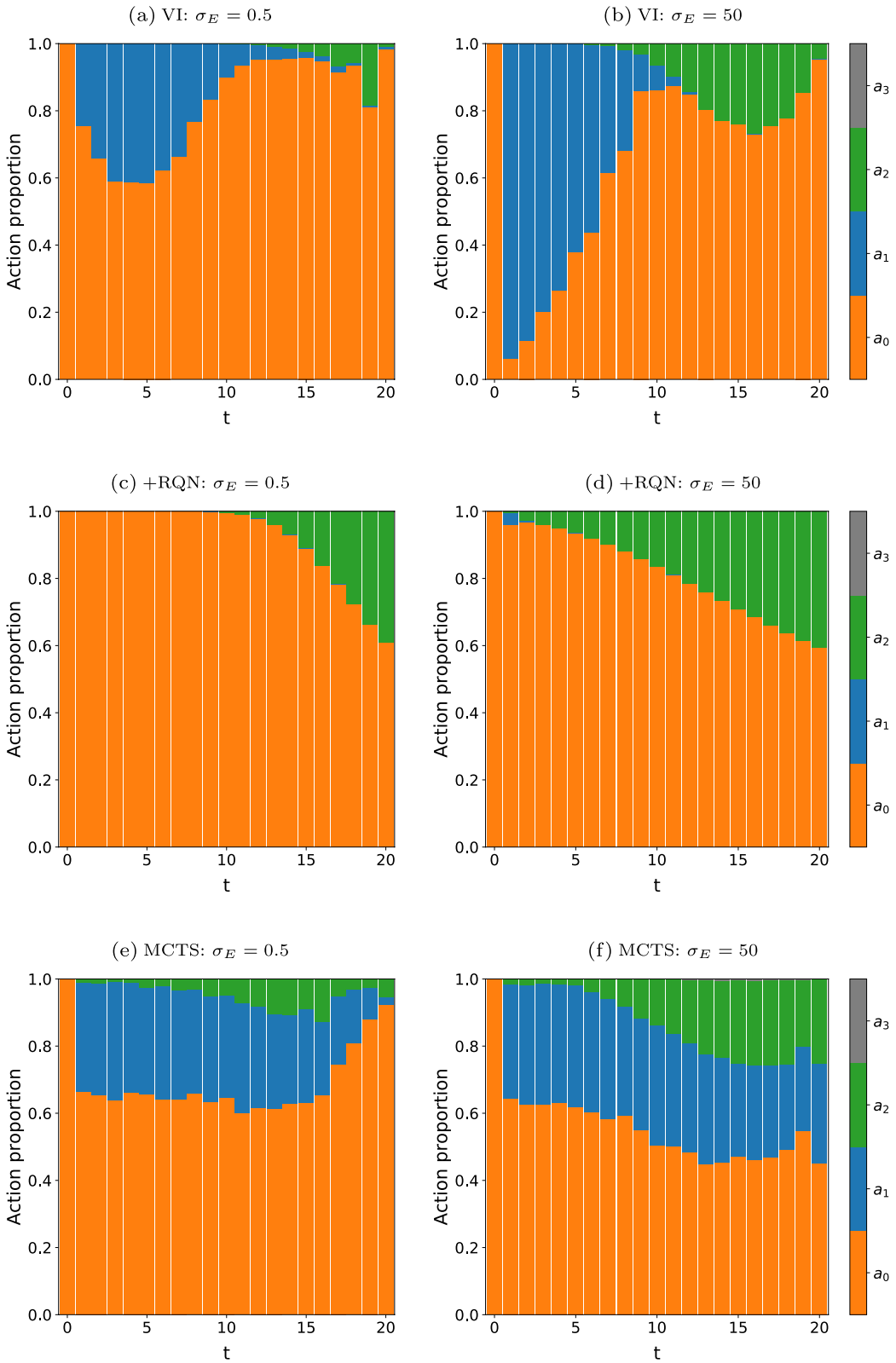

Fig. 5 Temporal evolution of the action statistics represented by bar charts for VI (a) & (b), +RQN (c) & (d), and MCTS (e) & (f) generated with $2 \times 10^6, 10^6, 2 \times 10^3$ and MC samples, respectively

for small σ_E, the statistic of the selected actions with MCTS is closer to the reference solution than the one of NN, even if the expected LCC achieved with the NN is smaller than the one achieved with MCTS.

To investigate and compare the resulting policies, we illustrate how the strategies manifest in the belief space. Figure 6 depicts the policies resulting from VI for the reference case of $\sigma_E = 50$ and $t = \{1, 10, 18, 20\}$. The occasional islands in otherwise continuous action bands in panels (a) and (b) result from the sampling-based estimation of the belief transition probabilities outlined in [62].

For comparison, we show the output of one run of the MCTS method (one for each $\boldsymbol{b}$ and t) in Fig. 7. The policies are similar to the VI policies in the choice of a_2 and a_3, i.e., the regions close to or beyond failure are primarily occupied with strips of a_2 and a_3. The extent of variation is determined by the magnitude of the measurement error as well as the remaining time until the end of the life cycle, e.g.,

