Shielded Learning for Resilience and Performance based on Statistical Model Checking in Simulink

Julius Adelt, Sebastian Bruch, Paula Herber, Mathis Niehage, and Anne Remke

University of Münster, Einsteinstr. 62, 48149 Münster, Germany
`firstname.lastname@uni-muenster.de`

Abstract. Safety, resilience and performance are crucial properties in intelligent hybrid systems, in particular if they are used in critical infrastructures or safety-critical systems. In this paper, we present a case study that illustrates how to construct provably safe and resilient systems that still achieve certain performance levels with a statistical guarantee in the industrially widely used modeling language Simulink. The key ideas of our paper are threefold: First, we show how to model failures and repairs in Simulink. Second, we use hybrid contracts to non-deterministically overapproximate the failure and repair model and to deductively verify safety properties in the presence of worst-case behavior. Third, we show how to learn optimal decisions using statistical model checking (SMC-based learning), which uses the results from deductive verification as a shield to ensure that only safe actions are chosen. We take component failures into account and learn a schedule that is optimized for performance and ensures resilience in a given Simulink model.

Keywords: Hybrid Systems · Resilience · Reinforcement Learning · Formal Verification · Statistical Model Checking

1 Introduction

The demands on the functionality and flexibility of cyber-physical systems are steadily increasing. At the same time, they are increasingly used in critical infrastructures, for example, controlling energy or water supply, and in safety-critical systems such as self-driving cars. Model-driven development frameworks such as MATLAB Simulink help to conquer the complexity and have gained increasing acceptance in industry. Simulink also provides extensions and toolboxes for learning. Learning enables the hybrid systems that are modeled in Simulink to adapt to dynamic changes in the environment, and thus significantly increases their flexibility. To ensure that such intelligent hybrid systems remain operational even in unexpected situations and under external disruptions is a major challenge. Existing approaches for the verification of hybrid systems either focus on the rigorous verification of safety guarantees [6,18,1], or employ probabilistic techniques to optimize the probability that a stochastic hybrid system satisfies a temporal logic formula [38,40,44]. While the former often involves worst-case considerations that impede the performance, the latter does not yield guarantees for all possible behaviors.

To overcome this problem, we have recently proposed a novel approach to combine deductive formal verification and quantitative analysis for intelligent hybrid systems, which takes uncertainties and learning into account [2]. However, our previous approach used a variety of heterogeneous formalisms and tools.

In this paper, we present a case study to construct provably safe and resilient systems that optimize the probability that performance properties are satisfied using shielded SMC-based learning in Simulink. Our work is based on three key ideas: First, we provide a way to model stochastic failure and repair times in Simulink using Simulink random and memory blocks to model variable delays and a decision logic to model component failures and repairs. Second, to enable deductive verification of a given Simulink model with stochastic extensions, we encapsulate the failure and repair model in a dedicated subsystem and use hybrid contracts to define a non-deterministic overapproximation of its behavior. With our previously proposed Simulink2dL [35,3] transformation, we can automatically transform the Simulink model together with the hybrid contract into the differential dynamic logic (d$\mathcal{L}$). We use the interactive theorem prover KeYmaera X [18] to deductively verify safety and resilience properties in the presence of worst-case behavior on the resulting model. Third, we use SMC-based learning within Simulink, which can learn a schedule that is optimized for performance but still ensures resilience in the presence of failures by using the deductive verification results as a shield on the learning component. We take component failures into account using our failure and repair model and learn near-optimal decisions using reinforcement learning.

Compared to our previous work [2], we make the following contributions:

- We model stochastic failure and repair times in Simulink.
- We capture the failure and repair model in a dedicated Simulink subsystem and use hybrid contracts to define a non-deterministic overapproximation.
- We enable SMC-based learning for Simulink and validate its results with a recent extension of the statistical model checker HYPEG.

Our case study is a stochastic extension of an intelligent water distribution system provided by MathWorks [59]. We formally verify safety and resilience on a d$\mathcal{L}$ model that is automatically generated from the Simulink model together with the hybrid contract of the failure and repair model. We combine the deductive verification with SMC-based learning, and we provide a near-optimal scheduler that is guaranteed to be safe and resilient by construction. It maximizes resilience and the probability that the energy consumption is kept below a given limit. We validate the results via the statistical model checker HYPEG on a hybrid Petri net model.

This paper is structured as follows: In Sec. 2 and 3, we introduce preliminaries and discuss related work. In Sec. 4, we combine deductive verification and SMC-based learning for our case study. We present experimental results in Sec. 5 and conclude in Sec. 6.

2 Background

In this section, we introduce reinforcement learning, Simulink, our deductive verification approach for Simulink, and statistical model checking.

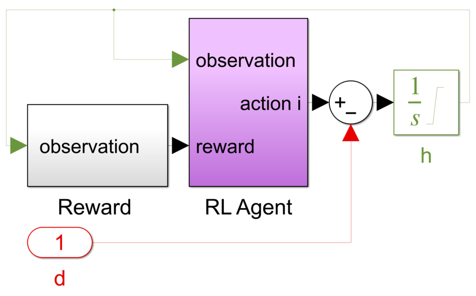
2.1 Reinforcement Learning (RL)

Reinforcement learning is a class of machine learning methods for learning in a trial and error approach by interacting with an environment through actions [55]. The goal of an RL algorithm is to optimize a reward by learning a policy $\pi(a|s)$ that determines which actions to take in which states. The mathematical basis are Markov decision processes (MDPs) [55]. An MDP is a tuple (S, A, R, p), where S is a set of states, A a set of actions, $R \subset \mathbb{R}$ a set of rewards, and p a probability distribution, which describes the MDPs dynamics. In an MDP, an agent and an environment interact in discrete time steps. At each step t, the agent chooses an action $a_t \in A$ to apply in the current state $s_t \in S$. Then the RL agent receives a new state $s_{t+1} \in S$ resulting from the applied action and a reward $r_{t+1} \in R$. The probabilities of states s, actions a and rewards r at times t are given by random variables S_t, A_t and R_t. The expected reward r and next state s' resulting from the application of a in s can be expressed as the probability distribution $p(s', r|s, a) \doteq Pr\{S_t = s',\ R_t = r|S_{t-1} = s,\ A_{t-1} = a\}$.

2.2 Simulink and the RL Toolbox

Simulink [56] is an industrially well established graphical modeling language for hybrid systems. It comes with a tool suite for simulation and automated code generation. Simulink models consist of blocks that are connected by discrete or continuous signals. The Simulink block library provides a large set of predefined blocks, from arithmetics over control flow blocks to integrators and complex transformations. Together with the MATLAB library, linear and non-linear differential equations can be modeled and simulated. Furthermore, the Simulink library provides random blocks to sample values from a random distribution.

Fig. 1a shows a hybrid Simulink model of a water tank controlled by an RL agent. The current water level h is computed by integrating the difference of an inflow i and demand d over time in a (time-continuous) integrator block. The water demand d is provided by an input port. The RL Toolbox [57] provides the *RL Agent* block, which enables the execution of RL algorithms directly within Simulink. In this example, the *RL Agent* controls the inflow i and has to meet the demand while preventing the tank from going empty. The *RL Agent* block acts in fixed sampling steps. In each step, it samples *observations* and *rewards*, and outputs an *action* chosen by its RL algorithm. Here, the observation corresponds to the water level h and the action is the chosen inflow i. The reward is calculated in a user defined *Reward* subsystem.



(a) Simulink Model

$$
\begin{array}{l}
pre \rightarrow [\{ \\
\quad d := *; ?(d_{min} \le d \le d_{max}); \\
\quad \{?(c_{RL} \ge t_s); \\
\qquad c_{RL} := 0; i := *; ?(HC); \\
\quad ++ \\
\quad\; ?(c < t_s); \} \\
\quad \{t' = 1, h' = i - d, \\
\quad c'_{RL} = 1 \& \ (c_{RL} \le t_s)\} \\
\}*]\, post
\end{array}
$$

(b) From Simulink to Differential Dynamic Logic

Fig. 1: Reinforcement Learning, Simulink, and d$\mathcal{L}$

2.3 Differential Dynamic Logic and Simulink2d$\mathcal{L}$

The semantics of Simulink is only informally defined. To enable deductive verification of Simulink models, we have proposed a fully-automatic transformation from Simulink into the differential dynamic logic (d$\mathcal{L}$) [50] in [35]. The d$\mathcal{L}$ is a logic for formally specifying and reasoning about properties of hybrid systems, which are modeled as hybrid programs.

