Repairing Learning-Enabled Controllers While Preserving What Works

Pengyuan Lu
Computer and Information Science
University of Pennsylvania
Philadelphia, PA, USA
pelu@seas.upenn.edu

Matthew Cleaveland
Electrical and Systems Engineering
University of Pennsylvania
Philadelphia, PA, USA
mcleav@seas.upenn.edu

Oleg Sokolsky
Computer and Information Science
University of Pennsylvania
Philadelphia, PA, USA
sokolsky@seas.upenn.edu

Insup Lee
Computer and Information Science
University of Pennsylvania
Philadelphia, PA, USA
lee@seas.upenn.edu

Ivan Ruchkin
Electrical and Computer Engineering
University of Florida
Gainesville, FL, USA
iruchkin@ece.ufl.edu

***Abstract*—Learning-enabled controllers have been adopted in various cyber-physical systems (CPS). When a learning-enabled controller fails to accomplish its task from a set of initial states, researchers leverage repair algorithms to fine-tune the controller's parameters. However, existing repair techniques do not preserve previously correct behaviors. Specifically, when modifying the parameters to repair trajectories from a subset of initial states, another subset may be compromised. Therefore, the repair may break previously correct scenarios, introducing new risks that may not be accounted for. Due to this issue, repairing the entire initial state space may be hard or even infeasible. As a response, we formulate the Repair with Preservation (RwP) problem, which calls for preserving the already-correct scenarios during repair. To tackle this problem, we design the Incremental Simulated Annealing Repair (ISAR) algorithm, which leverages simulated annealing on a barriered energy function to safeguard the already-correct initial states while repairing as many additional ones as possible. Moreover, formal verification is utilized to guarantee the repair results. Case studies on an Unmanned Underwater Vehicle (UUV) and OpenAI Gym Mountain Car (MC) show that ISAR not only preserves correct behaviors from previously verified initial state regions, but also repairs 81.4% and 23.5% of broken state spaces in the two benchmarks. Moreover, the average signal temporal logic (STL) robustnesses of the ISAR repaired controllers are larger than those of the controllers repaired using baseline methods.**

***Index Terms*—controller repair, neural network repair, learning-enabled controller**

I. Introduction

Learning-enabled controllers are increasingly being adopted in various cyber-physical systems (CPS), including autonomous vehicles, industrial control, and healthcare services [7], [25], [41], [45]. When a learned controller, such as a deep neural network, fails to accomplish its pre-defined task in some scenario, its engineers or users would like to fix its performance ideally without sacrificing the performance in other scenarios. Researchers have proposed various repair techniques to adjust a neural network's parameters, so that the network is repaired on a given set of inputs [11], [46].

Consider a motivating example of a healthcare service robot [16], [43]. The robot needs to attend to the patient in bed within a given time limit after an alarm is raised. Before the alarm, the robot can be in various physical locations, such as the living room or its charging station. Now, upon the alarm, the robot is able to reach the bed in a timely manner from almost all locations, except for the set of initial states in kitchen, where it loiters around and wastes time. Suppose the engineers have identified that the problem stems from the neural network-based controller. Then repair algorithms can be leveraged [11], [46] to fix the problem. Specifically, the ultimate goal is to adjust the controller network so that the system accomplishes the task from all initial locations, including the kitchen and other previously successful locations.

However, state-of-the-art techniques [11], [32], [44], [46] may not preserve previously successful behaviors during repair. Specifically, previously successful behaviors may be broken after the controller parameters are altered. This is because of the competing nature among the initial states: When the repair fixes the trajectories from a subset of initial states, the performance of trajectories from another subset may be compromised. In the healthcare robot example, a repair algorithm may correct the trajectories from the kitchen but break the trajectories from elsewhere, such as the living room. This may lead to unseen risks such as the robot tripping or crashing into objects, posing new potential dangers to the patient. Generally, it may be hard or even infeasible to identify a controller to accomplish the task from all initial states. Due to this reason, existing literature that aims to repair controller neural networks on the entire initial state space cannot establish guaranteed outcomes [32], [44], [46]. By contrast, instead of looking for a controller that correctly behaves under all scenarios, our approach is to preserve what is already correct while repairing as many other scenarios as possible. In the worst case, the repaired controller will still operate correctly on the previously successful initial states.

979-8-3503-6927-4/24/$31.00 ©2024 IEEE
DOI 10.1109/ICCPS61052.2024.00007

So the repair will not introduce any new erroneous behaviors.

We propose to solve the controller repair problem while safeguarding the previously successful behaviors. Specifically, our ultimate goal is to find an alternative controller π' that (i) still accomplishes the task from previously successful initial states and (ii) maximizes the number of additional repaired initial states. We denote this problem as the Repair with Preservation (RwP) problem. We identify three challenges when tackling this problem. First, the set of initial states may be uncountably infinite, posing difficulty in checking the performance from all initial states. Second, we need to protect the performance on the initial states for which the controller behaves correctly while repairing the performance of the controller on the other initial states. Third, we would like to formally prove that our repaired controller network works on both the previously successful initial states, as well as the newly repaired ones.

To solve this problem, we develop the Incremental Simulated Annealing Repair (ISAR) algorithm, which safeguards the previously successful initial states during repair and returns verified outcomes afterwards. To use this algorithm, the initial state set is partitioned into finitely many regions. Each region first passes through a sound but incomplete verifier, such as Verisig [17], [18], to formally prove whether trajectories from this entire region are able to accomplish the task. During repair, simulated annealing [19], [33], [38] is applied to a barrier-guarded energy function [5], [15], [26] to greedily repair the initial state regions one-by-one, while protecting the previously successful regions. After repair, the initial state regions will be checked by the verifier again to produce formally proven outcomes.

Case studies on an Unmanned Underwater Vehicle (UUV) and Mountain Car (MC) [1] demonstrate that ISAR is able to preserve verified correct behaviors specified in signal temporal logic (STL). Specifically, initial state regions that pass verification before repair can consistently be verified afterwards. Moreover, 81.4% and 23.5% of regions that fail verification can be repaired on UUV and MC, respectively. We show that ISAR defeats baseline methods in STL robustness, which measures STL-specified task performance. Our implementation can be found at `https://github.com/ericlupy/isar_rep`, with repeatability details in the Appendix.

Overall, we make the following contributions.

1) We formulate the Repair with Preservation (RwP) problem, which calls for protecting the initial states that already produces task-accomplishing trajectories during repair.
2) We propose the Incremental Simulated Annealing Repair (ISAR) algorithm to tackle the RwP problem , which safeguards the already-correct initial states and produces assurance on repair results via formal verification. This repair algorithm runs offline after task failures are identified in a deployed system.
3) We run ISAR on two case studies: an unmanned underwater vehicle (UUV) and OpenAI Gym Mountain Car (MC). Results show that ISAR is able to preserve correctness on verified initial state regions, while repairing 81.4% and 23.5% of broken state spaces, respectively. The ISAR-repaired controller also demonstrates higher STL robustness than baselines.

II. Related Work

Neural network repair algorithms can be divided into two major categories: modifying outputs and modifying network parameters. For modifying outputs, previous works generally patch the controller outputs to keep the system within a safe bound [21], [22], [36], [40]. However, since these methods modify on the outputs instead of the network parameters, the errors still persist if the same network is reused.

Researchers have also developed repair techniques that modify neural network parameters. Some consider specific types of networks, such as ReLU-activated layers [10] and two-level lattice structures [32]. Some focus on repairing on a finite set of inputs [37] or local input neighborhoods [23], and these procedures usually exhibit provable soundness, due to the assumption of being finite [11]. Different techniques are also adopted, including satisfiability modulo theory (SMT) solvers [4], causality analysis [39] and formal methods [6], [42]. The majority of repair publications work on general-purpose neural networks, while some literature concerns neural network-enabled controllers [32], [44], [46] that can be leveraged in CPS — here, the controller is repaired to produce state trajectories that accomplish a task, such as staying within a safe region.

So far, the repair literature has yet to discuss preserving correct behaviors during the repair. We are aware that researchers in machine learning (continual and transfer learning [3], [20], [31], [35], in particular) have already discussed a relevant topic, i.e., catastrophic forgetting prevention [9], [29], [34]. The difference between continual learning and repair is that the former adapts knowledge from one explicit domain to another, while the latter focuses on improving performance in a static domain. Whether the repair problem can be cast as implicit domain adaptation is left to future work.

