Integrating Graceful Degradation and Recovery through Requirement-driven Adaptation

Simon Chu
cchu2@andrew.cmu.edu
Carnegie Mellon University
Pittsburgh, PA, USA

Justin Koe
jkoe469@gmail.com
The Cooper Union
New York, NY, USA

David Garlan
dg4d@andrew.cmu.edu
Carnegie Mellon University
Pittsburgh, PA, USA

Eunsuk Kang
eunsukk@andrew.cmu.edu
Carnegie Mellon University
Pittsburgh, PA, USA

ABSTRACT

Cyber-physical systems (CPS) are subject to environmental uncertainties such as adverse operating conditions, malicious attacks, and hardware degradation. These uncertainties may lead to failures that put the system in a sub-optimal or unsafe state. Systems that are resilient to such uncertainties rely on two types of operations: (1) *graceful degradation*, for ensuring that the system maintains an acceptable level of safety during unexpected environmental conditions and (2) *recovery*, to facilitate the resumption of normal system functions. Typically, mechanisms for degradation and recovery are developed independently from each other, and later integrated into a system, requiring the designer to develop an additional, ad-hoc logic for activating and coordinating between the two operations.

In this paper, we propose a self-adaptation approach for improving system resiliency through automated triggering and coordination of graceful degradation and recovery. The key idea behind our approach is to treat degradation and recovery as *requirement-driven* adaptation tasks: Degradation can be thought of as temporarily *weakening* original (i.e., ideal) system requirements to be achieved by the system, and recovery as *strengthening* the weakened requirements when the environment returns within an expected operating boundary. Furthermore, by treating weakening and strengthening as *dual operations*, we argue that a single requirement-based adaptation method is sufficient to enable coordination between degradation and recovery. Given system requirements specified in *signal temporal logic (STL)*, we propose a run-time adaptation framework that performs degradation and recovery in response to environmental changes. We describe a prototype implementation of our framework and demonstrate the feasibility of the proposed approach using a case study in unmanned underwater vehicles.

KEYWORDS

Graceful degradation, recovery, self-adaptive systems, requirement-driven adaptation, signal temporal logic

SEAMS '24, April 15–16, 2024, Lisbon, AA, Portugal
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0585-4/24/04
https://doi.org/10.1145/3643915.3644090

ACM Reference Format:
Simon Chu, Justin Koe, David Garlan, and Eunsuk Kang. 2024. Integrating Graceful Degradation and Recovery through Requirement-driven Adaptation . In *19th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '24), April 15–16, 2024, Lisbon, AA, Portugal.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3643915.3644090

1 INTRODUCTION

Cyber-physical systems (CPS) encompasses systems with both physical and software components, such as autonomous vehicles, unmanned aerial vehicles (UAVs), smart grids, and robots. These systems are subject to environmental uncertainties such as adverse operating conditions (e.g., severe weather for vehicles), malicious attacks, and hardware faults (e.g., damaged or flaky sensors). Due to these uncertainties, CPS may encounter failures during their operation lifecycle. For example, an autonomous vehicle may deviate from the lane boundaries due to a distracted driver or bad road conditions, drones may enter unsafe airspace due to unexpected turbulences, and a city may encounter blackouts due to unexpectedly high demands on the electricity grid.

Achieving resiliency against such failures typically involves two types of operation [15]: (1) *graceful degradation*, for ensuring that the system maintains its most critical safety functions during unexpected environmental scenarios and (2) *recovery*, to enable the resumption of normal functions as the environment returns to its expected state. Typically, mechanisms for degradation and recovery are developed independently and later integrated into a single system. For instance, a designer of an automotive safety architecture may combine a degradation mechanism that uses secondary sensors (e.g., cameras) when primary ones fail (e.g., Lidar under inclement weather) and another mechanism for re-activating an optimal system function (e.g., self-driving mode) when the environmental conditions improve (e.g., Lidar providing accurate data again). There are two challenges with this approach: (i) The designer is responsible for developing and validating an application-specific logic that decides when degradation or recovery should be activated and (ii) as system requirements evolve over time, this logic may also need to be changed.