The syntax is as follows: $\alpha; \beta$ models a sequential composition of two hybrid programs α and β. $\alpha \cup \beta$ (or $\alpha \mathrel{++} \beta$) models a non-deterministic choice. A non-deterministic loop α^* executes α zero or more times. The hybrid program $x := e$ evaluates the term e and assigns it to the variable x. $x := *$ denotes a non-deterministic assignment. $?\mathcal{Q}$ is a test, which checks whether the formula $\mathcal{Q}$ is fulfilled. Finally, $\{x'_1 = \theta_1,\, x'_2 = \theta_2,\, x'_n = \theta_n \ \&\ \mathcal{Q}\}$ is a continuous evolution, which evolves a set of variables x with a set of differential equations θ. A continuous evolution may progress as long as the evolution domain $\mathcal{Q}$ is satisfied. d$\mathcal{L}$ provides two modalities for reasoning about reachable states of hybrid programs. $[\alpha]\phi$ states that a formula ϕ holds in every state reachable by α. $\langle\alpha\rangle\phi$ states that there exists a state reachable by α in which ϕ holds. Specifications for hybrid programs are defined as $pre \rightarrow [\alpha] post$.

A d$\mathcal{L}$ specification can be deductively verified with the interactive theorem prover KeYmaera X [18]. Deductive reasoning avoids the state space explosion problem and can also be used for parameterized and infinite-state systems, but requires high expertise, e.g., for the manual definition of invariants.

For the transformation from Simulink to d$\mathcal{L}$, we have defined d$\mathcal{L}$ expressions that precisely capture the semantics of all Simulink blocks in a given model, connect them according to the signal lines, and expand control conditions such that assignments and evaluations are only performed if the control conditions are satisfied [35]. To enable compositional verification, we have introduced the concept of hybrid contracts for Simulink [36] and we have extended this concept to RL components in [3]. A hybrid contract is a tuple $hc = (\phi_{in}, \phi_{out})$ that specifies assumptions on the inputs and guarantees on the outputs of a given Simulink subsystem or RL agent. Contracts replace subsystems and agents during transformation, which enables us to abstract from their inner workings. Simulink subsystems can be individually verified to ensure that they fulfill their contracts. For RL agents we use runtime monitoring to ensure that the contract holds. The d$\mathcal{L}$ model in Figure 1b corresponds to the simple Simulink water tank in Fig. 1a. The input d is modeled by a non-deterministic assignment constrained to the range $[d_{min}, d_{max}]$. The integrator block is captured by a continuous evolution. The water level h evolves with $h' = i - d$, the difference of current inflow i and demand d. The RL agent is captured by a discrete assignment and a continuous evolution. The discrete assignment selects a safe action according to the hybrid contract HC whenever the RL agent's sample time elapses. The clock variable c_{RL} is evolved in the continuous evolution and $c_{RL} \leq t_s$ is added to the evolution domain to ensure that no sampling steps are missed. The global simulation loop is modeled by a nondeterministic repetition.

2.4 Statistical Model Checking and Encoding of Properties

Statistical model checking is used to estimate the probability that a simulation run fulfills a linear-time property Ψ defined in signal temporal logic (STL) [40] over the state of a hybrid model. Whether a property specified in STL at time t is satisfied in a simulation of the stochastic hybrid model can be decided by a model checker which monitors the state evolution. The context-free grammar

$$\Psi ::= tt \mid AP \mid \neg\Psi \mid \Psi \wedge \Psi \mid \Psi\, U^{[t_1,t_2]}\, \Psi$$

constructs a signal temporal logic property Ψ. It consists of true (tt) and continuous atomic properties (AP) comparing a function $f : S \to \mathbb{R}$ with the value zero: $f(\Gamma) > 0$. They can be combined with a logical *and* or with a time-bounded *until*-operator and negated. STL as in [40] only supports continuous signals, i.e. continuous variables in our context. By mapping discrete modes to continuous signals, they can be included as well. We refer to our own previous work [49] for the semantics of STL in the context of hybrid Petri nets with general transitions (HPnG).

In the following, let $X_i \sim \text{Bernoulli}(p)$ be a random variable and let $x_1, ..., x_n$ be realizations of X_i for n simulation runs with $1 \leq i \leq n$, so that $X_i = 1$ if a property Ψ holds for the i-th simulation run. We denote by r the number of runs for which $X_i = 1$ (i.e., Ψ is fulfilled). Thus, the arithmetic mean of $x_1, ..., x_n$ is $\bar{x} = \frac{r}{n}$.

The desired level of confidence denotes as $\lambda \in [0, 1]$, so that the confidence interval covers in $(100 \cdot (1 - \lambda))\%$ of the times the real probability.

In [47], the calculation of different types of confidence intervals (CI) for statistical model checking of HPnGs is discussed. Since we are considering a Bernoulli random variable, we are able to use interval estimation approaches for binomial proportions, as presented by Agresti and Coull in [4].

This paper uses the *Score Confidence interval* [61], which is well suited for small n and Bernoulli random variables. In the following, z_c denotes the c quantile of the standard normal distribution. The Score CI is then determined by

$$\mathrm{CI}_{Score} = \left[\frac{\bar{x} + \frac{z^2_{(1-\lambda/2)}}{2n} - A}{(1 + \frac{z^2_{(1-\lambda/2)}}{n})} ; \frac{\bar{x} + \frac{z^2_{(1-\lambda/2)}}{2n} + A}{(1 + \frac{z^2_{(1-\lambda/2)}}{n})} \right], \tag{1}$$

with

$$A = z_{(1-\frac{\lambda}{2})} \sqrt{\frac{\bar{x}(1-\bar{x}) + \frac{z_{(1-\frac{\lambda}{2})}}{4n}}{n}}, \tag{2}$$

where the lower bound of CI_{Score} is zero for $r = 0$ and the upper bound is one for $r = n$.

3 Related Work

There has been some work on failure and repair modeling in Simulink. For example, in [20,21], the authors propose a failure analysis for Simulink using probabilistic model checking with Prism [32]. However, the proposed Simulink model consists of purely abstract components, annotated with failure probabilities at each subsystem. Similarly, the authors of [41] present a Simulink library that integrates Monte Carlo and fault tree methodologies. While specialized blocks are provided for fault tree modeling (e.g., *and*, *or*, *basic events*), the behavior of the underlying hybrid system is not considered. In [53], the authors propose *ErrorSim*, a tool for simulative error propagation analysis of Simulink models. They provide an external tool that enables the user to annotate fault injection types together with failure probabilities, and then use MATLAB callback functions to perform error simulation. With that, they enable the user to perform fault tree analysis and failure mode and effects analysis. However, failure and repair modeling is not supported directly within Simulink, and temporal logics properties cannot be analyzed. In several works, for example [60,39,29], the performance of a given deterministic Simulink model has been evaluated. However, in these works, the performance evaluation is carried out via a single simulation run for different parameter settings, and not with a statistical analysis or learning. Furthermore, the systems have not been formally verified.

There have been quite some efforts to enable the formal verification of systems that are modeled in Simulink. However, many of them, e.g. [7,28,51], including the Simulink Design Verifier [58], are limited to discrete subsets of Simulink. Formal verification methods that support hybrid systems modeled in Simulink are, e.g., proposed in [13,42,62,12]. However, none of these methods enables a systematic abstraction of failure and repair times, and, to the best of our knowledge, none of them enables the verification of intelligent hybrid Simulink models with RL components.

There also exist several approaches where formal methods are used to ensure the safety of reinforcement learning. In [5], the use of a shield is proposed, which substitutes unsafe for safe actions and is synthesized from a safety automaton and an abstraction of the environment. This idea has attracted quite some interest. A survey on shield synthesis for reinforcement learning is given in [31]. Recent work [11] applies reach-avoid shields to partially observable MDPs. In [19], the safety of an RL controller is ensured via verified runtime monitors based on a differential dynamic logic model, and we have adopted this approach for Simulink in [3,1]. However in all of these approaches, the resulting knowledge about safe actions is not used in quantitative analyses, i.e., obtaining an optimized probability for meeting an additional property while only choosing guaranteedly safe actions is not considered.