III. Background

Our problem setting is based on the following formalization. Let $t = 0, 1, 2, \ldots$ denote the discrete time steps, S be a continuous physical state space, and A be a continuous action space. Let $\pi : S \mapsto A$ denote the controller, i.e. the current action $a_t = \pi(s_t)$. The controller π is parameterized by some $\theta \in$ a fixed parameter space Θ, e.g., the weights and biases for some neural network architecture. We use π_θ to denote such parameterization. The environment dynamics is $f : S \times A \mapsto S$, i.e., the next state $s_{t+1} = f(s_t, a_t)$. Next, we introduce relevant concepts.

A. Signal Temporal Logic (STL) and Robustness Score

Signal temporal logic (STL) is a logical formalism to specify properties of continuous signals, such as trajectories of physical states in a continuous state space. In CPS, STL is used for both monitoring and verification of task accomplishment

[17], [18], [30]. On a trajectory denoted as $\bar{s} := s_0 s_1 \dots s_T$, $\forall s_t \in S$, the grammar of STL is defined as

$$\varphi ::= \top \mid g(\bar{s}) < 0 \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 U_{[t_1,t_2]} \varphi_2. \quad (1)$$

Here, $\top$ is tautology and $g : S^T \mapsto \mathbb{R}$ is a real-valued function on signals. Temporal operator $U_{[t_1,t_2]}$ denotes the until operator, and $\varphi_1 U_{[t_1,t_2]} \varphi_2$ means φ_2 must hold at some time $t \in [t_1, t_2]$ and φ_1 must always hold before t. The until operator can be converted into two additional temporal operators (i) eventually/finally operator $F_{[t_1,t_2]}$ and (ii) always/globally operator $G_{[t_1,t_2]}$. The exact definition of STL semantics can be found in the original paper [24].

Conventional STL semantics produce a Boolean outcome, indicating whether a given trajectory accomplishes its task. STL is also equipped with quantitative semantics [8], [12], [14], a robustness score to evaluate the degree of a trajectory satisfying an STL formula φ. Specifically, the robustness score $\rho : S^T \times \mathbb{N}_{\geq 0} \times \Phi \mapsto \mathbb{R}$ is a real-valued function that evaluates $\rho(\bar{s}, t, \varphi)$ on three inputs: a trajectory $\bar{s}$ of length T, a starting time step t of the trajectory , and an STL formula $\varphi \in \Phi$. The reader is referred to classic sources [8] for its definition. Based on this definition, $\rho \geq 0$ means that the trajectory starts at time t with length T is able to fulfill the STL formula φ, i.e., it accomplishes the specified task. Likewise, $\rho < 0$ means it fails. The larger absolute value of ρ, the higher degree of accomplishment or failure.

We assume all tasks are given in form of STL specifications, and the following subroutine is available to compute performance on a task φ from an initial state s_0, under dynamics f and controller π.

Definition 1 (Robustness Computing Subroutine)**.** Given a dynamics f and an STL-specified task φ, a robustness computing subroutine $\mathrm{rob}_{f,\varphi} : S \times \Pi \mapsto \mathbb{R}$ is a function that computes the STL robustness of a trajectory from an initial state s_0 under a controller π. That is,

$$\mathrm{rob}_{f,\varphi}(s_0, \pi) := \rho(\bar{s}, 0, \varphi) \quad (2)$$

Here, trajectory $\bar{s} = s_0 s_1 \cdots s_T$, with $a_t = \pi(s_t)$ and $s_{t+1} = f(s_t, a_t)$ for all $t \in [0, T)$. The set Π denotes the space of all controllers.

In other words, $\mathrm{rob}_{f,\varphi}(s_0, \pi)$ evaluates the degree of task accomplishment from initial state s_0 under controller π.

B. Simulated Annealing

Simulated annealing is a stochastic optimization algorithm that aims to find global optima of an objective function, which is often equivalently referred as the *energy function* [19], [33], [38]. We denote the energy function as $e : \Theta \mapsto \mathbb{R}$, i.e., a real-valued function that depends on variables $\theta \in \Theta$.

The algorithm works as follows. At every iteration, it randomly perturbs previous variables θ to obtain new variables θ', e.g. by adding Gaussian noise to each variable. Then, the energy function is evaluated to see if the new $e(\theta')$ is better than the previous $e(\theta)$. Specifically, if $\Delta_e := e(\theta') - e(\theta) \geq 0$, the new variables are accepted. Otherwise, if $\Delta_e < 0$, a second check is performed by tossing a biased coin with acceptance probability

$$\Pr[\text{acceptance}] := \exp(-\Delta_e / \tau). \quad (3)$$

Here, $\tau > 0$ is called the temperature, with higher values leading to more exploration. This second criterion is formally known as Metropolis-Hastings criterion, which encourages exploration of new variables, even if the new energy function is slightly worse. At the end of every iteration, the temperature cools down, e.g. by multiplying a cooling factor $\alpha \in (0, 1)$. So the amount of exploration decreases throughout the procedure. The algorithm runs until either the energy function converges or a maximum number of iterations is reached.

Compared to other optimization algorithms such as gradient descent, simulated annealing is able to escape local optima due to its stochastic nature and encouragement in exploration. We will use a modified version of this technique in our main algorithm.

C. The Learning-enabled Controller Repair Problem

The repair problem on learning-enabled controllers formulated by existing literature [32], [44], [46] takes a form as follows: for all $s_0 \in S_{init}$,

$$\begin{aligned} &\text{minimize}_{\pi'}\ c(s_0, \pi, \pi') \\ &\text{subject to}\ \mathrm{rob}_{f,\varphi}(s_0, \pi') \geq 0. \end{aligned} \quad (4)$$

Here, c is a generic cost function on initial state s_0, current controller π and repaired controller π'. For example, in minimally-deviating repair [46], the cost is the difference between the trajectory from s_0 produced by the repaired controller π' and the original controller π. Recall that a non-negative robustness is equivalent to task being accomplished. Therefore, the state-of-the-art is aiming for an ambitious goal, which guarantees task accomplishment while minimizing the cost on *all* initial states.

Despite being ideal, solving for Equation (4) on all $s_0 \in S_{init}$ simultaneously can be hard or even infeasible. This is because modifying the controller parameters to optimize for some initial states may compromise the others — a competing nature among the initial states under the controller capacity and system dynamics. Therefore, we seek a feasible problem setting.

IV. Formulating the Repair with Preservation (RwP) Problem

A. A Repair Problem that Considers Initial State Competition

To identify a feasible problem, we first define a partition of the initial states S_{init} under a controller π.

Definition 2 (Initial State Partition)**.** Given dynamics $f : S \times A \mapsto S$, a controller $\pi : S \mapsto A$ partitions the initial states $S_{init} \subset S$ into *successful* and *failed* ones: $S_{init} = S_\pi^{\mathsf{s}} \cup S_\pi^{\mathsf{f}}$. That is,

$$\begin{aligned} S_\pi^{\mathsf{s}} &:= \{s_0 \in S_{init} : \mathrm{rob}_{f,\varphi}(s_0, \pi) \geq 0\} \\ S_\pi^{\mathsf{f}} &:= S_{init} \setminus S_\pi^{\mathsf{s}} \end{aligned} \quad (5)$$

Definition 2 formalizes a partition of the initial states under a controller π. In plain words, S_π^{s} is the set of initial states from which π satisfies φ ("s" stands for "success"). Likewise, S_π^{f} is the set of initial states from which π does not satisfy, or equivalently, fails on φ ("f" stands for "failure"). With the formalization above, we identify a feasible repair problem that takes account of trade-offs in initial states.

Definition 3 (Repair with Preservation (RwP) Problem)**.** The RwP problem is formalized as

$$\begin{aligned} &\text{maximize}_{\pi'} \int_{S_\pi^{\mathsf{f}}} \mathbb{1}(\text{rob}_{f,\varphi}(s_0,\pi') \geq 0) ds_0 \\ &\text{subject to } S_\pi^{\mathsf{s}} \subseteq S_{\pi'}^{\mathsf{s}}, \end{aligned} \tag{6}$$

where $\mathbb{1}(\cdot)$ is the indicator function, which returns 0 if the input proposition is false and 1 if true.

Equation (6) aims to find an alternative controller π' to repair as many previously failed initial states in S_π^{f} as possible, while protecting the previously successful initial states S_π^{s} from being compromised. Trivially, a controller that satisfies the constraint is the original controller π. Therefore, it is guaranteed that there exists a solution to this problem.