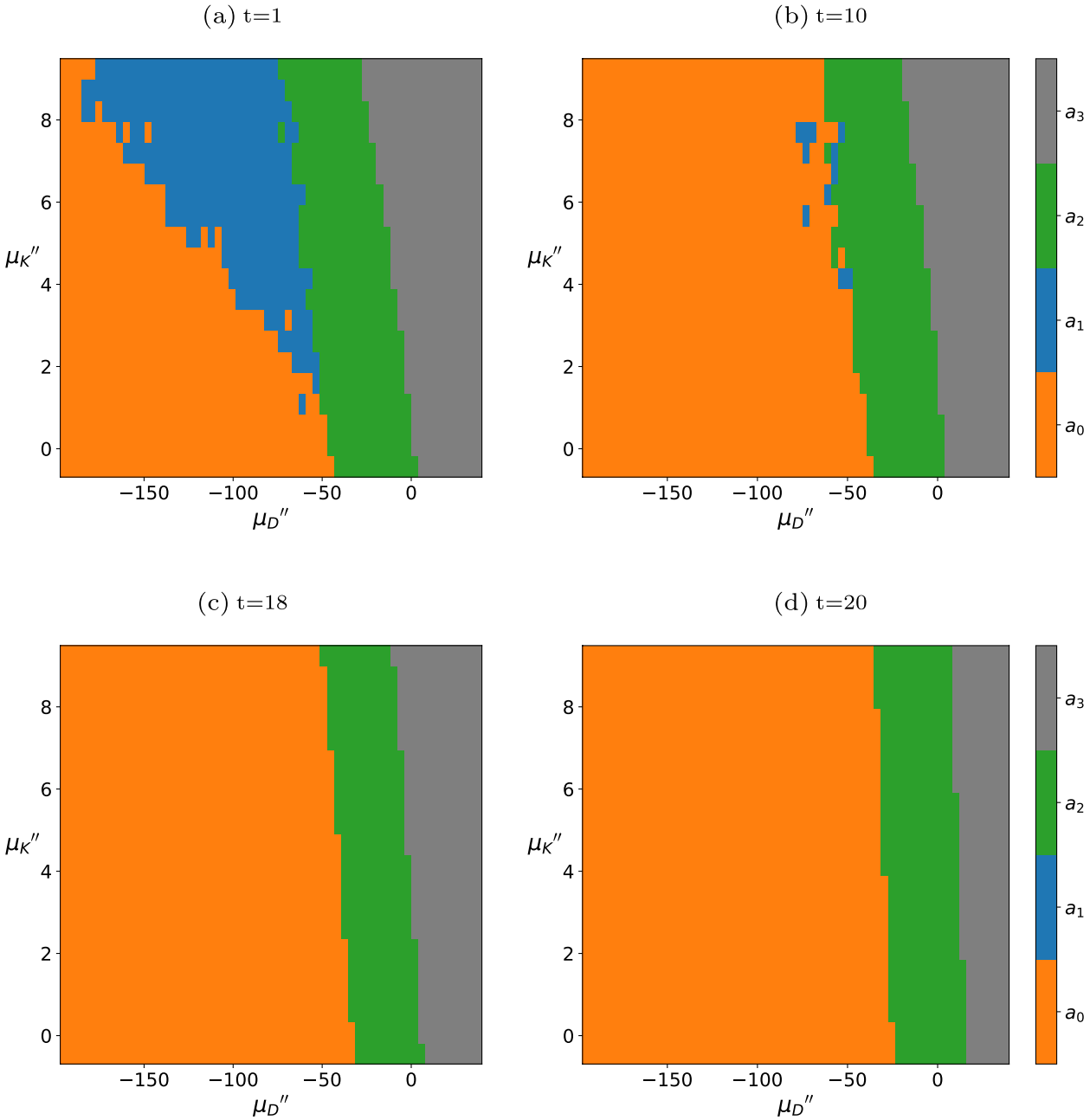

Fig. 6 Evolution of the VI optimal actions represented on a 60 × 20 belief grid for $t = 1$ (a), $t = 10$ (b), $t = 18$ (c), and $t = 20$ (d) for an observation error of $\sigma_E = 50$, where the cell mid-points are chosen as representatives for each cell region, respectively

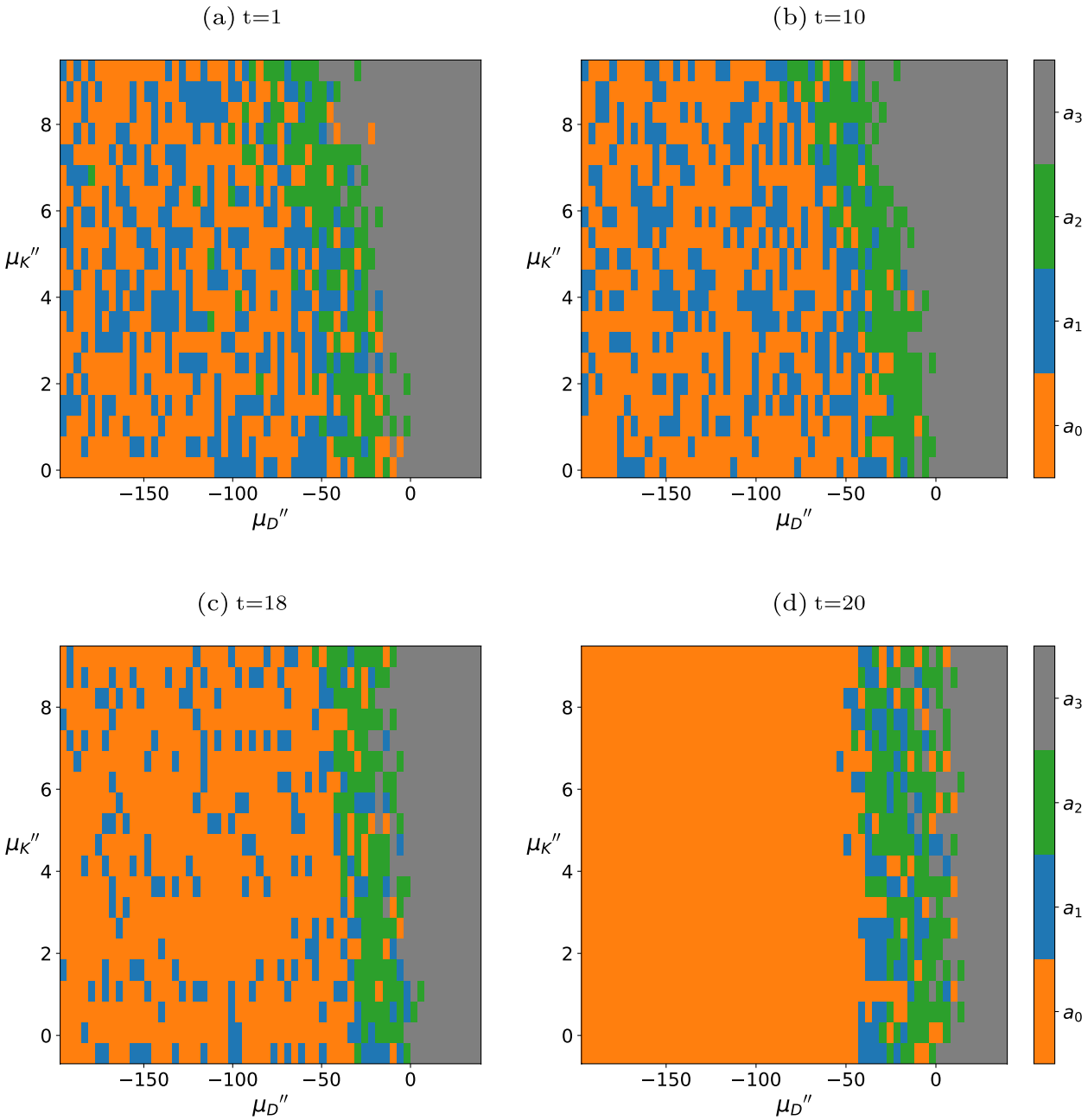

Fig. 7 Evolution of the MCTS recommended actions, same set-up as in Fig. 6

almost no variation for $\sigma_E << 1$ & $t > 10$, and high variation with no apparent structure for $\sigma_E > 100\ \forall t$. By contrast, the region far away from failure almost always shows high variability, and it seems that the choice between actions a_0 and a_1 is taken more or less randomly (except for very low σ_E at $t = 20$). The already mentioned variation tendencies for a_2 and a_3 also hold for a_0 and a_1. The large variance of MCTS (which could be reduced with increasing computational cost, see Appendix 3: MCTS parameter optimization technique section) leads to suboptimal policies.