There has been a number of works on statistical model checking (SMC) for Simulink. For example, in [63], the authors present an SMC approach based on Bayesian statistics and show that it is feasible for hybrid systems with stochastic transitions, a generalization of Simulink/Stateflow models. To model failures, they randomly introduce faults into a given Simulink model. In [34], the authors propose an extension of the statistical model-checker Plasma Lab [8] for Simulink. They use custom C-code blocks that generate independent sequences of random draws to model failure probabilities and check bounded linear temporal logic properties over sequences of states and time stamps. However, they neither consider failure and repair times nor learning. In [17], the authors propose a transformation from Simulink into stochastic timed automata (STA) and perform statistical model checking with UPPAAL SMC on the resulting network of STA. However, they do not consider random generation blocks and transform a given Simulink model into a deterministic STA model where all probabilities are one, so they can neither take failure probabilities nor repair times into consideration.

Statistical model checking has been proposed for different kinds of (hybrid) stochastic systems. For hybrid Petri nets, statistical model checking has been proposed for linear evolutions in [48,47] and for non-linear evolutions in [45]. While the Modest Toolset's [24] models simulator [9] supports stochastic hybrid models with linear dynamics as well as lightweight scheduler sampling [33] to approximate optimal schedulers, it provides the latter only for non-hybrid models [14,15]. While Simulink is able to capture a wide class of model instances, more restricted and formal models like MDPs have been considered to improve performance and safety with reinforcement learning (e.g. [26]), as well as to deal with *unknown* stochastic behavior [25,10], and with linear-time logic specifications (e.g. [10,52,23]).

Learning for stochastic hybrid systems is considered, e.g. in a variant of ProbReach, which applies the cross-entropy method for resolving nondeterminism [54]. Also [16] enriches an SMT-based approach with decision trees and AI planning algorithms to handle nondeterminism. Using reinforcement learning in Stochastic Hybrid Models to resolve nondeterminism in an optimal way has been proposed in [44] for Discrete Event Systems (DES) and applied to optimize (dis-)charging of a smart home in [46]. To handle the underlying uncountable state space, a discretization of the continuous variables is used and the optimality of the approach is proven for a decreasing discretization distance.

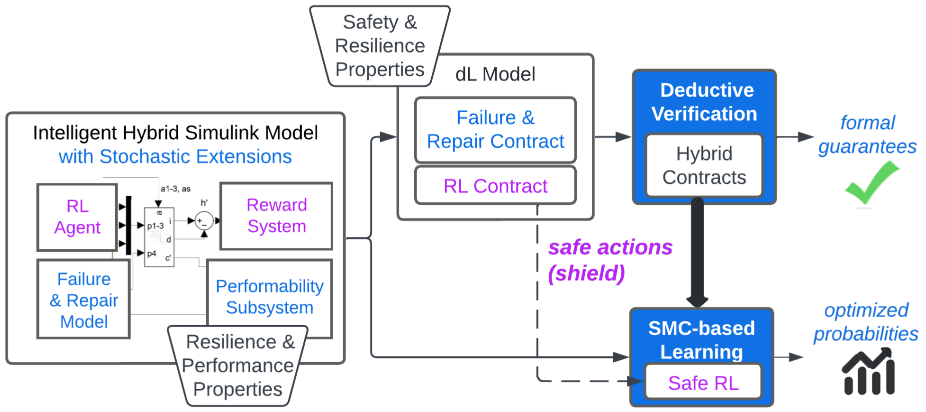


Fig. 2: Deductive Verification and Shielded SMC-based Learning for Simulink

Finally, there has been some work on combining rigorous formal and statistical methods. In [30], the authors incorporate statistical hypothesis testing to compute promising configurations of program verifiers automatically. However, they do not support hybrid systems, and they do not consider both safety and performance properties. In [22], the authors present a formal framework for an integrated qualitative and quantitative model-based safety analysis. This approach is more closely related to our work and also exploits the idea to combine the best out of both worlds from formal verification and quantitative analysis. However, they again do not support hybrid systems, do not consider deductive verification methods and also not the integration of learning components.

4 Shielded SMC-based Learning in Simulink

In this paper, we combine deductive formal verification with SMC-based learning to ensure safety and resilience of intelligent hybrid systems that are modeled in Simulink. In this section, we first apply shielded SMC-based learning in Section 4.1, before Section 4.2 recaps the intelligent water distribution system [59] also used in [2]. We present our stochastic extension of the given Simulink model with a failure and repair model in Section 4.3. Our approach for a non-deterministic over-approximation of the failure and repair model and its integration into our Simulink to dL transformation is presented in Section 4.4. Finally, we present our approach to extend the Simulink model with a performance analysis subsystem and to use this for SMC-based learning that optimizes a given STL property in Section 4.5.

4.1 Approach

Our overall approach is shown in Figure 2. Our goal is to to construct provably safe and resilient systems that still achieve certain performance levels in

Simulink. To achieve this, we propose to first enrich a given intelligent hybrid Simulink model, which may include an RL agent together with a reward subsystem for learning, with a stochastic failure and repair model. To provide this stochastic extension of a given Simulink model, we use Simulink random blocks to sample from stochastic distributions, for example, the time to the next failure of a component or its repair time.

To provide formal guarantees for the safe and resilient behavior of the given intelligent hybrid system with the stochastic extension, we extend our previously proposed transformation from Simulink to d$\mathcal{L}$ [35,37] with a failure and repair contract, as shown in the upper part of Figure 2. The failure and repair contract provides a non-deterministic over-approximation of the possible failures and thus enables us to verify that the system meets critical safety or resilience properties under worst-case considerations. As proposed in [3], we also use a hybrid contract to abstract from the RL agent. Then, we use the formal d$\mathcal{L}$ representation together with desired safety and resilience properties as an input to the interactive theorem prover KeYmaera X. Using KeYmaera X, we deductively verify that the overall system satisfies the given properties for an unbounded time and potentially even for a system model with unbounded parameters. The verified safety and resilience properties are formally guaranteed to hold under two assumptions: First, we assume that the failure and repair contract actually provides a safe over-approximation of the failures and repairs at runtime. Second, we assume that the RL agent adheres to the hybrid contract at runtime. To enforce the latter, we generate runtime monitors, which can be used to ensure that the RL agent may only choose safe actions. To combine our deductive verification approach with SMC-based learning, we use the safe actions defined by the generated runtime monitor as a shield for the RL agent in the second part of our approach shown in the lower part of Figure 2.

The aim of SMC-based learning is to provide a near-optimal scheduler, which maximizes or minimizes the probability that given resilience or performance properties are satisfied. To enable SMC-based learning directly on a given intelligent hybrid Simulink model, we extend it with a performability subsystem. The performability subsystem encodes the resilience and performance properties, which are given as STL formulas, in Simulink. This enables us to use the formally defined properties for reward shaping, and to combine reinforcement learning and SMC to compute a policy that optimizes the probability that the given STL formula is satisfied as proposed in [44]. As proposed in [2], we restrict the RL agent to safe actions to ensure that the resulting policy is correct by construction with respect to the (deductively) verified safety properties. With that, we compute an optimized probability that quantitative resilience properties (e.g. provide full service whenever possible) or performance properties (e.g. the energy consumption of the pump is never above a certain limit) are satisfied, while we still ensure that critical safety properties (e.g. the tank never runs empty) and qualitative resilience properties (e.g. always provide at least degraded service) are never violated.

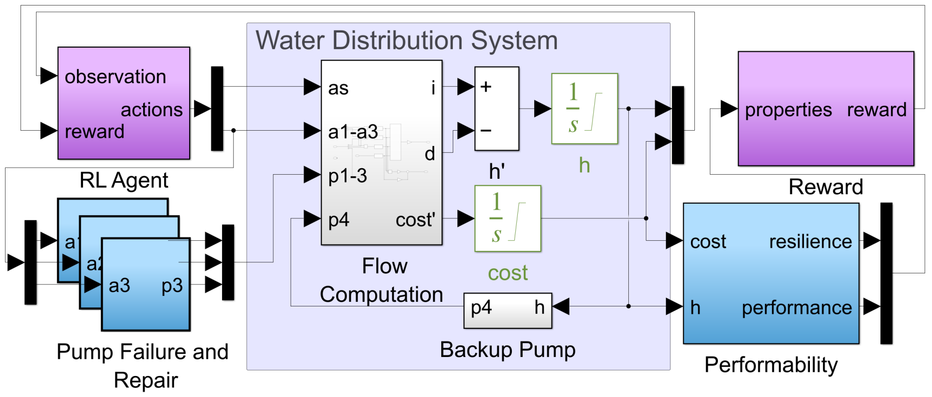


Fig. 3: Intelligent WDS inspired by [59]

4.2 Intelligent Water Distribution System in Simulink

The upper part of Figure 3 depicts the Simulink model of our intelligent water distribution system (WDS) inspired by [59]. The model uses an *RL Agent* to satisfy a water demand by consumers as best as possible, while being energy efficient. The system consists of three pumps, namely $(p_1, p_2, p_3), p_i \in \{0, 1\}$, which are used to constantly pump fluid from a reservoir into a water tank. The *RL Agent* receives *observations* of the current state and a *reward* signal. It decides on the pump activations $(a_1, a_2, a_3), a_i \in \{0, 1\}$ and a chosen maximum supply $a_s \in \mathbb{R}^+$, in discrete sampling steps. The stepsize is defined by a parameter $t_S \in \mathbb{R}^+$. The current inflow and demand are calculated by a *Flow Computation* subsystem. The inflow $i \in \mathbb{R}^+$ is determined by the number of pumps that are currently running. The demand $d \in \mathbb{R}^+$ is limited by the maximum possible supply a_s. The difference of inflow i and demand d is continuously integrated to calculate the current water height $h \in \mathbb{R}^+$. Pumps that are running require energy and thus are associated with cost. The energy consumption $cost' \in \mathbb{R}^+$ is integrated over time to keep track of the total energy consumption $cost \in \mathbb{R}^+$. The *Pump Failure and Repair* and *Performability* subsystems are extensions we use for failure and repair modeling and SMC-based learning and are described in the following subsections.