However, the objective function in Equation (6) contains a non-standard integral that is hard to compute. We hence approximate the problem by partitioning the initial state set into finitely many regions, i.e., $S_{init} = S_1 \cup \cdots \cup S_M$. There exists two types of initial state regions S_i: (i) successful: all states in the region are successful under a given controller π, i.e., $S_i \subseteq S_\pi^{\mathsf{s}}$ and (ii) failed: not all states in the region are successful under π, i.e., $S_i \not\subseteq S_\pi^{\mathsf{s}}$. The problem becomes

$$\begin{aligned} &\text{maximize}_{\pi'} \sum_{S_i \not\subseteq S_\pi^{\mathsf{s}}} \mathbb{1}(S_i \subseteq S_{\pi'}^{\mathsf{s}}) \\ &\text{subject to } S_i \subseteq S_\pi^{\mathsf{s}} \implies S_i \subseteq S_{\pi'}^{\mathsf{s}}. \end{aligned} \tag{7}$$

The repair problem in Equation (7) aims to repair as many failed regions as possible, while safeguarding the already successful ones. This discretized version of problem approaches Equation (6) as the partition becomes finer.

B. Feasible RwP Problem Aided by Verification

To solve Equation (7), one more key issue remains: how do we know whether *all states* in a region S_i would lead to success? To provide guarantees for infinite (but bounded) regions, we hereby introduce the use of a sound but incomplete verifier, such as Verisig [17], [18]. This type of verification is conservative. That is, if a region of initial states passes this verifier, it is guaranteed that all states in this region can successfully accomplish the task (i.e., the verifier is sound). However, the implication does not hold in the other direction (i.e., the verifier is incomplete). Formally, we have Definition 4.

Definition 4 (Sound but Incomplete Verifier)**.** Given dynamics f, a *verifier* of property φ is abstracted as a function $\text{ver}_{f,\varphi} : 2^S \times \Pi \mapsto \{0,1\}$. On a subset of initial states $S' \subseteq S$ for a given controller π, the verifier outputs whether all initial states in S' produce trajectories that satisfy ϕ. A verifier is sound but incomplete iff

$$\begin{aligned} &\Big(\forall S' \in 2^S \cdot \text{ver}_{f,\varphi}(S',\pi) = 1 \implies S' \subseteq S_\pi^{\mathsf{s}}\Big) \wedge \\ &\Big(\exists S'' \in 2^S \cdot S'' \subseteq S_\pi^{\mathsf{s}} \wedge \text{ver}_{f,\varphi}(S'',\pi) = 0\Big). \end{aligned} \tag{8}$$

In Equation (8), the term on the first line means soundness, and the second line means incompleteness.

With a verifier available, we aim to address the following main problem, which first requires to partition the initial state space into regions $S_1, \ldots, S_M$. We know that a region S_i is successful if it passes verification, i.e., $\text{ver}_{f,\varphi}(S_i, \pi_\theta) = 1$. We would like to protect such regions, preserving the correct behaviors. While protecting these regions, we would like to maximally repair failed regions. However, the incomplete property of the verifier means that a repaired region may still be classified as unsafe by the verifier. Consequently, we estimate a region S_i is repaired if all initial states in a uniformly sampled finite subset $\hat{S}_i \subset S_i$ can accomplish the task, i.e., STL robustness ≥ 0. The size of regions is a design choice. Since the verifier is conservative, the larger regions we have, the less likely we are able to capture the small subsets of initial states that are correct. However, larger regions would also lead to smaller numbers of regions, providing computational efficiency. Indeed, picking the sizes of regions exhibits an accuracy-efficiency trade-off. For simplicity, we assume the partition on regions is already given. Then the main problem is formalized as follows.

Main Problem (Verification-aided RwP). We would like to accomplish a given STL task φ under dynamics $f : S \times A \mapsto S$. The current controller $\pi : S \mapsto A$ is unable to accomplish this task from all initial states in $S_{init} \subset S$. Therefore, we aim to find an alternative controller π' by repairing a subset of initial states while safeguarding another. Specifically, we are given a partition $S_{init} = S_1 \cup \cdots \cup S_M$ of the initial state space. The regions to be repaired are

$$\mathcal{S}_\pi^{\mathsf{f}} := \{S_i : \exists s_0 \in \hat{S}_i, \text{rob}_{f,\varphi}(s_0,\pi) < 0\}, \tag{9}$$

where $\hat{S}_i \subset S_i$ is a uniformly and independently sampled finite subset from S_i, with size K. The regions to be protected are

$$\mathcal{S}_\pi^{\mathsf{s}} := \{S_i : \text{ver}_{f,\varphi}(S_i,\pi) = 1\} \tag{10}$$

Assume we have a robustness computation subroutine $\text{rob}_{f,\varphi} : S \times \Pi \mapsto \mathbb{R}$ as in Definition 1 and a sound but incomplete verifier $\text{ver}_{f,\varphi} : 2^S \times \Pi \mapsto \{0,1\}$ as in Definition 4. To find an alternative controller π', we solve the following optimization problem[1].

$$\begin{aligned} &\text{maximize}_{\pi'} \sum_{S_i \in \mathcal{S}_\pi^{\mathsf{f}}} \mathbb{1}(\forall s_0 \in \hat{S}_i, \text{rob}_{f,\varphi}(s_0,\pi') \geq 0) \\ &\text{subject to } S_i \in \mathcal{S}_\pi^{\mathsf{s}} \implies \text{ver}_{f,\varphi}(S_i,\pi') = 1. \end{aligned} \tag{11}$$

[1]Due to incompleteness of the verifier, there may exist some regions that does not pass verification but no failures can be found. We do not know their safety for sure and thus they are neither protected nor repaired.

That is, our goal is to design a repair algorithm that maximizes the number of repaired initial state regions and safeguard the already successful ones. The verifier provides a formal proof of constraint-satisfaction for a solution controller.

One remark is that this main problem seeks a single new control policy π'. There exists an alternative problem setting, which aims to obtain one policy per region S_i and switch between these policies at runtime. Although one model per region could provide high performance, retaining such a large number of models would require an unbounded large memory overhead, causing large latency, and energy during computation [2]. In contrast, researchers in multi-task and continual learning have already shown satisfactory performance in using one shared model for multiple sub-tasks [31]. Therefore, our research targets the solution of a shared controller representation. It would be interesting for future works to explore the alternatives.

V. Incremental Simulated Annealing Repair (ISAR) Algorithm

We design an algorithm for the main problem. First, in Section V-A, we design an optimization objective function (energy function) to be maximized. This energy function aims to improve the STL robustness of one set of initial states while protecting the positive robustness of another set of initial states. In Section V-B, we design a safeguarded version of simulated annealing to optimize the energy function. Finally, in Section V-C we propose the overall ISAR algorithm that uses the safeguarded simulated annealing as a subroutine to incrementally repair failed state regions.

A. Energy Function Design

We will use a simulated annealing-based approach to optimize the parameters θ for a controller π_θ, due to its ability to escape local optima.

To apply simulated annealing, our first step is to design the energy function. To repair a failed region of initial states $S_i \in \mathcal{S}_\pi^{\mathsf{f}}$ (as defined in the main problem), we aim to improve the STL robustness of its generated trajectories. This results in the following energy function

$$e(\theta) := \frac{1}{|S_i|}\int_{S_i} \mathrm{rob}_{f,\varphi}(s_0,\pi_\theta)ds_0. \tag{12}$$

Here, $|\cdot|$ denotes a volumetric measure on a continuous set, $|S| = \int_S ds_0$, i.e., the volume if the set S is Euclidean ($S \subset \mathbb{R}^n$). That is, we want to maximize the average STL robustness of trajectories from all initial states $s_0 \in S_i$.

However, this energy function does not protect any currently successful initial states. Formally, we want to repair a region $S_i \in \mathcal{S}_\pi^{\mathsf{f}}$ while safeguarding $S^{\mathsf{s}} := \bigcup_{S\in\mathcal{S}_\pi^{\mathsf{s}}} S$, as stated in the main problem. To do this, we employ a log-barrier function, as it is a smooth approximation of the constraint $\forall s_0 \in S^{\mathsf{s}}$, $\mathrm{rob}_{f,\varphi}(s_0,\pi_\theta) \geq 0$ and is easy to compute [5], [15], [26]. We have

$$\begin{aligned} e(\theta) := \frac{1}{|S_i|}\int_{S_i} \mathrm{rob}_{f,\varphi}(s_0,\pi_\theta)ds_0 \\ + \frac{\lambda}{|S^{\mathsf{s}}|}\int_{S^{\mathsf{s}}} \log(\mathrm{rob}_{f,\varphi}(s_0,\pi_\theta))ds_0 \end{aligned} \tag{13}$$

The hyperparameter $\lambda > 0$ is a balance factor. Larger values of λ favor protecting the successful states, while smaller values favor fixing the unsuccessful states. The energy function in Equation (13) aims to improve the robustness of all initial states in S_i, while safeguarding the robustness of all initial states in S^{s} with log barrier. Note that if the STL robustness is negative then the logarithm will be undefined. To avoid this, we set up a lower bound, e.g. -1000, that the log-robustness will take if the STL robustness is negative or if log robustness is lower than the bound.