For the NN, mapping all belief states to the optimal actions is not straightforward, as it takes observations and not beliefs as an input. However, the belief state can be tracked over time for sample trajectories, as shown in Fig. 8.

Once the trajectories in the belief space are available (Fig. 8), we can select a specific timestep and plot the actions taken by NN. This results in a point cloud in the belief space, which is shown in Fig. 9.

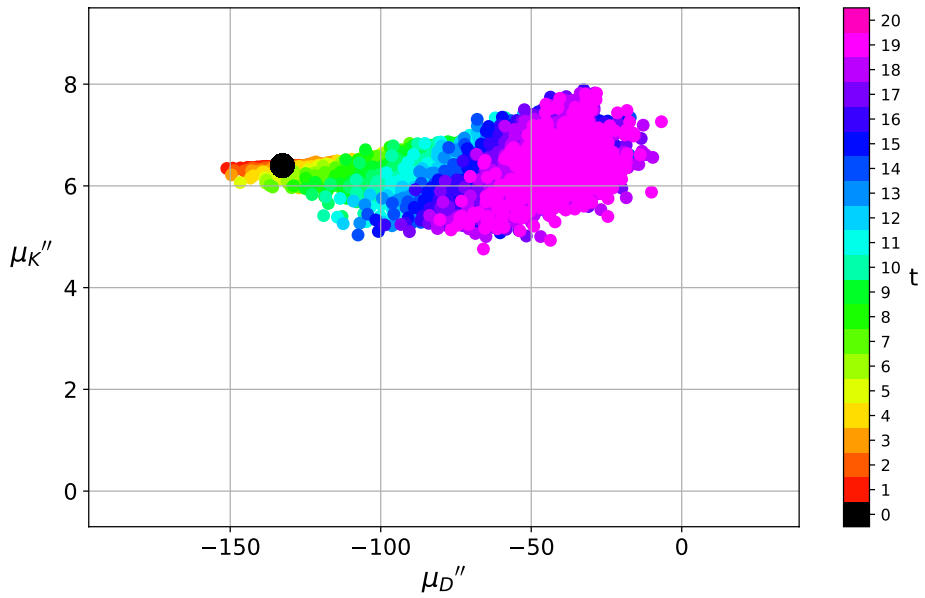

Fig. 8 Evolution of a batch of 500 trajectories through time, where each color represents a specific timestep according to the colorbar; generated with the NN trained on $\sigma_E = 50$

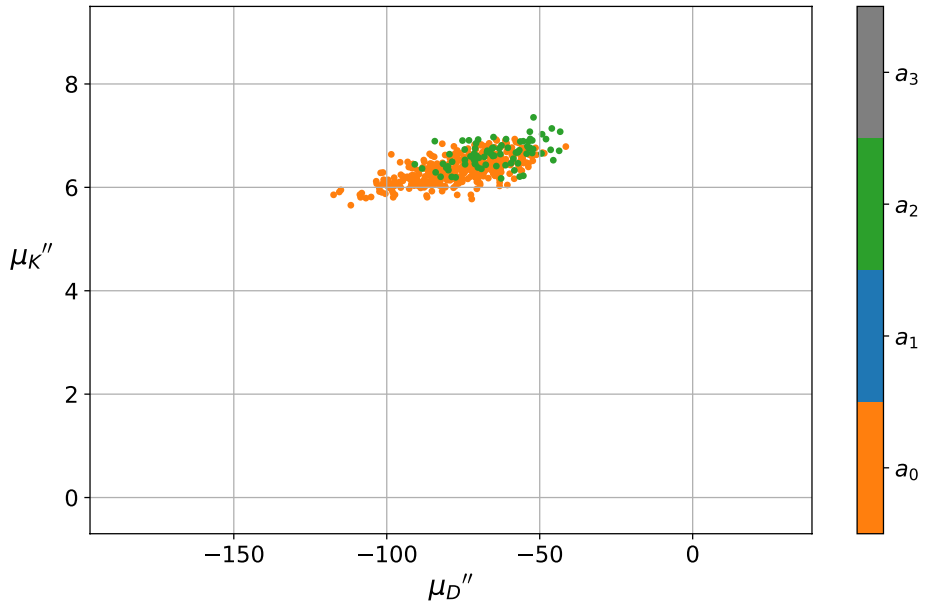

Fig. 9 NN snapshot of the suggested actions in the belief space for $\sigma_E = 50$ and 500 sample points at $t = 10$

Discussion

In this work, we develop a tailored NN architecture for solving the sequential decision making problem associated with maintenance of a component subject to deterioration. We evaluate its performance on a single component maintenance problem with continuous state space. We also compare the performance of an MCTS approach on this example.

There are several deep reinforcement learning approaches in the literature, some of which also solve the optimal inspection and maintenance problem for systems with many components [16, 20, 39, 40, 45]. These approaches work on discrete state spaces and compute the belief to translate the problem into a Markov decision process. The motivation for investigating the comparably simpler problem in this paper is our interest in approaches that work with continuous state spaces without a belief (even if the problem that we consider

actually has an easily tractable belief, which facilitates the evaluation of the algorithms in our investigation).

Computation time-wise, the NNs vastly outperform MCTS. This can be partly attributed to the implementation: PyTorch tensor operations on the GPU for NNs are much faster compared to standard Python list implementation on multiple CPUs for MCTS. The other part can be attributed to the nature of the methods: passing a state-action pair through the NN and retrieving the next action via the Q-values is much faster than performing a tree search for the next action at every timestep.

The results of our numerical investigation show that both the NN architecture as well as the MCTS approach perform suboptimally compared to the reference solution found by value iteration. It is possible to improve the performance of both approaches, in the case of NN, by additional training and hyperparameter tuning, and in the case of MCTS, by employing a larger number of samples. However, our results reflect an honest assessment of the capabilities of these methods.

The NN's solution highly depends on the local minimum found during training. This explains the non-smooth standard deviation curve and the large differences of resulting statistical strategy profiles in different training runs, as reflected in Fig. 4. Generally, the NN's strategy profile changes considerably for $\sigma_E > 100$ as the NN approaches the solution for the case of uninformative observations.

As evidenced by Fig. 9, the NN's policy is stochastic in the belief space, although it is deterministic in the observation space. As the optimal policy is deterministic in the belief space (as given by the VI solution shown in Fig. 6), one can observe that the trained NN is not yet able to capture implicitly the underlying belief space, which is one reason for its suboptimality. Thus, if the belief can be computed, it should be used as an input to the NN, as this will strongly enhance its performance (see, e.g., [46]) and facilitate interpretability.