The major safety requirements of the intelligent WDS is that the water tank should never run empty. To ensure that a minimum supply of water is always available, the system features a backup pump (p_4), which turns on as soon as the water level falls below a backup level h_b. For resilience, the system should always provide at least a degraded service, and full service whenever possible. Note that this definition of resilience comprises a qualitative part (always at least degraded service) and a quantitative part (full service whenever possible). As a performance requirement, the energy consumption should never exceed a given maximum.

4.3 Stochastic Extensions for Modeling Pump Failures and Repairs

To systematically model stochastic pump failures as well as repair times, we extended our Simulink model from [2] with a failure and repair model for pumps. Note that the failure and repair probabilities in [2] only apply to the hybrid Petri net model used in HYPEG and are not explicitly modeled in Simulink. Fig. 4 shows the contents of a *Pump Failure and Repair* subsystem. The core of this system is a Simulink *If Block*, which controls the execution of two subsystems. If the pump is currently not broken ($p_1 == 1$) and turned on by the agent ($a_1 == 1$) the *Pump Running* Subsystem is executed. If the pump is currently broken ($p_1 == 0$), the *Pump Broken* subsystem is executed. The output of both systems determines the current state of the pump. However, only one system can be activated at once. The merge block combines the output of both conditional systems into a single continuous signal. If none of the systems are currently executed, the previous signal is held.

The *Pump Running* system is shown in Fig. 5. The system models the behavior of a currently running pump, subject to stochastic failures. The state of the pump is forwarded to the output. Simulink provides blocks for sampling from different random distributions like gaussian and uniform distributions. Additional random distributions can be implemented using MATLAB functions. We use these blocks to sample delays for failures and repairs. The subsystem *Failure Delay Sampler* in *Pump Running* uses a random number block which samples from a Gaussian distributed random signal with configurable mean, variance and random seed. A new seed is randomly set in each simulation run. The sampled delay is forwarded through an *Abs* block to model a folded normal distribution.

The sampled delay is stored in the *Variable Transport Delay* block. Initially, the state of the variable transport delay and thus the state of the output is 1 (the pump is working). As soon as the *Failure Delay* has elapsed, the variable transport delay block outputs the value 0 and the pump state (p_1) switches to broken. Note that the delay time only elapses if the subsystem is enabled, i.e. if the pump is working and activated by the *RL Agent*. In the overall *Pump Failure and Repair* subsystem (Fig. 4), this leads to a change in the *If-Else* blocks conditions. Thus, *Pump Running* is disabled and *Pump Broken* is enabled. The *Pump Broken* subsystem works analogously to the *Pump Running* subsystem. It outputs $p1 == 0$ until its repair time is reached. Then p_1 is set to 1, *Pump Broken* is disabled and Pump Running is enabled again. Contrary to the *Pump Running* subsystem, the random delay is sampled from a uniform distribution with configurable min and max values and random seed. Note that *Pump Broken* and *Pump Running* reset to their initial state as soon as their respective delay time has elapsed. Each time a switch in pump state and thus between the two subsystems occurs, the now enabled subsystem is reset to its initial state and a new random number for the respective failure or repair delay is drawn.

4.4 Formal Verification of Safety and Resilience Contracts

To ensure safety and the qualitative part of resilience of the water distribution system, we apply our approach proposed in [3,37]. To this end, we transform

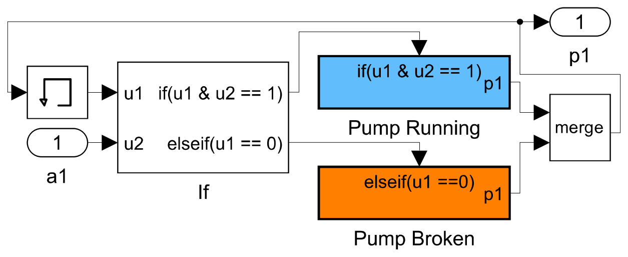


Fig. 4: Failure Repair Subsystem of a Pump

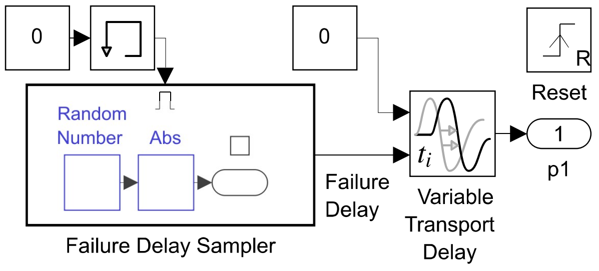


Fig. 5: Pump Running Subsystem

the Simulink model into d$\mathcal{L}$ [35]. Then, we capture the worst case behavior of the failure and repair model as well as the safe behavior of the RL agent with hybrid contracts in d$\mathcal{L}$, and verify the model under the assumption that the failure and repair contract provides a safe overapproximation and that the RL agent complies with its contract [3] deductively in KeYmaera X. This ensures a safe and resilient design by construction. For brevity, we omit the full d$\mathcal{L}$ model, the interested reader is referred to [2]. Our Simulink2dL transformation tool and all proofs can be found online[1].

The contracts for the RL agent, which we have also presented in [2], are shown in Table 1. The safety contract ensures that the maximum water supply is always limited to a value (a_s) that can be satisfied by the current water level (h) without falling below h_{min} until the next decision is made in one sampling step (t_S). In the resilience contract, the first guarantee ensures that the agent always supplies the full service level $a_s = s_{full}$ whenever the water level is above $h_{full} = s_{full} \cdot t_S + h_{min}$. The second guarantee specifies that the agent supplies the degraded service level if the water level is too low, i.e., it chooses $a_s = s_{deg}$ if $h \leq h_{deg} = s_{deg} \cdot t_S + h_{min}$. Note that these definitions imply that $h_{min} \leq h_{deg} \leq h_{full}$. This degraded service is guaranteed by the backup pump, which turns on before the tank runs empty. The third guarantee specifies that the agent supplies intermediate but safe levels between the two boundaries.

[1] https://www.uni-muenster.de/EmbSys/research/Simulink2dL.html

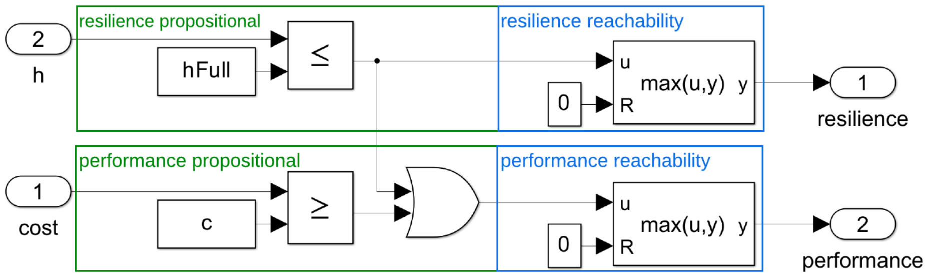


Fig. 6: Performability Subsystem

In contrast to [2], we take the failure and repair subsystems into account. To achieve this, we define a contract that overapproximates the worst-case behavior of the failure and repair model as shown in Table 2. The failure and repair contract provides a safe overapproximation of the possible outputs of the failure repair system by assuming that all pumps can fail or be repaired at any point in time.