One computational issue is that Equation (13) has no convenient closed-form solution, making it hard to evaluate directly. In response, we use Monte Carlo integration [28] to approximate the two integrals. So, we uniformly sample states from each region given in the main problem, which we denote as $\hat{S}_i \subset S_i$ and $\hat{S}^{\mathsf{s}} \subset S^{\mathsf{s}}$, respectively. The cardinality of $\hat{S}_i$ and $\hat{S}^{\mathsf{s}}$ are K and LK (K samples per region and $L = |\mathcal{S}_\pi^{\mathsf{s}}|$ regions to protect). So the energy function now becomes

$$\begin{aligned} \hat{e}(\theta) := \frac{1}{K}\sum_{s_0\in\hat{S}_i} \mathrm{rob}_{f,\varphi}(s_0,\pi_\theta) \\ + \frac{\lambda}{LK}\sum_{s_0\in\hat{S}^{\mathsf{s}}} \log(\mathrm{rob}_{f,\varphi}(s_0,\pi_\theta)). \end{aligned} \tag{14}$$

The following proposition shows that $\hat{e}(\theta)$ and $e(\theta)$ can be arbitrarily close.

Proposition 1 (Error in Monte Carlo Integrated Energy Function)**.** The energy functions $\hat{e}(\theta)$ in Equation (14) and $e(\theta)$ in Equation (13) satisfy

$$\mathbb{E}[\hat{e}(\theta)] = e(\theta) \text{ and}$$
$$\mathrm{Var}[\hat{e}(\theta)] = \left(\frac{1}{K}\right)^2 \mathrm{Var}_{\hat{S}_i}[\mathrm{rob}_{f,\varphi}] + \left(\frac{\lambda}{LK}\right)^2 \mathrm{Var}_{\hat{S}^{\mathsf{s}}}[\log\circ\,\mathrm{rob}_{f,\varphi}]. \tag{15}$$

Here, $\mathrm{Var}_{\hat{X}}[g]$ denotes the variance of a function g on a finite set of inputs $\hat{X}$.

Our proof for Proposition 1 is provided in the Appendix. Proposition 1 states that the Monte Carlo integrated energy function $\hat{e}(\theta)$ has an expected value of the energy function $\hat{e}$. Moreover, as we sample more initial states (larger K), the variance of $\hat{e}(\theta)$ decreases. Therefore, by increasing K, the approximation can be arbitrarily close to the original energy function. Specifically, we are able to compute confidence intervals based on the size of sampled state sets [27].

B. Safeguarded Simulated Annealing

The energy function in Equation (14) is a non-standard function with a potentially large number of local optima.

Therefore, we utilize simulated annealing to maximize it, which has an advantage of escaping local optima as mentioned in Section III-B. However, due to its stochastic nature, simulated annealing may compromise the protected initial states, lowering their trajectories' STL robustness. We tolerate such compromise unless the robustness drops below 0, which means the task is failed. Therefore, we propose the following safeguarded version of simulated annealing in Algorithm 1, which has an additional check on the STL robustness of the protected initial states.

Algorithm 1 Safeguarded Simulated Annealing

Input: Finite set of states to be repaired $\hat{S}_i$, finite set of states to be protected $\hat{S}^{\mathsf{s}}$, controller parameters θ, energy balance factor $\lambda > 0$, perturbation standard deviation $\sigma > 0$, initial temperature $\tau > 0$, cooling factor $\alpha \in (0, 1)$, max iteration $maxIter$

Output: Intermediate repaired controller parameters θ

1: $\hat{e}, \rho_{\hat{S}^{\mathsf{s}}} \leftarrow \text{evaluateEnergy}(\hat{S}_i, \hat{S}^{\mathsf{s}}, \theta, \lambda)$
2: **for** $iter = 1, \ldots, maxIter$ **do**
3: $\quad\theta' \leftarrow \theta + \mathcal{N}(0, \sigma^2)$
4: $\quad\hat{e}', \rho_{\hat{S}^{\mathsf{s}}} \leftarrow \text{evaluateEnergy}(\hat{S}_i, \hat{S}^{\mathsf{s}}, \theta', \lambda)$
5: $\quad\Delta_{\hat{e}} \leftarrow \hat{e}' - \hat{e}$
6: $\quad$Draw $x \sim Bernoulli(\exp(-\Delta\hat{e}/\tau))$
7: $\quad$**if** $\Delta_{\hat{e}} \geq 0$ or $x = 1$ **then**
8: $\quad\quad$**if** $\rho_{\hat{S}^{\mathsf{s}}} \geq 0$ **then**
9: $\quad\quad\quad\theta \leftarrow \theta'$, $\hat{e} \leftarrow \hat{e}'$
10: $\quad\quad$**end if**
11: $\quad$**end if**
12: $\quad\tau \leftarrow \tau \times \alpha$
13: **end for**

Algorithm 1 presents a safeguarded version of simulated annealing. Specifically, on a given S_i to be repaired, it aims to improve robustness on the finite sampled initial set $\hat{S}_i$ while protecting the robustness of $\hat{S}^{\mathsf{s}}$. The key subroutine is evaluateEnergy at lines 1 and 4, which takes as input $\hat{S}_i$, $\hat{S}^{\mathsf{s}}$, current controller parameters θ, and balance factor λ and outputs energy $\hat{e}$ as in Equation (14) and the minimum robustness $\rho_{\hat{S}^{\mathsf{s}}}$ of protected initial states, i.e.,

$$\rho_{\hat{S}^{\mathsf{s}}}(\theta) := \min_{s_0 \in \hat{S}^{\mathsf{s}}} \text{rob}_{f,\varphi}(s_0, \pi_\theta). \tag{16}$$

From lines 2 to 13 we have the main loop of simulated annealing. The parameters are perturbed with Gaussian noise at line 3, and energy is evaluated again at line 4. Line 7 provides the standard criterion check of simulated annealing as explained in Section III-B, i.e., Metropolis-Hastings criterion. However, this criterion check is insufficient to safeguard all initial states in $\hat{S}^{\mathsf{s}}$, since a new parameter θ' may break them. Hence, we add an additional safeguard on the minimum robustness at line 8, to ensure that all $s_0 \in \hat{S}^{\mathsf{s}}$ still produce trajectories that accomplish the task. Finally, the temperature cools down at line 13 to gradually discourage exploration.

C. The ISAR Algorithm

With the safeguarded simulated annealing defined in Algorithm 1, we use it as a subroutine in our main ISAR algorithm, detailed in Algorithm 2.

Algorithm 2 Incremental Simulated Annealing Repair (ISAR)

Input: Partitioned continuous initial state set $S_{init} = S_1 \cup \cdots \cup S_M$, controller parameters θ, sample size K, energy balance factor $\lambda > 0$, perturbation std $\sigma > 0$, initial temperature $\tau > 0$, cooling factor $\alpha \in (0, 1)$, max iteration $maxIter$

Output: Repaired controller parameters θ and final verification outputs $v'_1, \ldots, v'_M$