The MCTS provides suboptimal but still decent results for $\sigma_E \leq 50$, where it trades $\overline{\text{LCC}}$ for lower variance. This can also be seen in Fig. 5, where the NN employs only a_2 leading to an overall lower mean cost but higher variance due to the acceptance of occasional failures. For higher observation errors, the MCTS performance decreases significantly, showing a limited ability to handle uninformative observations. This is exemplified by slightly changing strategy profiles with increasing σ_E in Fig. 5. Interestingly, Figs. 7 and 6 show that the MCTS' general solution is similar to the VI optimal solution provided. However, the inherent stochasticity of the method results in a stochastic policy in the belief space. This property is most apparent at the beginning of the life cycle, where the long-term effects of some actions are difficult to estimate.

A disadvantage of the MCTS approach is that it has no memory; thus, each sample trajectory has to be computed independently and expensively. By contrast, NNs, once trained, contain all the information in the weights, and the evaluation can be performed swiftly. In addition, we speculate that this memorylessness of the MCTS leads to worse performance compared to the NNs, which can learn the degradation behaviour through observed trajectories.

Overall, and possibly expected, the neural networks are the preferred choice. However, there are numerous opportunities for further enhancements of both solution approaches.

The performance investigation of the NN could be extended, for example, by studying its dependence on the network size, its generalization capabilities (e.g., increased lifetime, different distributions), or by using the belief as an input instead of the observations for comparison. Moreover, the NN architecture can be extended by incorporating a double deep Q-network (DDQN) or by replacing the LSTM architecture with transformers (see, e.g., [46, 47]).

The MCTS method could be extended by, e.g., using erroneous observations instead of exact beliefs [36] for performance comparison or by switching to continuous state MCTS to dispense with discretization. NN and MCTS can also be combined by adding a planning step to the NN-based solution.

Conclusion

In this work, we propose the +RQN architecture for POMDP and I &M planning, which requires merely the erroneous observations and the previous action taken as an input. The resulting neural networks are computationally fast and achieve good performance for measurement errors over several magnitudes through policy adaption. However, NNs, in general, inherently suffer from interpretation difficulties. The trained model consists already for small problems of thousands of weights. Interpreting the results or gaining underlying physical insights and properties of the system is non-trivial. This characteristic is evident in policy extraction, which is challenging to conduct in the belief space, as beliefs cannot be imposed but only tracked along the NN's trajectories.

By contrast, computing many histories with the MCTS method is computationally much slower. In addition, it is inherently based on constructing a tree that exponentially grows with increasing depth, which needs large amounts of memory. The results of the MCTS are comparable to the NNs for small to medium observation errors. However, for high observation errors, the MCTS method fails to adapt its policy and achieves significantly worse results compared to the NNs and VI. The key advantage of the MCTS method lies in the evaluation of their policies. Any belief

combination can be specified as a starting point which greatly facilitates the interpretation of the results.

Appendix 1: Model Info

Model data

The specific parameters for the model used in this work are outlined in Appendix 1: Table 1.

Table 1 Summary of model and cost parameters

Model		Costs	
Parameter	**Value**	**Parameter**	**Value**
μ_{D_0}	−132.64	c_{a_0}	0
μ_{K_0}	6.4	c_{a_1}	1
σ_{D_0}	20.85	c_{a_2}	5
σ_{K_0}	1	c_{a_3}	100
Δd	10.5	c_F	150
Δk	0.2	γ	$\frac{1}{1.02}$

Effect of actions

The (belief) state of the system is influenced by the four available actions, whose effects are detailed in Appendix 1: Table 2.

Table 2 Mathematical description of the action a_i effects on individual D and K states as well as their corresponding beliefs μ_D and μ_K. At a_3, the replacement is conducted by drawing new samples $\tilde{D}_0$ and $\tilde{K}_0$ (from the distribution in Eq. (20)) and not by reusing the samples from the current simulation

Action	Effect	State level	Belief level
a_0	do	$D_t = D_{t-1} + K_{t-1}$	$\mu'_{D_t} = \mu''_{D_{t-1}} + \mu''_{K_{t-1}}$
	nothing	$K_t = K_{t-1}$	$\mu'_{K_t} = \mu''_{K_{t-1}}$
a_1	reduce	$D_t = D_{t-1} + K_{t-1} - \Delta k$	$\mu'_{D_t} = \mu''_{D_{t-1}} + \mu''_{K_{t-1}} - \Delta k$
	det. rate	$K_t = K_{t-1} - \Delta k$	$\mu'_{K_t} = \mu''_{K_{t-1}} - \Delta k$
a_2	improve	$D_t = D_{t-1} + K_{t-1} - \Delta d$	$\mu'_{D_t} = \mu''_{D_{t-1}} + \mu''_{K_{t-1}} - \Delta d$
	det. state	$K_t = K_{t-1}$	$\mu'_{K_t} = \mu''_{K_{t-1}}$
$^*a_3$	replace	$D_t = \tilde{D}_0 + \tilde{K}_0$	$\mu'_{D_t} = \mu''_{D_0} + \mu''_{K_0}$
	system	$K_t = \tilde{K}_0$	$\mu'_{K_t} = \mu''_{K_0}$

Action a_3 consists of sampling new values for the deterioration state and deterioration rate from the following multivariate normal distribution:

$$\left.\begin{bmatrix} D_t \\ K_t \end{bmatrix}\right|_{a_{3,t-1}} \sim \mathcal{N}_2\left(\begin{bmatrix} \mu_{D_0} + \mu_{K_0} \\ \mu_{K_0} \end{bmatrix}, \begin{bmatrix} \sigma'^2_{D_t} & \rho'_t \sigma'_{D_t} \sigma'_{K_t} \\ \rho'_t \sigma'_{D_t} \sigma'_{K_t} & \sigma'^2_{K_t} \end{bmatrix}\right), \tag{20}$$

where we denote with "′" and "″" the prior and posterior distributions, respectively. The corresponding analytical terms are detailed in the following.

Transition probabilities - state level

At every timestep $t \geq 1$, after observing O_t, the updated distribution of D_t and K_t is a binormal distribution, with mean $\mu''_{D,t}$, $\mu''_{K,t}$, standard deviations $\sigma''_{D,t}$, $\sigma''_{K,t}$ and correlation coefficient ρ''_t.