We have verified the same properties as in [2], now under the assumption that both the failure and repair contract and the RL agent contract hold, namely that the water tank never runs dry, that the system always offers at least degraded service, and that full service is provided whenever sufficient water is available and whenever the pumps are available for a sufficiently long time. By using the hybrid contracts as a shield for SMC-based learning, we ensure that the safety and resilience guarantees we have proven with KeYmaera X are guaranteed for the optimized model by construction. Note that our formal d$\mathcal{L}$ model uses symbolic constants for system parameters like sampling time t_S, service levels s_{full}, s_{deg} and minimum water height h_{min}. This means that the properties are proven for a range of safe system parameters and for every possible input scenario.

4.5 Performability Subsystem and SMC-based learning

To optimize the quantitative part of resilience and performance, we define a *Performability* subsystem, which implements a monitor that checks the validity of predefined properties during every simulation run. This enables us to perform SMC-based learning by distributing a reward based on the satisfaction of the property during the training process and computing confidence intervals of the probability that the property holds during statistical model checking. For the purpose of this paper, we consider resilience and performance properties similar to [2] as follows: For resilience, we aim to minimize the probability that a water level is reached that is not sufficient for full service ($h \leq h_{full} \in \mathbb{R}^+$) within the maximum simulation time $t_{max} \in \mathbb{R}^+$. This corresponds to the STL formula Φ_r in Table 3.

Performance in terms of energy consumption is expressed as $\Psi = (h \leq h_{full}) \vee (cost > c)$, where $c \in \mathbb{R}^+$ forms an upper bound for the incurred energy cost.

Table 1: RL Agent Contracts

Safety Contract		Resilience Contract	
Assumption	Guarantee	Assumption	Guarantee
true	$h - a_s \cdot t_S > h_{min}$	$h > h_{full}$	$a_s = s_{full}$
		$h \leq h_{deg}$	$a_s = s_{deg}$
		$h > h_{deg} \wedge h \leq h_{full}$	$h - a_s \cdot t_S > h_{min}$

Table 2: Failure and Repair Contract

Assumption	Guarantee
true	$(p_1 = 0 \vee p_1 = 1) \wedge (p_2 = 0 \vee p_2 = 1) \wedge (p_3 = 0 \vee p_3 = 1)$

We minimize the probability that Ψ holds, which intuitively means that states are avoided, where either no full service is delivered or the maximum energy consumption is exceeded. Together with the reachability, the corresponding STL formula is Φ_p in Table 3.

The *Performability* subsystem of the Simulink model is shown in Figure 6. The subsystem receives information relevant for the resilience and performance properties as input. In our model the information received are the current water level h and the accumulated energy *cost* in the current run. The Performability subsystem consists of the following groups of blocks: The left two blocks (depicted in green) represent the propositional part of the resilience and performance properties. The *max* blocks (blue) in the right part of the model then encode the reachability for the respective properties. The *max* blocks take the maximum of the boolean input signals over time and hence, keep returning true, as soon as the desired propositional property is reached once. The reachability values are used in the *Reward* subsystem (cf. Figure 3) for training the agent, i.e. we distribute a reward based on the satisfaction of the property. The reachability values are written back to the MATLAB workspace for statistical model checking after training.

Summarizing, the safety and resilience contracts, which are used for shielding, ensure that the water level stays safe and that full service is always provided if the water level is actually sufficient. The SMC-based learning optimizes the probabilities that a sufficient amount of water for full service is available and that the energy consumption stays below a given maximum.

5 Evaluation of results and validation

This section presents and discusses the results obtained by SMC-based learning, when using safety and resilience contracts from Table 1 as a shield. Previous work [2] has shown the efficiency of using safety contracts as a shield when simulating optimal system properties. For validation the statistical simulation

Table 3: Minimizing STL properties for resilience and performance similar to [2].

resilience	$\Phi_r = tt\ U^{[0.0,t_{max}]}(h \leq h_{full})$
performance	$\Phi_p = tt\ U^{[0.0,t_{max}]}(h \leq h_{full} \vee cost > c)$

Table 4: Fail and repair distributions for the three pumps.

pump	fail: folded normal		repair: uniform	
	μ	σ	a	b
p_1	30	6	7	10
p_2	20	4	3	5
p_3	5	1	1	2

tool HYPEG [47] is used with its recent extension to Q-learning, called HYPEG ML [44] which simulates a hybrid Petri-net model of the water tank.

Section 5.1 provides parameters for the model proposed in Section 4 and the settings for SMC-based learning in Section 5.2. Quantitative results for *resilience* and *performance* are presented in Section 5.3 and Section 5.4, respectively.

5.1 Model Parameters

The intelligent water distribution system modeled in Simulink, as illustrated in Figure 3, initially has a water level of 5.5 m and is filled by each active pump with 0.15 m per hour. Each pump consumes energy when active which results in cost of 0.1 for p_1, p_2 and p_3 and cost of 0.15 for p_4. The RL agent can choose from the action set $\mathcal{A} = \{(a_1, a_2, a_3) \mid a_i \in \{0, 1\}\}$, where a_i describes the activation status of pump p_i. Note that $a_i = 1$ does not necessarily corresponds to an active pump, as the pump might be broken. The backup pump is turned on, as soon as the water level falls below 2.1 m and can not fail. The failure and repair times of the three normal pumps follow the probability distributions indicated in Table 4.

While the pump activation can be chosen by the RL agent at the beginning of every simulation run and after every sampling time $t_S = 10$ from the action set $\mathcal{A}$, the service level is determined by the safety and resilience shield (cf. Table 1). We consider the following service levels $\mathcal{S} = \{s_{full}, s_{mid}, s_{deg}\}$ with $s_{full} = 0.3, s_{mid} = 0.2$, and $s_{deg} = 0.15$. Note that setting s_{mid} to a fixed value, represents a simplification of the model, such that not necessarily the maximum possible demand a_s is chosen by the RL agent in case $h_{deg} < h \leq h_{full}$.

We set the remaining parameters of the safety and resilience contracts as follows: the minimum level to $h_{min} = 2$ m in the safety contract and set $h_{full} =$ 5 m and $h_{deg} = 4$ m in the resilience contract. The maximum cost c accepted is given by $0.225 \cdot t_{max}$, i.e. in average slightly more than 2 pumps working.

5.2 Settings for Simulink and SMC-based learning

We run Simulink with a variable-step solver and a maximum step size of 0.06 in combination with an explicit Runge-Kutta method for solving differential equations (ode45). We use the parallel version of Simulink, however note that Q-learning does not support parallelization. For Deep Q-Learning, we use a default MATLAB Deep Q-Network agent (DQN agent), which uses a neural network which consists of six layers: an input layer for observations, two fully connected layers each followed by ReLU activation layer, and a fully connected output layer. The neural network is a multi-output critic, featuring individual outputs corresponding to each possible discrete action. Each output provides the Q-value for taking the corresponding action based on the current observation. Contrary to the default settings, we set the number of nodes in the hidden layers to 512 instead of 256. To improve the convergence of learning, Simulink uses a target critic and memory replay as introduced in [43]. We apply default smoothing, with a *TargetSmoothFactor* of 10^{-3} and set the *ExperienceBufferLength* to 100 000 and *MiniBatchSize* to 256.

HYPEG ML[2] uses Q-learning, as proposed in [44] on the discretized state-space of the hybrid Petri-net modeling the intelligent water distribution system with stochastic pump failures and repairs, as presented in [2]. SMC-based learning, as proposed in [44], evaluates every simulation run w.r.t. a STL property and during the training period provides a single reward, i.e. either 1 or -1, at the end of a simulation run, i.e. t_{max}. HYPEG ML uses a fixed number of 3 million training runs and finishes every run before evaluating the validity of the STL property. In contrast, Simulink finishes the training period as soon as the property is fulfilled in the last 100 training runs. Due to the duration of simulation runs in Simulink, we have set the maximum number of training runs to 5000. We compare the results obtained via two learning methods, i.e., Q-learning and Deep Q-learning.

For both methods in Simulink two different reward distributions are applied: (i) one reward is given based on the satisfaction of the STL property at the end of a simulation run, (ii) at each sample time t_S a reward is given if the property is not violated yet. In both cases, if the property is violated, the current run is aborted and the agent receives a negative reward.

The confidence intervals in both tools are computed after 5000 simulation runs with confidence level $\lambda = 0.95$.