1: **for** each S_i **do**
2: $\quad v_i \leftarrow \text{ver}_{f,\varphi}(S_i, \pi_\theta)$
3: **end for**
4: **for** each S_i **do**
5: $\quad$Uniformly sample K states in S_i to form finite set $\hat{S}_i$
6: $\quad$**for** each $s_0^j \in \hat{S}_i$ **do**
7: $\quad\quad\rho_i^j \leftarrow \text{rob}_{f,\varphi}(s_0^j, \pi_\theta)$
8: $\quad$**end for**
9: **end for**
10: $\mathcal{S}^{\mathsf{s}} \leftarrow \{S_i : v_i = 1\}$ ▷ regions to be protected
11: $\hat{S}^{\mathsf{s}} \leftarrow \bigcup_{S_i \in \mathcal{S}^{\mathsf{s}}} \hat{S}_i$
12: $\mathcal{S}^{\mathsf{f}} \leftarrow \{S_i : \exists j, \rho_i^j < 0\}$ ▷ regions to be repaired
13: Sort $\mathcal{S}^{\mathsf{f}}$ in decreasing order of $\sum_j \rho_i^j$ of each S_i
14: **while** $\mathcal{S}^{\mathsf{f}}$ is not empty **do**
15: $\quad S_i \leftarrow$ first region in $\mathcal{S}^{\mathsf{f}}$ in sorted order
16: $\quad\hat{S}^{\mathsf{f}} \leftarrow \{s_0^j \in \hat{S}_i : \rho_i^j < 0\}$
17: $\quad\theta' \leftarrow \text{safeSimAnnealing}(\hat{S}^{\mathsf{f}}, \hat{S}^{\mathsf{s}}, \theta, \lambda, \sigma, \tau, \alpha, maxIter)$
18: $\quad$**if** $\theta' \neq \theta$ **then** ▷ new controller identified
19: $\quad\quad$**for** each $S_k \in \mathcal{S}^{\mathsf{f}}$ **do** ▷ re-evaluate robustness
20: $\quad\quad\quad$**for** each $s_0^j \in S_k$ **do**
21: $\quad\quad\quad\quad\rho_k^j \leftarrow \text{rob}_{f,\varphi}(s_0^j, \pi_{\theta'})$
22: $\quad\quad\quad$**end for**
23: $\quad\quad$**end for**
24: $\quad\quad$Sort $\mathcal{S}^{\mathsf{f}}$ in decreasing order of $\sum_j \rho_k^j$ of each S_k
25: $\quad\quad\mathcal{S}' \leftarrow \{S_k \in \mathcal{S}^{\mathsf{f}} : \min_j \rho_k^j \geq 0\}$ ▷ newly repaired regions
26: $\quad\quad\mathcal{S}^{\mathsf{f}} \leftarrow \mathcal{S}^{\mathsf{f}} \setminus \mathcal{S}'$, $\mathcal{S}^{\mathsf{s}} \leftarrow \mathcal{S}^{\mathsf{s}} \cup \mathcal{S}'$
27: $\quad\quad\hat{S}^{\mathsf{s}} \leftarrow \bigcup_{S_k \in \mathcal{S}^{\mathsf{s}}} \hat{S}_k$, $\theta \leftarrow \theta'$
28: $\quad$**end if**
29: **end while**
30: **for** each S_i **do**
31: $\quad v'_i \leftarrow \text{ver}_{f,\varphi}(S_i, \pi_\theta)$
32: **end for**

Algorithm 2 can be divided into three parts: preparation (lines 1-13), main repair loop (lines 14-29) and final verification (lines 30-32). In the preparation phase each initial state region S_i is verified under current controller π_θ by calling the sound but incomplete verifier $\text{ver}_{f,\varphi}$. This step can be parallelized on individual regions. At lines 4-9 we uniformly sample K initial states from each region S_i to form a finite set, in order to estimate the energy function via Monte

Carlo integrals. The robustness of each sampled initial state is evaluated at line 7. This sampling and robustness computation can also be parallelized.

The regions and states to be protected are defined in lines 11 and 12, respectively, while the regions to be repaired are defined in line 12. As a final preparation step, we sort the to-be-repaired regions by decreasing order of average sampled robustness, since we expect regions with high robustness to be easier to repair.

In the main repair loop from line 14 to 29, we first select the next failed region to repair (line 14) and identify which of its sampled states in $\hat{S}_i$ fail to accomplish the task. These states then get repaired by the safeguarded simulated annealing (Algorithm 1 called at line 17). After simulated annealing, we check whether a new controller is obtained. This information can be passed down from Algorithm 1. If we have a new controller, robustness of the sampled states in the remaining to-be-repaired regions are evaluated again (lines 19-23), and the regions are sorted based on the evaluation (line 24). Repairing a region S_i may also repair other regions, so we identify all the repaired regions as $\mathcal{S}'$ (line 25). These regions are taken away from to-be-repaired regions and added to to-be-protected ones (line 26), so that we safeguard both the previously verified regions and the newly repaired.

Finally, after the repair is done, we rerun the verifier at lines 30-32 to obtain the final verification results. The verifier is not called during the repair procedure because it is generally expensive compared to computing robustness on individual sampled states.

We identify two potential challenges in execution of Algorithm 2. First, there may exist a protected region that does not pass verification at the end. One thing we can do is to increase the number of protected sampled states (larger K) in that region, or to adjust the balance factor λ. However, we did not encounter this case during case studies. Second, the main repair loop can take a long time. In practice, this loop can be terminated early when the repairing speed is slow, i.e., very few to no additional regions are repaired and removed at line 26. This observation implies that θ is converged.

VI. Case Studies

A. Experiment Setup

We perform two case studies. In both, the hyperparameters in Algorithm 2 are $K = 100$, $\lambda = 1$, $\sigma = 0.01$, $\tau = 1$, $\alpha = 0.95$, $maxIter = 100$.

We compare ISAR with two baselines: [2]

1) **Gradient ascent:** When maximizing the energy function in Equation (14), this baseline performs gradient ascent instead of simulated annealing. To compute the gradient $\partial\hat{e}/\partial\theta$, we soften the STL robustness function to make the energy function differentiable, as per the existing literature [13]. Then the controller parameter θ is updated as $\theta' \leftarrow \theta + \eta \times (\partial\hat{e}/\partial\theta)$, with step size $\eta \in \{0.01, 0.001, 0.0001\}$. A new controller is accepted only if it does not lower the protected states' robustness below 0.
2) **Non-safeguarded simulated annealing:** This baseline maximizes an energy function without log barrier, i.e., the Monte Carlo integral approximation of Equation (12). No initial states are protected.

[2]We reached out to the authors of another relevant technique [46] to use it as a baseline but did not receive the source code in time for this submission.

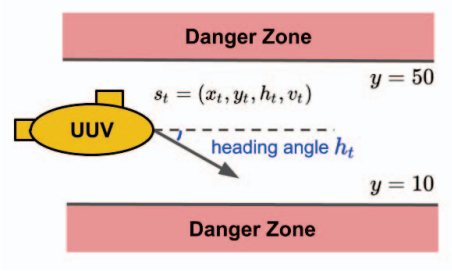

Fig. 1. Unmanned Underwater Vehicle

The following concepts help easily parse our results.

1) $\mathcal{S}^{\mathsf{s}}_{\pi}$: the set of regions that pass verification under a controller π (same as Equation (10)).
2) $\tilde{\mathcal{S}}^{\mathsf{s}}_{\pi}$: the set of regions that do not pass verification under a controller π, but for which no failure is identified by sampling $K = 100$ initial states per region.
3) $\mathcal{S}^{\mathsf{f}}_{\pi}$: the set of regions with failure sampled under a controller π (same as Equation (9)).

We track three metrics according to the main problem. First, we check how many verified regions are lost, i.e., the number of regions in $\mathcal{S}^{\mathsf{s}}_{\pi}$ that no longer pass verification. Second, we check how many broken regions are repaired, i.e., the number of regions in $\mathcal{S}^{\mathsf{f}}_{\pi}$ that no longer contain any failed sampled states. Third, we evaluate the STL robustness of the sampled states. We use Verisig [17], [18] as our verifier.

The experiments are run on Intel(R) Xeon(R) Gold 6148 CPU @ 2.40Hz. All parallel computations are distributed to CPUs of this type. The OS is Ubuntu 20.04.6 LTS, with kernel Linux 5.4.0-159-generic and architecture x86-64. We provide our code in a repeatability package, detailed in the Appendix.

B. Case Study 1: Unmanned Underwater Vehicle

The Unmanned Underwater Vehicle (UUV) control problem is based on a challenge problem from the DARPA Assured Autonomy program [17], [30]. The UUV has a four-dimensional state space (x_t, y_t, h_t, v_t), where (x_t, y_t) are the two-dimensional coordinates, h_t is the heading angle and v_t is the velocity, as illustrated in Figure 1. The x-coordinate always starts at 0, and the velocity is fixed at 0.4855 m/s.