Prior and posterior covariance matrix of D_t and K_t

For the covariance matrix, the transition from $''_{t-1}$ to $''_t$ does not depend on O_t or A_t, hence is deterministic:

$$\sigma'_{D,t} = \sqrt{\sigma''^2_{K,t-1} + \sigma''^2_{D,t-1} + 2\rho''_{t-1}\sigma''_{K,t-1}\sigma''_{D,t-1}} \tag{21}$$

$$\sigma''_{D,t} = \frac{\sigma_E \sigma'_{D,t}}{\sqrt{\sigma_E^2 + \sigma'^2_{D,t}}} \tag{22}$$

$$\sigma'_{K,t} = \sigma''_{K,t-1} \tag{23}$$

$$\sigma''_{K,t} = \frac{\sigma'_{K,t}\sqrt{\sigma_E^2 + \sigma'^2_{D,t}\left(1 - \rho'^2_t\right)}}{\sqrt{\sigma_E^2 + \sigma'^2_{D,t}}} \tag{24}$$

$$\rho'_t = \frac{\rho''_{t-1}\sigma''_{D,t-1} + \sigma''_{K,t-1}}{\sqrt{\sigma''^2_{K,t-1} + \sigma''^2_{D,t-1} + 2\rho''_{t-1}\sigma''_{K,t-1}\sigma''_{D,t-1}}} \tag{25}$$

$$\rho''_t = \frac{\rho'_t \sigma_E}{\sqrt{\sigma_E^2 + \sigma'^2_{D,t}\left(1 - \rho'^2_t\right)}}. \tag{26}$$

Posterior mean values of D_t and K_t

Conversely to the covariance matrix, the posterior mean values of D_t and K_t depend on the value of the observation O_t

$$\mu''_{D,t} = \frac{\sigma''^2_{D,t}}{\sigma_\epsilon^2} O_t + \frac{\sigma''^2_{D,t}}{\sigma'^2_{D,t}} \mu'_{D,t} \tag{27}$$

$$\mu''_{K,t} = \frac{\rho'_t \sigma'_{D,t} \sigma'_{K,t}}{\sigma_\epsilon^2 + \sigma'^2_{D,t}} (O_t - \mu'_{D,t}) + \mu'_{K,t}. \tag{28}$$

Transition probabilities - belief level

The covariance of D_t and K_t is fully known (does not depend on O_t). The means of the distributions are fully observed at each timestep (see Eqs. (27) and (28)). The belief B_t at time t is composed of the two posterior means,

$\mu''_{D,t}$ and $\mu''_{K,t}$. From Eq. (27) and $O_t|D_t \sim \mathcal{N}(D_t, \sigma_\epsilon)$, which gives $O_t \sim \mathcal{N}(\mu'_{D,t}, \sqrt{\sigma_\epsilon^2 + \sigma_{D,t}'^2})$, we obtain that

$$\mu''_{D,t}|B_{t-1}, A_t \sim \mathcal{N}\left[\mu'_{D,t}(B_{t-1}, A_t), \frac{\sigma_{D,t}'^2}{\sqrt{\sigma_\epsilon^2 + \sigma_{D,t}'^2}}\right]. \tag{29}$$

One can show that $\mu''_{K,t}$ is fully correlated with $\mu''_{D,t}$ conditional on the belief at t_1:

$$\mu''_{K,t} = \frac{\rho'_t \sigma'_{K,t}}{\sigma'_{D,t}}(\mu''_{D,t} - \mu'_{D,t}) + \mu'_{K,t}. \tag{30}$$

Appendix 2: Neural network specifications

Fixed NN parameters

The number of hidden layers loosely follows the architecture from [25]. It is possible that the network achieves better performance or equal performance with shorter training time with other configurations.

The output dimensions of O, a, A, V, and Q are fixed by our problem formulation, i.e., we have one observation variable and four available one-hot encoded actions. The output dimensions of all other layers - the three FC layers and the LSTM layer - can be freely chosen. The dimensions of all customizable layers have been selected heuristically. The fully connected layers are numbered according to the order in which they appear from left to right, i.e., there are two FC1 and FC2 layers. The sizes of FC2 (and FC1) have been chosen to have the same dimension to not impose an ad hoc ranking of importance before they enter the LSTM layer. The exact values for the number of nodes in each layer and all other parameters set heuristically are given in Appendix 2: Table 3.

Table 3 Summary of new heuristically chosen network and optimizer parameters

Network		Optimizer	
Parameter	**Value**	**Parameter**	**Value**
FC1 size	20	Optim. type	Adam [76]
FC2 size	25	Learning rate	0.001 [77]
Hidden state size	80	Betas	(0.9, 0.999) [77]
FC3 size	160	AMSGRAD	Included [78]
FC activation funcs.	Leaky ReLU	Batch size	500
Leaky ReLU slope	0.3	Epochs	500 (at most)
Target update	3	ϵ decrease	0.1

The total number of parameters of our NN architecture for the specific values given in Appendix 2: Table 3 is 57,195.

We train the networks for at most 500 epochs. However, early stopping is also implemented, i.e., training is interrupted if the training loss does not further decrease over an extended period [68].

Optimized NN parameters

Our chosen parameters to optimize are given in the following list.

1. Weight decay parameter λ (L2 regularization) [68]
2. Maximum ϵ value (coupled with a decrease)
3. Learning rate step size
4. Learning rate multiplication factor

Including weight decay, the loss function gets an additional term:

$$\mathcal{L} = \mathcal{L}_{\text{MSE}} + \lambda \cdot R(\boldsymbol{W}), \tag{31}$$

where $\boldsymbol{W}$ is a matrix containing all network weights, R denotes the regularization function, which is the squared sum of all network weights (L_2), and λ is a scaling parameter determining the relative importance of the regularization term compared to the MSE loss. We search for an optimal value of λ.

We implement our behaviour policy, i.e., the policy with which we select the next action when generating a batch of trajectories, as a decreasing ϵ-greedy method which starts with the value ϵ in the beginning to fuel exploration but decreases to 0 for exploitation of the final policy. However, one can also choose a different minimum ϵ-value (e.g., 0.1 in [25]) to always force some exploration. Our update scheme takes the form of:

$$\epsilon \longleftarrow \epsilon - 0.1. \tag{32}$$

Therefore, our scheme implements a simple linear reduction. The starting value of ϵ is optimized.

We also implement a learning rate scheduler, where the learning rate starts at a high value and periodically decreases, which helps both generalization and optimization [79]. We implement a simple step decay schedule that reduces the learning rate by a constant factor η every constant number of epochs m [80]. Hence, we search for the optimal values of m and η.

There are plenty more common practices for training NNs, e.g., weight initialization, batch normalization, and dropout. For most of these, we follow the default settings of PyTorch; these will not be further explained here.

NN Optimization technique

Several search techniques can be employed to find good NN hyperparameters. The most common one is manual search, which is simple and effective for finding reasonable estimates (e.g., initial learning rate), but becomes unstructured and ineffective when the search space of the parameters to tune grows. Therefore, we use grid search, where we define a set of points for each of our desired hyperparameters and iterate over all possible combinations [68]. During the procedure, we

track the performances of each network and select the best-performing one.