If Q-learning is used, the observation space must be discretized by both, Simulink and HYPEG ML. The continuous variables are discretized to the first decimal place in HYPEG ML and Simulink. The observation space in Simulink with Q-learning had to be restricted to the water level h and the accumulated $cost$, as training in Simulink is not efficient enough to identify optimal results for a large observation space. In contrast, HYPEG ML uses the full state-space as observation space. Hence, it keeps track of the time that has elapsed since a pump has switched state. The Deep Q-learning agent in Simulink works directly

[2] https://zivgitlab.uni-muenster.de/ag-sks/tools/hypeg

Table 5: Estimated confidence intervals using Wilson score CI for Φ_r, for learned decisions of the action set $\mathcal{A}$, different time bounds and 5000 simulation runs.

		t_{max}	24 h	48 h	72 h	96 h
HYPEG ML	Q-learn one	midpoint	0.0004	0.003	0.002	0.002
		CI	[0.000,0.0008]	[0.001,0.004]	[0.0006,0.003]	[0.0007,0.003]
		train time	1156.7 s	1861.5 s	3235.6 s	3669.3 s
		train. runs	3 000 000	3 000 000	3 000 000	3 000 000
		sim. time	2.4 s	3.1 s	5.4 s	6.4 s
Simulink	Q-learn one	midpoint	0.0088	0.0495	0.1027	0.3435
		CI	[0.006,0.0113]	[0.044,0.0555]	[0.094,0.1111]	[0.330,0.3567]
		train. time	478.6 s	2645.3 s	3642.7 s	3906.1 s
		train. runs	1179	5000	5000	5000
		sim. time	907.3 s	1062.2 s	1508.2 s	1718.3 s
	Q-learn mult	midpoint	0.0194	0.0436	0.0539	0.0859
		CI	[0.016,0.0232]	[0.038,0.0492]	[0.048,0.0602]	[0.078,0.0937]
		train. time	402.9 s	2985.4 s	3697.5 s	3745.8 s
		train. runs	1152	5000	5000	5000
		sim. time	740.5 s	1128.3 s	1285.2 s	1498.9 s
	DQL one	midpoint	0.0050	0.0008	0.0004	0.0004
		CI	[0.003,0.0069]	[0.000,0.0015]	[0.000,0.0008]	[0.000,0.0008]
		train. time	881.0 s	648.2 s	396.4 s	676.6 s
		train. runs	2919	2092	1789	1962
		sim. time	463.1 s	456.7 s	558.8 s	461.2 s
	DQL mult	midpoint	0.0004	0.0004	0.0268	0.0012
		CI	[0.000,0.0008]	[0.000,0.0008]	[0.022,0.0312]	[0.0003,0.002]
		train. time	347.9 s	310.0 s	1672.1 s	1477.5 s
		train. runs	2565	2080	5000	4012
		sim. time	450.1 s	452.7 s	480.3 s	482.8 s

on the continuous state space. Its observations consists of the current pump status, the times since a pump has last switched state, the current water level and the accumulated cost.

5.3 Optimizing Resilience

Similar to [2], resilience is formalized in terms of the STL property Φ_r, as shown in Table 3. We use SMC-based learning to optimize the probability that Φ_r holds, using the runtime monitoring subsystem, explained in Section 4.5 to ensure that the safety and resilience contracts hold. We compare results obtained from the statistical model checker HYPEG ML with results obtained from Simulink.

All results are summarized in Table 5. It can be seen that the Simulink results obtained with Q-learning with either *one reward* per simulation run or with *multiple rewards* differ considerably w.r.t. results obtained with Deep Q-learning in Simulink. Deep Q-learning is able to achieve considerably smaller probabilities, which is desirable, as we defined resilience as property $\varPhi_r$ to be minimized. We believe that the number of training runs is too small and the discretization in Simulink is too coarse to explore the state-space sufficiently with Q-learning. For an infinite number of training runs and a converging discretization, Q-learning is guaranteed to compute optimal probabilities.

Note that HYPEG ML is able to perform considerably more training runs in reasonable time and has a larger observation space, which results in lower midpoints compared to Q-learning in Simulink. Applying Q-learning within Simulink suffers from the reduced state-space and the long training and simulation times per run. Hence, Q-learning within Simulink is not able to efficiently compute (near-)optimal probabilities as computed by HYPEG ML in this setting. We assume that the performance difference between both simulators stems from the fact that HYPEG ML applies discrete-event simulation, which is especially fast when used on piece-wise constant differential equations. In contrast, Simulink optimizes simulation accuracy in complex dynamic systems with a potentially high computational overhead. Note that the maximum step size in Simulink is chosen relatively small to ensure consistent results in the validation. Increasing the step size will improve performance, however our experiments have shown that the magnitude of feasible training runs does not change.

While Deep Q-learning can learn directly on the continuous observation space, it lacks convergence guarantees and the robustness of the approach, as well as the reproducibility of the results are subject to many factors, e.g., the neural network architecture and the choice of hyper-parameters [27].

The result obtained by HYPEG ML with Q-learning for $t_{max} = 24\,\mathrm{h}$ matches the confidence intervals computed by Deep Q-learning for multiple rewards. For larger t_{max} Simulink with Deep Q-learning and one reward computes confidence intervals which overlap with the confidence intervals computed by HYPEG ML. However note that Deep Q-learning achieves smaller midpoints for $t_{max} > 24\,\mathrm{h}$.

While Q-learning in Simulink achieves lower midpoints with multiple rewards, the impact of the reward structure with Deep Q-learning is unclear.

Note that the simulation times in HYPEG ML are much smaller than all simulation times in Simulink. The fact that HYPEG ML is able to simulate individual runs much faster than Simulink also translates to the training time per run. This makes it difficult to compare training times between HYPEG ML and Simulink. The chosen termination condition of 5000 runs in Simulink yields comparable training times to HYPEG ML with 3 million runs, however with larger midpoints. Only in case $t_{max} = 24\,\mathrm{h}$ Simulink with Q-learning terminates before 5000 runs and its training is about twice as fast as HYPEG ML.

Deep Q-learning takes considerably less training runs than Q-learning in Simulink. The number of training runs varies for all t_{max}, which in turn leads to varying training times. Only the case with multiple rewards for $t_{max} = 72\,\mathrm{h}$ uses 5000 training runs and computes a large midpoint. This is attributed to the lack of robustness of Deep Q-learning.

Table 6: Estimated confidence intervals using Wilson score CI for Φ_p for learned decisions of the action set $\mathcal{A}$.

		t_{max}	24 h	48 h	72 h	96 h
HYPEG ML	Q-learn one	midpoint	0.001	0.001	0.002	0.002
		CI	[0.000,0.001]	[0.000,0.001]	[0.001,0.003]	[0.001,0.003]
		train time	1017.9 s	1933.4 s	2949.2 s	3775.8 s
		train. runs	3 000 000	3 000 000	3 000 000	3 000 000
		sim. time	1.9 s	3.1 s	5.3 s	6.6 s
Simulink	DQL one	midpoint	0.028	0.014	0.005	0.001
		CI	[0.024,0.033]	[0.011,0.017]	[0.003,0.007]	[0.000,0.001]
		train. time	1061.0 s	1541.9 s	1716.0 s	693.3 s
		train. runs	3499	4775	5000	1979
		sim. time	550.3 s	475.7 s	561.4 s	505.1 s
	DQL mult	midpoint	0.081	0.024	0.032	0.022
		CI	[0.073,0.088]	[0.020,0.028]	[0.027,0.037]	[0.018,0.026]
		train. time	734.2 s	1620.3 s	1753.4 s	1906.3 s
		train. runs	2442	5000	5000	5000
		sim. time	500.5 s	467.9 s	480.2 s	491.9 s

5.4 Optimizing Performance

The second property of interest is given by the STL property Φ_p (cf. Table 3), i.e. limiting the energy cost and still providing the full service level. Note that our recent work [2] optimized performance for $0.2 \cdot t_{max}$, however Simulink was not able to optimize this property, as the value of 0.2 does not allow for any error in the taken decisions. Relaxing the problem to $0.225 \cdot t_{max}$ instead allows for taking some non-optimal decisions.

Results obtained from Simulink and HYPEG ML are summarized in Table 6. We recomputed optimal performance with HYPEG ML for the adapted problem and compare with results obtained by Simulink for Deep Q-learning with a single reward and with multiple rewards per simulation run. We did not consider Q-learning for performance in Simulink, as Deep Q-learning performed much better when optimizing resilience.