The UUV must travel within a range of distances from a pipe that it is scanning. The upper and lower distance bounds are at $y = 10$ and $y = 50$, respectively. The controller's goal is to keep the UUV within this safe range for the next 30 seconds. We formalize this task in STL as

$$\varphi = G_{t\in[0,30]}((y_t > 10) \wedge (y_t < 50)). \tag{17}$$

TABLE I
UUV REPAIR RESULTS ($\pi' = \pi$ BEFORE REPAIR)

	$\lvert\mathcal{S}^{\mathrm{s}}_{\pi'}\rvert : \lvert\tilde{\mathcal{S}}^{\mathrm{s}}_{\pi'}\rvert : \lvert\mathcal{S}^{\mathrm{f}}_{\pi'}\rvert$	# of regions in $\mathcal{S}^{\mathrm{s}}_{\pi}$ broken	# of regions in $\mathcal{S}^{\mathrm{f}}_{\pi}$ repaired	Min rob per region in $\mathcal{S}^{\mathrm{f}}_{\pi'}$	Min rob per region in $\mathcal{S}^{\mathrm{s}}_{\pi'} \cup \tilde{\mathcal{S}}^{\mathrm{s}}_{\pi'}$	Min rob per region overall
Before repair	$141 : 963 : 896$	N/A	N/A	-0.24 ± 0.06	2.79 ± 1.89	1.49 ± 2.05
Gradient ascent	$141 : 963 : 896$	0 (0%)	0 (0%)	-0.24 ± 0.06	2.79 ± 1.89	1.49 ± 2.05
Non-safeguarded sim annealing	$132 : 1702 : 166$	9 (6.4%)	730 (81.4%)	-0.1 ± 0.03	4.73 ± 2.79	4.27 ± 3.01
ISAR (ours)	$173 : 1661 : 166$	0 (0%)	730 (81.4%)	-0.1 ± 0.04	4.92 ± 2.8	4.51 ± 3.02

TABLE II
MEANS AND STANDARD DEVIATIONS OF ISAR COMPUTATION TIMES IN THE UUV CASE STUDY

	Verification (per region)	Sim annealing (per iter)	Rob. check (per iter)
Time (s)	233.29 ± 262.69	180.63 ± 13.1	50.2 ± 45.1

Every second, the UUV gets two measurements: the heading angle to the pipe and the distance to the lower edge of pipe. We assume the measurement noise is negligible. The UUV then computes a one-dimensional turning angle action Δh_t based on the two inputs via a neural network controller, which has 2 hidden layers with 32 neurons each. Each layer is tanh-activated [17], [30].

The initial state space of the UUV is $S_{init} = \{(y, h) \mid y \in [12, 22], h \in [10, 30]\}$. To repair the controller, we partition S_{init} into rectangular regions, with step size 0.1m for y and 1.0 degree for h. As a result, we obtained 2000 regions in total.

With an alternative π' identified for each method, Table I shows the repair results. First, the gradient ascent method fails to identify new controller parameters without lowering at least one protected initial state's robustness below 0 under all step sizes η. In other words, it cannot escape its local optimum without compromising existing correct states. This shows the STL robustness's sensitivity to controller parameters. For both non-safeguarded simulated annealing and ISAR, we are able to repair 81.4% of regions with failure. However, due to a lack of protection, this baseline is unable to preserve all the verified regions, with 9 verified regions becoming broken. On the other hand, ISAR not only protects but also increments the verified regions from 141 to 173, as illustrated in the left column of Figure 2. Moreover, ISAR also obtains higher STL robustness due to its protection. Here, we evaluate the minimum sampled robustness per region, with ISAR improving the overall score from 1.49 to 4.51 on average — approximately 3 times.

In terms of computation time, repairing the UUV controller takes 7 main repair loops in the ISAR algorithm (lines 14-30 in Algorithm 2). The major temporal overheads are recorded in Table II. Here, verification is distributed to multiple CPU cores, with one thread per region. Calling the safeguarded simulated annealing subroutine is another major overhead,

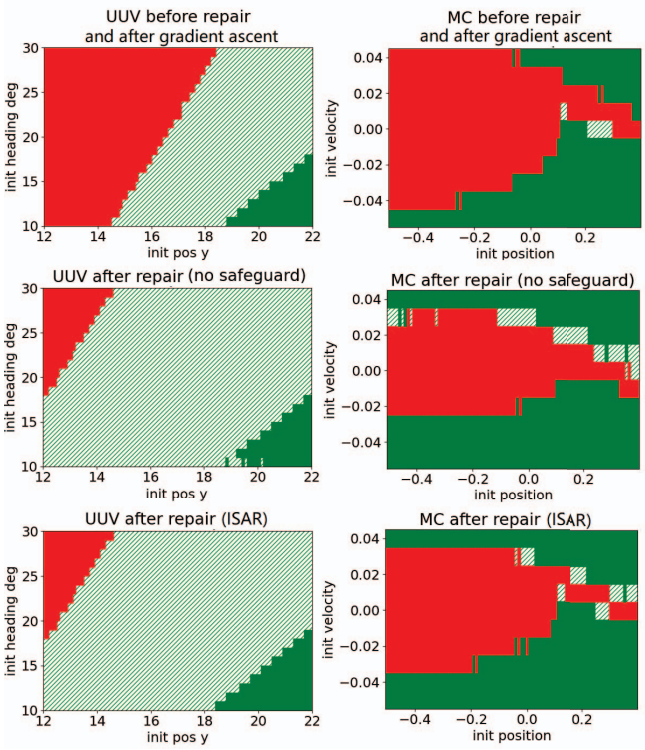

Fig. 2. Repair results on initial state spaces of UUV (left column) and MC (right column). Solid green: regions in $\mathcal{S}^{\mathrm{s}}_{\pi'}$; green hatches: regions in $\mathcal{S}^{\mathrm{s}'}_{\pi'}$; red: regions in $\mathcal{S}^{\mathrm{f}}_{\pi'}$.

with the computational overhead relatively constant at each iteration. A third overhead is checking the robustness scores on all sampled states in all regions (line 20-24). With the number of states to be checked decreasing per iteration, this overhead also decreases.

C. Case Study 2: Mountain Car

Mountain Car (MC) is a control problem in OpenAI Gym [1]. Here, we control a car with a two-dimensional state space (x_t, v_t), where x_t is the one-dimensional coordinate (in left-right direction) and v_t is the velocity. The goal is to drive the car from the bottom of a valley to the top of a mountain to

TABLE III
MOUNTAIN CAR REPAIR RESULTS ($\pi' = \pi$ BEFORE REPAIR)

	$\lvert\mathcal{S}^{\mathsf{s}}_{\pi'}\rvert : \lvert\tilde{\mathcal{S}}^{\mathsf{s}}_{\pi'}\rvert : \lvert\mathcal{S}^{\mathsf{f}}_{\pi'}\rvert$	# of regions in $\mathcal{S}^{\mathsf{s}}_{\pi}$ broken	# of regions in $\mathcal{S}^{\mathsf{f}}_{\pi}$ repaired	Min rob per region in $\mathcal{S}^{\mathsf{f}}_{\pi'}$	Min rob per region in $\mathcal{S}^{\mathsf{s}}_{\pi'} \cup \tilde{\mathcal{S}}^{\mathsf{s}}_{\pi'}$	Min rob per region overall
Before repair	$366 : 11 : 523$	N/A	N/A	-0.37 ± 0.22	0.14 ± 0.02	-0.12 ± 0.3
Gradient ascent	$366 : 11 : 523$	0 (0%)	0 (0%)	-0.37 ± 0.22	0.14 ± 0.02	-0.12 ± 0.3
Non-safeguarded sim annealing	$485 : 52 : 363$	29 (7.9%)	189 (36.1%)	-0.39 ± 0.19	0.14 ± 0.02	-0.05 ± 0.29
ISAR (ours)	$469 : 21 : 410$	0 (0%)	123 (23.5%)	-0.39 ± 0.17	0.15 ± 0.01	-0.03 ± 0.27

TABLE IV
MEANS AND STANDARD DEVIATIONS OF ISAR COMPUTATION TIMES IN THE MC CASE STUDY

	Verification (per region)	Sim annealing (per iter)	Rob. check (per iter)
Time (s)	1959.58 ± 557.5	155.27 ± 10.5	24.52 ± 23.6

its right ($x \geq 0.45$) within 110 seconds, formalized in STL as

$$\varphi = F_{t\in[0,110]}(x_t \geq 0.45). \tag{18}$$

For MC, we repair a feed-forward neural network controller π [30], with 2 hidden layers of 16 neurons each. The hidden layers are sigmoid-activated and the output layer is tanh-activated. The initial state space is $S_{init} = \{(x, v) : x \in [-0.505, 0.395], v \in [-0.055, 0.045]\}$, with partition step sizes of 0.1 and 0.01, respectively. There are 900 regions in total.