Appendix 3: MCTS tuning

A number of parameters influence the performance of the tree search method and hence need to be optimized. The disadvantage of MCTS compared to NNs is that parameters cannot be passed as an input, and the method finds the optimal values by itself. In addition, it takes too much time to generate an accurate representation of the performance of the tree; therefore, we cannot use an extensive grid search as we did with the NNs (Appendix 2: NN Optimization technique section). Thus, we try to minimize the number of parameters we need to optimize. The remaining parameters are then analysed sequentially with appropriate assumptions.

Fixed MCTS parameters

The variable c in Eq. (18) is also called the *exploration constant*, since it expresses the weight of exploration (second term) compared to exploitation (first term). When $c = 0$, one has a purely greedy policy [81]. On the other hand, when $c \longrightarrow \infty$, one has a purely exploratory policy. We conveniently set $c = 1$, but other approaches exit (see, e.g., [36].

The next parameters that we fix are the upper and lower bounds of the observation buckets. The MCTS algorithm works with discrete observations, but our case study concerns a continuous deterioration and a continuous observation space. Although there exist MCTS variations which can deal with continuous action and state spaces (see, e.g., [82]), we can easily transform our problem to the discrete space by bucketing our observations, i.e., a certain bucket points to a range of observations. The question that now arises is how to choose these buckets. Generally, the bucket size does not have to be constant, but for simplicity, we choose buckets of equal size (with the exception of the first and last bucket). Therefore, we only need to define the ceiling (d_{ce}) and floor (d_{fl}) bound, as well as the number of desired observation buckets to fully define our buckets. The general case of N_{ob} equal-sized observation buckets is depicted in Appendix 3: Fig. 10:

Thus, we need to find some reasonable values for the floor and ceiling values d_{fl} and d_{ce}. We can relate d_{fl} to a percentile of the initial distribution of D, and d_{ce} to a percentile of the final distribution of D when letting the system evolve without intervention (i.e., the "worst" case). This leads to:

$$d_{fl} = -159.36 \tag{33}$$

$$d_{ce} = 26.67. \tag{34}$$

Tunable MCTS parameters

There are further parameters that we do not set a priori but still highly influence the MCTS' performance. The parameters to be optimized are given in the following list.

1. Tree iterations N_T
2. Rollout runs N_R
3. Observation buckets N_{ob}

The number of tree iterations N_T dictates the depth the tree reaches, i.e., the number of timesteps it looks into the future. In addition, a higher number of tree iterations increases the accuracy of the Q-value estimate. However, the possible number of nodes in our tree grows exponentially with increasing depth, and one can also increase the accuracy with the number of rollout runs N_R from a given system state. Averaging over multiple rollouts instead of relying on a single run greatly reduces the susceptibility to high variances resulting from large differences in action and failure costs.

Lastly, the number of observation buckets N_{ob} is also a crucial parameter, as it influences the reachable depth of the tree given a fixed number of tree iterations. In addition, it represents the degree of precision with which the observations are discretized. Hence, it is essential to find the right balance between depth and resolution.

MCTS parameter optimization technique

What remains now is to outline an optimization procedure for the three parameters of Appendix 3: Tunable MCTS parameters section considering the observation

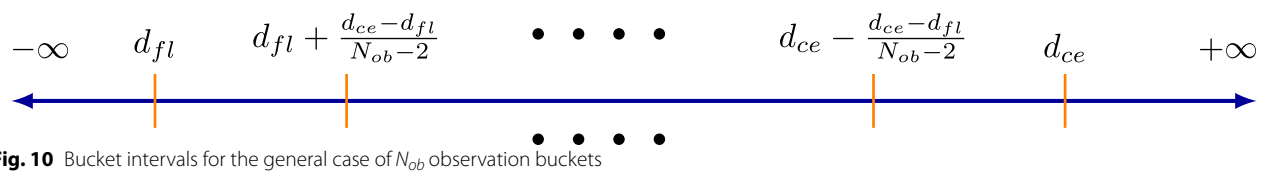

Fig. 10 Bucket intervals for the general case of N_{ob} observation buckets

error. Generally, it is assumed that the optimal number of observation buckets is dependent on the observation error with regards to minimizing the LCC.

The first analysis is conducted on the time dependence of N_T and N_R, where we impose some threshold of computation time needed to traverse a whole life cycle with the MCTS method to stay in a computationally feasible domain. It is assumed that the computation time is independent of the observation error and is only minimally affected by the choice of N_{ob}, which is why they are fixed.

The result of the analysis is a set of different possible combinations of the two parameters, which satisfy our imposed computation threshold. To settle for a single combination, the influence of N_T and N_R on the LCC will be taken into account. We assume that the resulting curves qualitatively hold for any N_{ob} and σ_E, which is why they are fixed again.

Secondly, once N_T and N_R have been fixed with the time constraints and LCC maximization, we search for the optimal number of observation buckets given a set of observation errors of interest.

Abbreviations

+RQN	Action-specific Deep Dueling Recurrent Q-network
DCMAC	Deep Centralized Multi-agent Actor Critic
DDMAC	Deep Decentralized Multi-agent Actor Critic
DRL	Deep Reinforcement Learning
DQN	Deep Q-Network
DDQN	Double Deep Q-Network
FC	Fully Connected
I[MYAMP	M] Inspection and Maintenance
LCC	Life Cycle Cost
LSTM	Long Short-Term Memory
MC	Monte Carlo
MCTS	Monte Carlo Tree Search
MDP	Markov Decision Process
MSE	Mean-Squared Error
NN	Neural Network
POMDP	Partially Observable Markov Decision Process
RL	Reinforcement Learning
RV	Random Variable
UCT	Upper Confidence Bound for Trees
VI	Value Iteration

Authors' contributions

D.K. worked on the investigation and visualization. D.K. and E.B. developed the methodology, software, and the original draft of the manuscript. E.B. and D.S. supervised and validated the work. All authors worked on the conceptualization, reviewed and edited the manuscript.

Funding

Open Access funding enabled and organized by Projekt DEAL. The study was partially supported by the TUM Georg Nemetschek Institute Artificial Intelligence for the Built World.

Availability of data and materials

The environment used and/or analysed during the current study are available from the corresponding author on reasonable request.

Declarations

Competing interests

The authors declare no competing interests.