It can be seen, that for larger time-bounds $t_{max} = 72$ h and $t_{max} = 96$ h the computed confidence intervals of HYPEG ML and Simulink, in case one reward is distributed, overlap. For the smaller time-bound, none of the computed confidence intervals overlap. Since we are able to perform 3 million training runs in HYPEG ML in reasonable time, the computed midpoints in HYPEG ML are almost always smaller than those computed by Simulink with at most 5000 training runs. Due to the convergence guarantees available for Q-learning in combination with a large number of training runs we suspect, that the lower midpoints actually indicate better results and are not caused by a statistical error. Hence, we assume that the quality of the Simulink results suffers from the relatively long training times, which have in turn lead to our restriction to 5000 runs. Since no convergence guarantees exist for Deep Q-learning, it cannot be concluded that a larger maximum of training runs automatically leads to smaller midpoints.

Observe that performance results obtained with one reward per run are smaller than those obtained with multiple rewards. We suspect that for optimizing performance, multiple rewards stimulate cost-expensive behavior in the beginning which benefits resilience, however incurs higher cost towards the end.

6 Conclusion

Building on recent advances in deductive verification and SMC-based learning, we present a unified approach to optimize resilience and performance in Simulink models while ensuring safety and safety-relevant resilience properties via contracts. We use a transformation from Simulink to dL and deductive verification as proposed in [35,3] to ensure that the overall system is safe and resilient if learning components adhere to their contracts. Then, we use these contracts as a shield which restrict the action space of the learning method to safe actions. To do this, we have extended an existing Simulink model with stochastic failure and repair components and a performability subsystem, and have used this for SMC-based learning, as proposed in [44].

We apply (Deep) Q-learning in Simulink with either a single reward at the end of the simulation run, or with multiple rewards, i.e., one after each sampling time. Results are validated with those obtained via a recent extension of the statistical model checker HYPEG for a similar model, as presented in [2].

The evaluation of the results shows that in this case study Q-learning is not able to optimize resilience on the considered Simulink observation space. We believe that the required number of training runs would yield excessive training times. While Deep Q-learning performs much better and is able to optimize resilience and performance efficiently, it requires a larger effort for identifying a suitable neural network architecture. Note that an efficient SMC-based learning tool like HYPEG ML helped to identify the neural network applied.

In future work, we plan to apply our approach to other case studies and to further investigate the scalability of both the deductive verification and the SMC-based learning. Furthermore, we plan to investigate the effect of more fine-granular service levels on the learning results.

References

1. Adelt, J., Brettschneider, D., Herber, P.: Reusable contracts for safe integration of reinforcement learning in hybrid systems. In: Automated Technology for Verification and Analysis: 20th International Symposium, ATVA 2022, Virtual Event, October 25–28, 2022, Proceedings. pp. 58–74. Springer (2022)
2. Adelt, J., Herber, P., Niehage, M., Remke, A.: Towards safe and resilient hybrid systems in the presence of learning and uncertainty. In: Leveraging Applications of Formal Methods, Verification and Validation. Verification Principles: 11th International Symposium, ISoLA 2022, Rhodes, Greece, October 22–30, 2022, Proceedings, Part I. pp. 299–319. Springer (2022)

3. Adelt, J., Liebrenz, T., Herber, P.: Formal Verification of Intelligent Hybrid Systems that are modeled with Simulink and the Reinforcement Learning Toolbox. In: Formal Methods 2021. LNCS, vol. 13047. Springer (2021)
4. Agresti, A., Coull, B.: Approximate is better than "exact" for interval estimation of binomial proportions. The American Statistician **52**, 119–126 (1998)
5. Alshiekh, M., Bloem, R., Ehlers, R., Könighofer, B., Niekum, S., Topcu, U.: Safe Reinforcement Learning via Shielding. Proceedings of the AAAI Conference on Artificial Intelligence **32** (2018)
6. Alur, R.: Formal verification of hybrid systems. In: ACM Int. Conference on Embedded Software (EMSOFT). pp. 273–278 (2011)
7. Araiza-Illan, D., Eder, K., Richards, A.: Formal verification of control systems' properties with theorem proving. In: UKACC Int. Conference on Control (CONTROL). pp. 244–249. IEEE (2014)
8. Boyer, B., Corre, K., Legay, A., Sedwards, S.: Plasma-lab: A flexible, distributable statistical model checking library. In: Quantitative Evaluation of Systems: 10th International Conference, QEST 2013, Buenos Aires, Argentina, August 27-30, 2013. Proceedings 10. pp. 160–164. Springer (2013)
9. Budde, C.E., D'Argenio, P.R., Hartmanns, A., Sedwards, S.: An efficient statistical model checker for nondeterminism and rare events. Int. J. Softw. Tools Technol. Transf. **22**(6), 759–780 (2020)
10. Cai, M., Peng, H., Li, Z., Kan, Z.: Learning-Based Probabilistic LTL Motion Planning With Environment and Motion Uncertainties. IEEE Transactions on Automatic Control **66**(5), 2386–2392 (2021)
11. Carr, S., Jansen, N., Junges, S., Topcu, U.: Safe reinforcement learning via shielding under partial observability. Proceedings of the AAAI Conference on Artificial Intelligence **37**(12), 14748–14756 (2023)
12. Chen, M., Han, X., Tang, T., Wang, S., Yang, M., Zhan, N., Zhao, H., Zou, L.: MARS: A toolchain for modelling, analysis and verification of hybrid systems. In: Provably Correct Systems, pp. 39–58. Springer (2017)
13. Chutinan, A., Krogh, B.H.: Computational techniques for hybrid system verification. In: IEEE Trans. on Automatic Control. vol. 48(1), pp. 64–75. IEEE (2003)
14. D'Argenio, P., Legay, A., Sedwards, S., Traonouez, L.M.: Smart sampling for lightweight verification of Markov decision processes. International Journal on Software Tools for Technology Transfer **17**(4), 469–484 (Aug 2015)
15. D'Argenio, P.R., Hartmanns, A., Sedwards, S.: Lightweight Statistical Model Checking in Nondeterministic Continuous Time. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation. Verification, vol. 11245, pp. 336–353. Springer, Cham (2018), lNCS
16. Ellen, C., Gerwinn, S., Fränzle, M.: Statistical model checking for stochastic hybrid systems involving nondeterminism over continuous domains. Int. Journal on Software Tools for Technology Transfer **17**(4), 485–504 (2015)
17. Filipovikj, P., Mahmud, N., Marinescu, R., Rodriguez-Navas, G., Seceleanu, C., Ljungkrantz, O., Lönn, H.: Analyzing industrial simulink models by statistical model checking (2017)
18. Fulton, N., Mitsch, S., Quesel, J.D., Völp, M., Platzer, A.: KeYmaera X: An axiomatic tactical theorem prover for hybrid systems. In: Int. Conference on Automated Deduction. LNCS, vol. 9195, pp. 527–538. Springer (2015)
19. Fulton, N., Platzer, A.: Safe Reinforcement Learning via Formal Methods: Toward Safe Control Through Proof and Learning. Proceedings of the AAAI Conference on Artificial Intelligence **32** (2018)