Like with the UUV, we list the results in Table III. Again, gradient ascent is unable to escape the local optimum and does not find an alternative controller. Although non-safeguarded simulated annealing repairs more failed regions, the cost is breaking 29 verified ones, whereas ISAR does not break any. The regions are illustrated in the right column of Figure 2. The maximal STL robustness achievable in this case study is 0.15, since the largest x-coordinate is 0.6 and the robustness score is the maximal value of $x_t - 0.45$. Overall, ISAR is able to achieve higher STL robustness on average. The computation times are shown in Table IV. The verification time is longer than UUV due to the longer time horizon of 110 seconds.

VII. DISCUSSION AND CONCLUSION

Customized Criteria for Protection and Repair. We use a sound but incomplete verifier to identify the states to be protected, and STL robustness to construct the objective function in ISAR. However, our algorithm can be flexibly applied to customized criteria. For example, one can use arbitrary rules like "protecting the initial states in the living room" when repairing a service robot controller, simply because one cares about these initial conditions. User customization is also enabled by different numbers of sampled states per region K and balance factor λ. Different K's and λ's can be assigned to different regions to signify degrees of importance.

Connection to Preventing Catastrophic Forgetting. As mentioned in Section II, the RwP problem is similar to preventing catastrophic forgetting in continual and transfer learning: both aim to preserve existing knowledge in a learned agent. A future research subject is to analyze the relationship between these two, to see whether the RwP problem can be reduced to domain adaptation.

To summarize, this paper identified a gap in the repair algorithms of learning-enabled controllers, that current techniques have yet to preserve existing correct behaviors during repair. To fill this gap, we formulate the RwP problem and its corresponding solution algorithm. Case studies show that our ISAR algorithm is not only able to preserve previously verified initial state regions — but also repair a large set of incorrect ones, improving a CPS controller's STL robustness with respect to its task and outperforming simpler baselines.

ACKNOWLEDGEMENT

This research was supported in part by NSF 2143274 and by ARO W911NF-20-1-0080. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office or the U.S. Government.

REFERENCES

[1] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[2] Francky Catthoor, Sven Wuytack, GE De Greef, Florin Banica, Lode Nachtergaele, and Arnout Vandecappelle. *Custom memory management methodology: Exploration of memory organisation for embedded multimedia system design*. Springer Science & Business Media, 2013.

[3] Zhiyuan Chen and Bing Liu. *Lifelong machine learning*, volume 1. Springer, 2018.

[4] Dor Cohen and Ofer Strichman. Automated repair of neural networks. *arXiv preprint arXiv:2207.08157*, 2022.

[5] D Den Hertog, Cornelis Roos, and Tamás Terlaky. On the classical logarithmic barrier function method for a class of smooth convex programming problems. *Journal of Optimization Theory and Applications*, 73(1):1–25, 1992.

[6] Guoliang Dong, Jun Sun, Jingyi Wang, Xinyu Wang, and Ting Dai. Towards repairing neural networks correctly. *arXiv preprint arXiv:2012.01872*, 2020.

[7] Pablo Escandell-Montero, Milena Chermisi, Jose M Martinez-Martinez, Juan Gomez-Sanchis, Carlo Barbieri, Emilio Soria-Olivas, Flavio Mari, Joan Vila-Francés, Andrea Stopper, Emanuele Gatti, et al. Optimization of anemia treatment in hemodialysis patients via reinforcement learning. *Artificial intelligence in medicine*, 62(1):47–60, 2014.

[8] Georgios E Fainekos and George J Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science*, 410(42):4262–4291, 2009.

[9] Robert M French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999.

[10] Feisi Fu and Wenchao Li. Sound and complete neural network repair with minimality and locality guarantees. *arXiv preprint arXiv:2110.07682*, 2021.

[11] Feisi Fu, Zhilu Wang, Jiameng Fan, Yixuan Wang, Chao Huang, Xin Chen, Qi Zhu, and Wenchao Li. Reglo: Provable neural network repair for global robustness properties. In *Workshop on Trustworthy and Socially Responsible Machine Learning, NeurIPS 2022*, 2022.

[12] Yann Gilpin, Vince Kurtz, and Hai Lin. A smooth robustness measure of signal temporal logic for symbolic control. *IEEE Control Systems Letters*, 5(1):241–246, 2020.

[13] Iman Haghighi, Noushin Mehdipour, Ezio Bartocci, and Calin Belta. Control from signal temporal logic specifications with smooth cumulative quantitative semantics. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, pages 4361–4366. IEEE, 2019.

[14] Nathaniel Hamilton, Preston K Robinette, and Taylor T Johnson. Training agents to satisfy timed and untimed signal temporal logic specifications with reinforcement learning. In *International Conference on Software Engineering and Formal Methods*, pages 190–206. Springer, 2022.

[15] John Hauser and Alessandro Saccon. A barrier function method for the optimization of trajectory functionals with constraints. In *Proceedings of the 45th IEEE Conference on Decision and Control*, pages 864–869. IEEE, 2006.

[16] Jane Holland, Liz Kingston, Conor McCarthy, Eddie Armstrong, Peter O'Dwyer, Fionn Merz, and Mark McConnell. Service robots in the healthcare sector. *Robotics*, 10(1):47, 2021.

[17] Radoslav Ivanov, Taylor Carpenter, James Weimer, Rajeev Alur, George Pappas, and Insup Lee. Verisig 2.0: Verification of neural network controllers using taylor model preconditioning. In *International Conference on Computer Aided Verification*, pages 249–262. Springer, 2021.

[18] Radoslav Ivanov, James Weimer, Rajeev Alur, George J Pappas, and Insup Lee. Verisig: verifying safety properties of hybrid systems with neural network controllers. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pages 169–178, 2019.

[19] Scott Kirkpatrick, C Daniel Gelatt Jr, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[20] Bing Liu. Lifelong machine learning: a paradigm for continuous learning. *Frontiers of Computer Science*, 11:359–361, 2017.

[21] Pengyuan Lu, Ivan Ruchkin, Matthew Cleaveland, Oleg Sokolsky, and Insup Lee. Causal repair of learning-enabled cyber-physical systems. In *2023 IEEE International Conference on Assured Autonomy (ICAA)*, pages 1–10, 2023.

[22] Deyun Lyu, Jiayang Song, Zhenya Zhang, Zhijie Wang, Tianyi Zhang, Lei Ma, and Jianjun Zhao. Autorepair: Automated repair for ai-enabled cyber-physical systems under safety-critical conditions. *arXiv preprint arXiv:2304.05617*, 2023.

[23] Keyvan Majd, Siyu Zhou, Heni Ben Amor, Georgios Fainekos, and Sriram Sankaranarayanan. Local repair of neural networks using optimization. *arXiv preprint arXiv:2109.14041*, 2021.

[24] Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In *International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 152–166. Springer, 2004.

[25] Clara Mosquera-Lopez, Leah M Wilson, Joseph El Youssef, Wade Hilts, Joseph Leitschuh, Deborah Branigan, Virginia Gabo, Jae H Eom, Jessica R Castle, and Peter G Jacobs. Enabling fully automated insulin delivery through meal detection and size estimation using artificial intelligence. *npj Digital Medicine*, 6(1):39, 2023.

[26] Roman Polyak. Modified barrier functions (theory and methods). *Mathematical programming*, 54:177–222, 1992.

[27] Kristopher J Preacher and James P Selig. Advantages of monte carlo confidence intervals for indirect effects. *Communication Methods and Measures*, 6(2):77–98, 2012.

[28] Christian P Robert, George Casella, Christian P Robert, and George Casella. Monte carlo integration. *Monte Carlo statistical methods*, pages 71–138, 1999.

[29] Anthony Robins. Catastrophic forgetting, rehearsal and pseudorehearsal. *Connection Science*, 7(2):123–146, 1995.

[30] Ivan Ruchkin, Matthew Cleaveland, Radoslav Ivanov, Pengyuan Lu, Taylor Carpenter, Oleg Sokolsky, and Insup Lee. Confidence composition for monitors of verification assumptions. In *2022 ACM/IEEE 13th International Conference on Cyber-Physical Systems (ICCPS)*, pages 1–12. IEEE, 2022.

[31] Paul Ruvolo and Eric Eaton. Active task selection for lifelong machine learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 27, pages 862–868, 2013.