Received: 9 August 2023 Revised: 23 February 2024 Accepted: 9 April 2024
Published online: 29 April 2024

References

1. Rioja F (2013) What Is the Value of Infrastructure Maintenance? A Survey. Infrastruct Land Policies 13:347–365
2. Daniela L, Di Sivo M (2011) Decision-support tools for municipal infrastructure maintenance management. Procedia Comput Sci 3:36–41
3. Frangopol DM, Kallen MJ, Noortwijk JMV (2004) Probabilistic models for life-cycle performance of deteriorating structures: review and future directions. Prog Struct Eng Mater 6(4):197–212
4. Bismut E, Straub D (2021) Optimal Adaptive Inspection and Maintenance Planning for Deteriorating Structural Systems. Reliab Eng Syst Saf 215:107891
5. Straub D (2021) Lecture Notes in Engineering Risk Analysis. Technical University of Munich, Germany
6. Sullivan TJ (2015) Introduction to Uncertainty Quantification, vol 63. Springer
7. Madanat S (1993) Optimal infrastructure management decisions under uncertainty. Transp Res C Emerg Technol 1(1):77–88
8. Luque J, Straub D (2019) Risk-based optimal inspection strategies for structural systems using dynamic Bayesian networks. Struct Saf 76:68–80
9. Melchers RE, Beck AT (2018) Structural reliability analysis and prediction. Wiley
10. Rausand M, Hoyland A (2003) System reliability theory: models, statistical methods, and applications, vol 396. Wiley
11. ASCE (2021) 2021 Report Card for America's Infrastructure; Energy. https://infrastructurereportcard.org/wp-content/uploads/2020/12/Energy-2021.pdf. Accessed 17 July 2022
12. Yuen KV (2010) Bayesian Methods for Structural Dynamics and Civil Engineering. Wiley
13. Kim S, Frangopol DM, Soliman M (2013) Generalized Probabilistic Framework for Optimum Inspection and Maintenance Planning. J Struct Eng 139(3):435–447
14. Kim S, Frangopol DM, Zhu B (2011) Probabilistic Optimum Inspection/Repair Planning to Extend Lifetime of Deteriorating Structures. J Perform Constr Facil 25(6):534–544
15. Kochenderfer MJ (2015) Decision Making Under Uncertainty: Theory and Application. MIT Press, Cambridge
16. Andriotis C, Papakonstantinou K (2021) Deep reinforcement learning driven inspection and maintenance planning under incomplete information and constraints. Reliab Eng Syst Saf 212:107551
17. Kaelbling LP, Littman ML, Cassandra AR (1998) Planning and acting in partially observable stochastic domains. Artif Intell 101(1–2):99–134
18. Papadimitriou CH, Tsitsiklis JN (1987) The Complexity of Markov Decision Processes. Math Oper Res 12(3):441–450
19. Meng L, Gorbet R, Kulić D (2021) Memory-based Deep Reinforcement Learning for POMDPs. In: 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, pp 5619–5626
20. Andriotis C, Papakonstantinou K (2019) Managing engineering systems with large state and action spaces through deep reinforcement learning. Reliab Eng Syst Saf 191:106483
21. Schöbi R, Chatzi EN (2016) Maintenance planning using continuous-state partially observable Markov decision processes and non-linear action models. Struct Infrastruct Eng 12(8):977–994