20. Gomes, A., Mota, A., Sampaio, A., Ferri, F., Buzzi, J.: Systematic model-based safety assessment via probabilistic model checking. In: Leveraging Applications of Formal Methods, Verification, and Validation: 4th International Symposium on Leveraging Applications, ISoLA 2010, Heraklion, Crete, Greece, October 18-21, 2010, Proceedings, Part I 4. pp. 625–639. Springer (2010)
21. Gomes, A., Mota, A., Sampaio, A., Ferri, F., Watanabe, E.: Constructive model-based analysis for safety assessment. International Journal on Software Tools for Technology Transfer **14**, 673–702 (2012)
22. Gudemann, M., Ortmeier, F.: A framework for qualitative and quantitative formal model-based safety analysis. In: IEEE Int. Symposium on High Assurance Systems Engineering. pp. 132–141. IEEE (2010)
23. Hahn, E.M., Perez, M., Schewe, S., Somenzi, F., Trivedi, A., Wojtczak, D.: Faithful and effective reward schemes for model-free reinforcement learning of omega-regular objectives. In: Int. Symposium on Automated Technology for Verification and Analysis (ATVA). LNCS, vol. 12302, pp. 108–124. Springer (2020)
24. Hartmanns, A., Hermanns, H.: The Modest Toolset: An Integrated Environment for Quantitative Modelling and Verification. In: 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. LNCS, vol. 8413, pp. 593–598. Springer, Berlin, Heidelberg (2014)
25. Hasanbeig, M., Kantaros, Y., Abate, A., Kroening, D., Pappas, G.J., Lee, I.: Reinforcement Learning for Temporal Logic Control Synthesis with Probabilistic Satisfaction Guarantees. In: IEEE Conference on Decision and Control (CDC). pp. 5338–5343. IEEE, Nice, France (2019)
26. Hasanbeig, M., Abate, A., Kroening, D.: Cautious Reinforcement Learning with Logical Constraints. p. 483–491. AAMAS '20, Int. Foundation for Autonomous Agents and Multiagent Systems (2020)
27. Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., Meger, D.: Deep reinforcement learning that matters. In: Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018. vol. 32, pp. 3207–3214. AAAI Press (2018)
28. Herber, P., Reicherdt, R., Bittner, P.: Bit-precise formal verification of discrete-time MATLAB/Simulink models using SMT solving. In: Int. Conference on Embedded Software (EMSOFT). pp. 1–10. IEEE (2013)
29. Kanwar, K., Vajpai, D.J.: Performance evaluation of different models of pv panel in matlab/simulink environment. Applied Solar Energy **58**(1), 86–94 (2022)
30. Knüppel, A., Thüm, T., Schaefer, I.: GUIDO: Automated Guidance for the Configuration of Deductive Program Verifiers. In: IEEE/ACM Int. Conference on Formal Methods in Software Engineering (FormaliSE). pp. 124–129. IEEE (2021)
31. Könighofer, B., Lorber, F., Jansen, N., Bloem, R.: Shield synthesis for reinforcement learning. In: International Symposium on Leveraging Applications of Formal Methods. pp. 290–306. Springer (2020)
32. Kwiatkowska, M., Norman, G., Parker, D.: Prism: Probabilistic symbolic model checker. In: Computer Performance Evaluation: Modelling Techniques and Tools: 12th International Conference, TOOLS 2002 London, UK, April 14–17, 2002 Proceedings 12. pp. 200–204. Springer (2002)
33. Legay, A., Sedwards, S., Traonouez, L.M.: Scalable Verification of Markov Decision Processes. In: Software Engineering and Formal Methods, vol. 8938, pp. 350–362. Springer, Cham (2015), lNCS

34. Legay, A., Traonouez, L.M.: Statistical model checking of simulink models with plasma lab. In: Formal Techniques for Safety-Critical Systems: 4th International Workshop, FTSCS 2015, Paris, France, November 6-7, 2015. Revised Selected Papers 4. pp. 259–264. Springer (2016)
35. Liebrenz, T., Herber, P., Glesner, S.: Deductive verification of hybrid control systems modeled in Simulink with KeYmaera X. In: Int. Conference on Formal Engineering Methods (ICFEM). vol. 11232, pp. 89–105. Springer (2018)
36. Liebrenz, T., Herber, P., Glesner, S.: A service-oriented approach for decomposing and verifying hybrid system models. In: Int. Conference on Formal Aspects of Component Software (FACS). vol. 12018, pp. 127–146. Springer (2019)
37. Liebrenz, T., Herber, P., Glesner, S.: Service-oriented decomposition and verification of hybrid system models using feature models and contracts. Science of Computer Programming **211**, 102694 (2021)
38. Lygeros, J., Prandini, M.: Stochastic Hybrid Systems: A Powerful Framework for Complex, Large Scale Applications. European Journal of Control **16**(6), 583–594 (2010)
39. Mahto, R.K., Kaur, J., Jain, P.: Performance analysis of robotic arm using simulink. In: 2022 IEEE World Conference on Applied Intelligence and Computing (AIC). pp. 508–512. IEEE (2022)
40. Maler, O., Nickovic, D.: Monitoring Temporal Properties of Continuous Signals. In: Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems, LNCS, vol. 3253, pp. 152–166. Springer (2004)
41. Manno, G., Chiacchio, F., Compagno, L., D'Urso, D., Trapani, N.: Matcarlore: An integrated ft and monte carlo simulink tool for the reliability assessment of dynamic fault tree. Expert Systems with Applications **39**(12), 10334–10342 (2012)
42. Minopoli, S., Frehse, G.: SL2SX translator: from Simulink to SpaceEx models. In: Int. Conf. on Hybrid Systems: Computation and Control. pp. 93–98. ACM (2016)
43. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A., Veness, J., Bellemare, M., Graves, A., Riedmiller, M., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015)
44. Niehage, M., Hartmanns, A., Remke, A.: Learning optimal decisions for stochastic hybrid systems. In: ACM-IEEE Int. Conference on Formal Methods and Models for System Design (MEMOCODE). pp. 44–55. ACM (2021)
45. Niehage, M., Pilch, C., Remke, A.: Simulating hybrid petri nets with general transitions and non-linear differential equations. In: VALUETOOLS 2020: 13th EAI International Conference on Performance Evaluation Methodologies and Tools, Tsukuba, Japan, May 18-20, 2020. pp. 88–95. ACM (2020)
46. Niehage, M., Remke, A.: Learning that Grid-Convenience Does Not Hurt Resilience in the Presence of Uncertainty. In: Formal Modeling and Analysis of Timed Systems, vol. 13465, pp. 298–306. Springer International Publishing, Cham (2022), series Title: Lecture Notes in Computer Science
47. Pilch, C., Edenfeld, F., Remke, A.: HYPEG: Statistical Model Checking for hybrid Petri nets: Tool Paper. In: EAI Int. Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS). pp. 186–191. ACM Press (2017)
48. Pilch, C., Niehage, M., Remke, A.: HPnGs go Non-Linear: Statistical Dependability Evaluation of Battery-Powered Systems. In: IEEE Int. Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS). pp. 157–169. IEEE (2018)

49. Pilch, C., Remke, A.: Statistical Model Checking for Hybrid Petri Nets with Multiple General Transitions. In: Annual IEEE/IFIP Int. Conference on Dependable Systems and Networks (DSN). pp. 475–486. IEEE (2017)
50. Platzer, A.: Differential dynamic logic for hybrid systems. Journal of Automated Reasoning **41**(2), 143–189 (2008)
51. Reicherdt, R., Glesner, S.: Formal verification of discrete-time MATLAB/Simulink models using Boogie. In: Int. Conference on Software Engineering and Formal Methods. vol. 8702, pp. 190–204. Springer (2014)
52. Sadigh, D., Kim, E.S., Coogan, S., Sastry, S.S., Seshia, S.A.: A learning based approach to control synthesis of Markov decision processes for linear temporal logic specifications. In: IEEE Conference on Decision and Control. pp. 1091–1096. IEEE (2014)
53. Saraoğlu, M., Morozov, A., Söylemez, M.T., Janschek, K.: Errorsim: A tool for error propagation analysis of simulink models. In: Computer Safety, Reliability, and Security: 36th International Conference, SAFECOMP 2017, Trento, Italy, September 13-15, 2017, Proceedings 36. pp. 245–254. Springer (2017)
54. Shmarov, F., Zuliani, P.: Probabilistic hybrid systems verification via SMT and Monte Carlo techniques. In: Hardware and Software: Verification and Testing - Int. Haifa Verification Conference (HVC). LNCS, vol. 10028, pp. 152–168 (2016)
55. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. The MIT Press Cambridge, Massachusetts London, England, 2nd edn. (2018)
56. The MathWorks: Simulink . https://de.mathworks.com/products/simulink.html
57. The MathWorks: Reinforcement Learning Toolbox. https://www.mathworks.com/products/reinforcement-learning.html
58. The MathWorks: Simulink Design Verifier. https://de.mathworks.com/products/simulink-design-verifier.html
59. The MathWorks: Simulink Example: Water Distribution System Scheduling Using Reinforcement Learning. https://de.mathworks.com/help/reinforcement-learning/ug/water-distribution-scheduling-system.html
60. Tsoutsanis, E., Meskin, N., Benammar, M., Khorasani, K.: Dynamic performance simulation of an aeroderivative gas turbine using the matlab simulink environment. In: ASME International Mechanical Engineering Congress and Exposition. vol. 56246, p. V04AT04A050. American Society of Mechanical Engineers (2013)
61. Wilson, E.: Probable inference, the law of succession, and statistical inference. Journal of the American Statistical Association **22**(158), 209–212 (1927)
62. Zou, L., Zhan, N., Wang, S., Fränzle, M.: Formal Verification of Simulink/Stateflow Diagrams. In: Int. Symposium on Automated Technology for Verification and Analysis (ATVA). vol. 9364, pp. 464–481. Springer (2015)
63. Zuliani, P., Platzer, A., Clarke, E.M.: Bayesian statistical model checking with application to stateflow/simulink verification. Formal Methods in System Design **43**, 338–367 (2013)