[32] Ulices Santa Cruz, James Ferlez, and Yasser Shoukry. Safe-by-repair: A convex optimization approach for repairing unsafe two-level lattice neural network controllers. In *2022 IEEE 61st Conference on Decision and Control (CDC)*, pages 3383–3388. IEEE, 2022.

[33] Paolo Serafini. Simulated annealing for multi objective optimization problems. In *Multiple Criteria Decision Making: Proceedings of the Tenth International Conference: Expand and Enrich the Domains of Thinking and Application*, pages 283–292. Springer, 1994.

[34] Joan Serra, Didac Suris, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. In *International conference on machine learning*, pages 4548–4557. PMLR, 2018.

[35] Daniel L Silver, Qiang Yang, and Lianghao Li. Lifelong machine learning systems: Beyond learning algorithms. In *2013 AAAI spring symposium series*, 2013.

[36] Jeongju Sohn, Sungmin Kang, and Shin Yoo. Search based repair of deep neural networks. *arXiv preprint arXiv:1912.12463*, 2019.

[37] Matthew Sotoudeh and Aditya V Thakur. Provable repair of deep neural networks. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, pages 588–603, 2021.

[38] Balram Suman and Prabhat Kumar. A survey of simulated annealing as a tool for single and multiobjective optimization. *Journal of the operational research society*, 57:1143–1160, 2006.

[39] Bing Sun, Jun Sun, Long H Pham, and Jie Shi. Causality-based neural network repair. In *Proceedings of the 44th International Conference on Software Engineering*, pages 338–349, 2022.

[40] Shogo Tokui, Susumu Tokumoto, Akihito Yoshii, Fuyuki Ishikawa, Takao Nakagawa, Kazuki Munakata, and Shinji Kikuchi. Neurecover: Regression-controlled repair of deep neural networks with training history. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 1111–1121. IEEE, 2022.

[41] Cumhur Erkan Tuncali, James Kapinski, Hisahiro Ito, and Jyotirmoy V Deshmukh. Reasoning about safety of learning-enabled components in autonomous cyber-physical systems. In *Proceedings of the 55th Annual Design Automation Conference*, pages 1–6, 2018.

[42] Muhammad Usman, Divya Gopinath, Youcheng Sun, Yannic Noller, and Corina S Păsăreanu. Nn repair: Constraint-based repair of neural network classifiers. In *Computer Aided Verification: 33rd International Conference, CAV 2021, Virtual Event, July 20–23, 2021, Proceedings, Part I 33*, pages 3–25. Springer, 2021.

[43] Shaohua Wan, Zonghua Gu, and Qiang Ni. Cognitive computing and wireless communications on the edge for healthcare service robots. *Computer Communications*, 149:99–106, 2020.

[44] Xiaodong Yang, Tom Yamaguchi, Hoang-Dung Tran, Bardh Hoxha, Taylor T Johnson, and Danil Prokhorov. Neural network repair with reachability analysis. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 221–236. Springer, 2022.

[45] Chao Yu, Jiming Liu, Shamim Nemati, and Guosheng Yin. Reinforcement learning in healthcare: A survey. *ACM Computing Surveys (CSUR)*, 55(1):1–36, 2021.

[46] Weichao Zhou, Ruihan Gao, BaekGyu Kim, Eunsuk Kang, and Wenchao Li. Runtime-safety-guided policy repair. In *Runtime Verification: 20th International Conference, RV 2020, Los Angeles, CA, USA, October 6–9, 2020, Proceedings 20*, pages 131–150. Springer, 2020.

APPENDIX

PROOF OF PROPOSITION 1

We hereby provide the proof of Proposition 1.

Proof. Monte Carlo integration has the following property [28]: an integral $I = \int_X g(x)dx$ is approximated by $\hat{I} = (V/K)\sum_{x\in\hat{X}} g(x)$. Here, $\hat{X}$ consists of K uniformly and independently sampled points in X and volume $V = \int_X dx$. Existing work [28] has shown this approximation satisfies (i) $\mathbb{E}[\hat{I}] = I$ and (ii) $\text{Var}[\hat{I}] = V^2\text{Var}_{\hat{X}}[g]/K^2$.

The above is an established result of Monte Carlo integration. In our case, the energy function is a linear combination of two Monte Carlo integrals. Therefore, we have

$$\begin{aligned}
\mathbb{E}[\hat{e}(\theta)] &= \frac{1}{K}\mathbb{E}[\sum_{s_0\in\hat{S}_i}\text{rob}_{f,\varphi}(s_0,\pi_\theta)] \\
&\quad + \frac{\lambda}{LK}\mathbb{E}[\sum_{s_0\in\hat{S}^{\mathsf{s}}}\log(\text{rob}_{f,\varphi}(s_0,\pi_\theta))] \\
&= \frac{1}{|S_i|}\underbrace{\mathbb{E}[\frac{|S_i|}{K}\sum_{s_0\in\hat{S}_i}\text{rob}_{f,\varphi}(s_0,\pi_\theta)]}_{\mathbb{E}[\hat{I}]} \\
&\quad + \frac{\lambda}{|S^{\mathsf{s}}|}\underbrace{\mathbb{E}[\frac{|S^{\mathsf{s}}|}{LK}\sum_{s_0\in S^{\mathsf{s}}}\log(\text{rob}_{f,\varphi}(s_0,\pi_\theta))]}_{\mathbb{E}[\hat{I}]} \\
&= \frac{1}{|S_i|}\int_{S_i}\text{rob}_{f,\varphi}(s_0,\pi_\theta)ds_0 \\
&\quad + \frac{\lambda}{|S^{\mathsf{s}}|}\int_{S^{\mathsf{s}}}\log(\text{rob}_{f,\varphi}(s_0,\pi_\theta))ds_0 \\
&= e(\theta).
\end{aligned} \tag{19}$$

Likewise, because the initial states are independently sampled,

$$\begin{aligned}
\text{Var}[\hat{e}(\theta)] &= \frac{1}{K^2}\text{Var}[\sum_{s_0\in\hat{S}_i}\text{rob}_{f,\varphi}(s_0,\pi_\theta)] \\
&\quad + \frac{\lambda^2}{(LK)^2}\text{Var}[\sum_{s_0\in\hat{S}^{\mathsf{s}}}\log(\text{rob}_{f,\varphi}(s_0,\pi_\theta))] \\
&= \frac{1}{|S_i|^2}\underbrace{\text{Var}[\frac{|S_i|}{K}\sum_{s_0\in\hat{S}_i}\text{rob}_{f,\varphi}(s_0,\pi_\theta)]}_{\text{Var}[\hat{I}]} \\
&\quad + \frac{\lambda^2}{|S^{\mathsf{s}}|^2}\underbrace{\text{Var}[\frac{|S^{\mathsf{s}}|}{LK}\sum_{s_0\in\hat{S}_2}\log(\text{rob}_{f,\varphi}(s_0,\pi_\theta))]}_{\text{Var}[\hat{I}]} \\
&= \frac{1}{|S_i|^2}\frac{|S_i|^2\text{Var}_{\hat{S}_i}[\text{rob}_{f,\varphi}]}{K^2} + \frac{\lambda^2}{|S^{\mathsf{s}}|^2}\frac{|S^{\mathsf{s}}|^2\text{Var}_{\hat{S}^{\mathsf{s}}}[\log\circ\text{rob}_{f,\varphi}]}{(LK)^2} \\
&= \frac{1}{K^2}\text{Var}_{\hat{S}_i}[\text{rob}_{f,\varphi}] + \frac{\lambda^2}{(LK)^2}\text{Var}_{\hat{S}^{\mathsf{s}}}[\log\circ\text{rob}_{f,\varphi}].
\end{aligned} \tag{20}$$

□

REPEATABILITY PACKAGE

We provide a repeatability package that includes the code to reproduce the results of UUV and MC case studies as described in Section VI. It will output figures that distinguishes between three types of regions (counterexamples found, counterexamples not found but verification fails, verification succeeds) as in Figure 2 using `matplotlib`. It will also output results in the same way as in Table I and Table III in standard output. Moreover, running the verifier will output `.txt` log files that record verification time, and running the incremental repair algorithm will output repair time after each iteration. These are the times we recorded as in Table II and IV. Please refer to our GitHub repository for detailed instructions.

The prerequisites of our code are already configured in the Docker image, and there is no specific requirement on the system. However, we recommend to run our code (especially the verification part) on a machine with a large number of CPUs available. This will significantly reduce the overall computational time by dispatching one thread per CPU. In our experiments, we parallelize the verification on 40 CPUs.