22. Corotis RB, Hugh Ellis J, Jiang M (2005) Modeling of risk-based inspection, maintenance and life-cycle cost with partially observable Markov decision processes. Struct Infrastruct Eng 1(1):75–84
23. Hausknecht M, Stone P (2015) Deep Recurrent Q-Learning for Partially Observable MDPs. In: 2015 AAAI fall symposium series
24. Lample G, Chaplot DS (2017) Playing FPS Games with Deep Reinforcement Learning. In: Thirty-First AAAI Conference on Artificial Intelligence
25. Zhu P, Li X, Poupart P, Miao G (2017) On Improving Deep Reinforcement Learning for POMDPs. arXiv preprint arXiv:170407978
26. Song DR, Yang C, McGreavy C, Li Z (2018) Recurrent Deterministic Policy Gradient Method for Bipedal Locomotion on Rough Terrain Challenge. In: 2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV). IEEE, pp 311–318
27. Wang C, Wang J, Shen Y, Zhang X (2019) Autonomous Navigation of UAVs in Large-Scale Complex Environments: A Deep Reinforcement Learning Approach. IEEE Trans Veh Technol 68(3):2124–2136
28. Duan Y, Chen X, Houthooft R, Schulman J, Abbeel P (2016) Benchmarking Deep Reinforcement Learning for Continuous Control. In: International conference on machine learning. PMLR, pp 1329–1338
29. Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, Riedmiller M (2013) Playing atari with deep reinforcement learning. arXiv preprint arXiv:13125602
30. Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G et al (2015) Human-level control through deep reinforcement learning. Nature 518(7540):529–533
31. Brim A (2020) Deep Reinforcement Learning Pairs Trading with a Double Deep Q-Network. In: 2020 10th Annual Computing and Communication Workshop and Conference (CCWC). IEEE, pp 0222–0227
32. Lv P, Wang X, Cheng Y, Duan Z (2019) Stochastic double deep q-network. IEEE Access 7:79446–79454
33. Haarnoja T, Zhou A, Abbeel P, Levine S (2018) Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: International conference on machine learning. PMLR, pp 1861–1870
34. Silver D, Huang A, Maddison CJ, Guez A, Sifre L, Van Den Driessche G, Schrittwieser J, Antonoglou I, Panneershelvam V, Lanctot M et al (2016) Mastering the game of Go with deep neural networks and tree search. Nature 529(7587):484–489
35. Silver D, Hubert T, Schrittwieser J, Antonoglou I, Lai M, Guez A, Lanctot M, Sifre L, Kumaran D, Graepel T et al (2018) A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. Science 362(6419):1140–1144
36. Silver D, Veness J (2010) Monte-Carlo Planning in Large POMDPs. Adv Neural Inf Process Syst 23:2164–2172
37. Katt S, Oliehoek FA, Amato C (2017) Learning in POMDPs with Monte Carlo Tree Search. In: International Conference on Machine Learning. PMLR, pp 1819–1827
38. Shao K, Tang Z, Zhu Y, Li N, Zhao D (2019) A Survey of Deep Reinforcement Learning in Video Games. arXiv preprint arXiv:191210944
39. Zhou W, Miller-Hooks E, Papakonstantinou KG, Stoffels S, McNeil S (2022) A Reinforcement Learning Method for Multiasset Roadway Improvement Scheduling Considering Traffic Impacts. J Infrastruct Syst 28(4):04022033
40. Saifullah M, Andriotis C, Papakonstantinou K, Stoffels S (2022) Deep reinforcement learning-based life-cycle management of deteriorating transportation systems. In: Bridge Safety, Maintenance, Management, Life-Cycle, Resilience and Sustainability. CRC Press, pp 293–301
41. Skordilis E, Moghaddass R (2020) A Deep Reinforcement Learning Approach for Real-time Sensor-Driven Decision Making and Predictive Analytics. Comput Ind Eng 147:106600
42. Huang J, Chang Q, Arinez J (2020) Deep Reinforcement Learning based Preventive Maintenance Policy for Serial Production Lines. Expert Syst Appl 160:113701
43. Nguyen VT, Do P, Vosin A, Iung B (2022) Artificial-intelligence-based maintenance decision-making and optimization for multi-state component systems. Reliab Eng Syst Saf 228:108757
44. Mohammadi R, He Q (2022) A deep reinforcement learning approach for rail renewal and maintenance planning. Reliab Eng Syst Saf 225:108615
45. Morato PG, Andriotis CP, Papakonstantinou KG, Rigo P (2023) Inference and dynamic decision-making for deteriorating systems with probabilistic dependencies through Bayesian networks and deep reinforcement learning. Reliability Engineering & System Safety, vol 235. Elsevier, pp 109144
46. Arcieri G, Hoelzl C, Schwery O, Straub D, Papakonstantinou KG, Chatzi E (2023) POMDP inference and robust solution via deep reinforcement learning: An application to railway optimal maintenance. submitted to Machine Learning
47. Hettegger D, Buliga C, Walter F, Bismut E, Straub D, Knoll A (2023) Investigation of Inspection and Maintenance Optimization with Deep Reinforcement Learning in Absence of Belief States. In: 14th International Conference on Applications of Statistics and Probability in Civil Engineering, ICASP14
48. Shang Y, Wu W, Liao J, Guo J, Su J, Liu W, Huang Y (2020) Stochastic Maintenance Schedules of Active Distribution Networks Based on Monte-Carlo Tree Search. IEEE Trans Power Syst 35(5):3940–3952
49. Hoffman M, Song E, Brundage MP, Kumara S (2021) Online improvement of condition-based maintenance policy via monte carlo tree search. IEEE Trans Autom Sci Eng 19(3):2540–2551
50. Holmgren V (2019) General-purpose maintenance planning using deep reinforcement learning and Monte Carlo tree search. Linköping University, Sweden
51. Wang Z, Schaul T, Hessel M, Hasselt H, Lanctot M, Freitas N (2016) Dueling Network Architectures for Deep Reinforcement Learning. In: International conference on machine learning. PMLR, pp 1995–2003
52. Morato PG, Papakonstantinou KG, Andriotis CP, Nielsen JS, Rigo P (2022) Optimal inspection and maintenance planning for deteriorating structural components through dynamic Bayesian networks and Markov decision processes. Struct Saf 94:102140
53. Berenguer C, Chu C, Grall A (1997) Inspection and maintenance planning: an application of semi-Markov decision processes. J Intell Manuf 8:467–476
54. Faber MH, Sørensen JD, Tychsen J, Straub D (2005) Field Implementation of RBI for Jacket Structures. J Offshore Mech Arctic Eng 127(3):220–226
55. Ranjith S, Setunge S, Gravina R, Venkatesan S (2013) Deterioration Prediction of Timber Bridge Elements Using the Markov Chain. J Perform Constr Facil 27(3):319–325
56. Noichl F (2019) Sequential decision problems with uncertain observations: Value of Information with erroneous assumptions. Master's thesis, TU München
57. Braziunas D (2003) POMDP solution methods. University of Toronto
58. Dong H, Dong H, Ding Z, Zhang S, Chang (2020) Deep Reinforcement Learning. Springer
59. Cassandra AR, Kaelbling LP, Littman ML (1994) Acting Optimally in Partially Observable Stochastic Domains. AAAI 94:1023–1028
60. Walraven E, Spaan MT (2019) Point-Based Value Iteration for Finite-Horizon POMDPs. J Artif Intell Res 65:307–341
61. Oliehoek FA, Spaan MT, Vlassis N (2008) Optimal and Approximate Q-value Functions for Decentralized POMDPs. J Artif Intell Res 32:289–353
62. Straub D (2009) Stochastic Modeling of Deterioration Processes through Dynamic Bayesian Networks. J Eng Mech 135(10):1089–1099
63. Hauskrecht M (2000) Value-function approximations for partially observable markov decision processes. J Artif Intell Res 13:33–94
64. Brownlee J (2020) Data Preparation for Machine Learning: Data Cleaning, Feature Selection, and Data Transforms in Python. Machine Learning Mastery
65. Hochreiter S, Schmidhuber J (1997) Long Short-Term Memory. Neural Comput 9(8):1735–1780
66. Nielsen MA (2015) Neural Networks and Deep Learning, vol 25. Determination press, San Francisco
67. Bottou L et al (1991) Stochastic Gradient Learning in Neural Networks. Proc Neuro-Nımes 91(8):12
68. Niessner M, Leal-Taixé L (2021) Introduction to Deep Learning. Technical University of Munich, Germany
69. Vodopivec T, Samothrakis S, Ster B (2017) On Monte Carlo Tree Search and Reinforcement Learning. J Artif Intell Res 60:881–936
70. Metropolis N, Ulam S (1949) The Monte Carlo Method. J Am Stat Assoc 44(247):335–341
71. Tarsi M (1983) Optimal Search on Some Game Trees. J ACM (JACM) 30(3):389–396
72. Gibbons R et al (1992) A Primer in Game Theory. Harvester Wheatsheaf, New York
73. Abramson B (2014) The Expected-Outcome Model of Two-Player Games. Morgan Kaufmann, San Mateo

74. Browne CB, Powley E, Whitehouse D, Lucas SM, Cowling PI, Rohlfshagen P, Tavener S, Perez D, Samothrakis S, Colton S (2012) A Survey of Monte Carlo Tree Search Methods. IEEE Trans Comput Intell AI Games 4(1):1–43
75. Kocsis L, Szepesvári C (2006) Bandit based Monte-Carlo Planning. In: European conference on machine learning. Springer, pp 282–293
76. Kingma DP, Ba J (2014) Adam: A method for stochastic optimization. arXiv preprint arXiv:14126980
77. PyTorch (2022) Adam. https://pytorch.org/docs/stable/generated/torch.optim.Adam.html#torch.optim.Adam. Accessed 03 July 2022
78. Reddi SJ, Kale S, Kumar S (2019) On the convergence of adam and beyond. arXiv preprint arXiv:190409237
79. You K, Long M, Wang J, Jordan MI (2019) How does learning rate decay help modern neural networks? arXiv preprint arXiv:190801878
80. Ge R, Kakade SM, Kidambi R, Netrapalli P (2019) The Step Decay Schedule: A Near Optimal, Geometrically Decaying Learning Rate Procedure For Least Squares. Adv Neural Inf Process Syst 32:14977-14988
81. Gelly S, Silver D (2011) Monte-Carlo tree search and rapid action value estimation in computer Go. Artif Intell 175(11):1856–1875
82. Couetoux A (2013) Monte Carlo Tree Search for Continuous and Stochastic Sequential Decision Making Problems. PhD thesis, Université Paris Sud-Paris XI

Publisher's Note