In this paper, we propose a self-adaptation approach for improving system resiliency through automated coordination of graceful degradation and recovery. The key idea behind our approach is to treat degradation and recovery as *requirement-driven* adaptation tasks: Degradation can be thought of as temporarily *weakening* an original system requirement to be achieved by the system, and recovery as *strengthening* a previously weakened requirement when the environmental conditions improve. Furthermore, weakening and strengthening can be regarded as *dual* operations: The former relaxes the set of system behaviors that are deemed acceptable, and the latter restricts it. Based on this idea, we propose a single, unified requirement-driven adaptation framework that is capable of automatically switching between degradation and recovery, depending on the changes that arise in the environment. Our proposed approach overcomes the above two challenges, by (i) removing the need to develop an application-specific logic for coordinating degradation and recovery, and (ii) allowing the designer to plug in a different requirement without modifying the underlying logic.

To concretize this approach, we propose a self-adaptation framework that takes system requirements specified in *signal temporal logic (STL)* [19], a type of temporal logic that is particularly well-suited for specifying time-varying behaviors of CPS (e.g., "The vehicle must never deviate outside the lane for more than 2 seconds"). We show how the weakening and strengthening operations can be formulated formally as the problem of relaxing and strengthening a given STL specification, respectively. In addition, it would be desirable to reduce the impact of degradation (i.e., apply minimal weakening necessary) and maximize the rate of recovery (i.e., strengthen the requirement as much as possible). To support such *optimal* degradation and recovery, we also describe how the problem of generating *minimal weakening* and *maximal strengthening* can be encoded and solved as an instance of *mixed-integer linear programming (MILP)*.

We have developed a prototype implementation of our proposed adaptation framework. To demonstrate its feasibility, we have applied the framework to a case study involving an *unmanned underwater vehicle (UUV)*, where the system may encounter environmental uncertainties such as low water visibility and thruster failures while on a mission to inspect an underwater pipeline [29]. We compare our approach against the state-of-the-art adaptation framework called TOMASys [5]. Our experimental results are promising, showing that our approach can achieve a higher level of requirement satisfaction throughout the adaptation process while incurring a reasonable amount of overhead.

In summary, our main contributions are as follows:

(1) A theoretical framework that combines graceful degradation and recovery using incremental weakening and strengthening of STL-based requirements.
(2) A runtime architecture that performs weakening and strengthening to support system adaptation given changing environmental conditions.
(3) An approach for enabling optimal system behavior by finding a minimal weakening or maximum strengthening through MILP.
(4) An implementation of the proposed adaptation framework and a case study involving UUVs.

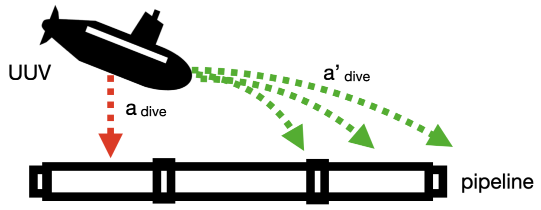

Figure 1: Illustration of UUV inspecting underwater pipeline

2 MOTIVATING EXAMPLE

Consider an UUV (illustrated in Figure 1) that is on a mission to inspect underwater pipelines, inspired by the SUAVE exemplar system [29]. The mission involves the UUV simultaneously following and inspecting a pipeline. There are two mission objectives that the system aims to achieve. First, it must maintain a clear line of sight with the underwater pipeline; if the visual contact is lost, the UUV must regain the contact within the next few seconds. Second, the thruster in the UUV must continually provide enough thrust to allow the vehicle to complete the mission within given time T.

During the mission, the UUV is subject to two sources of uncertainties: (U_1) change in water visibility and (U_2) thruster failures. These uncertainties may result in the system failing to meet the mission objectives stated above. For example, U_1 may result in the UUV losing visual contact with the pipeline and unable to regain the visual contact in time. U_2 may result in the engine being unable to provide enough thrust to complete the mission in time. In either scenario, the violation of the mission objectives may result in hardware damage and loss of the mission entirely.

Existing Approach. To safely degrade the functionality of the UUV during an adverse environmental condition, the developer may first construct a monitor that raises a warning when water visibility falls below a certain threshold (e.g., 5 meters) and triggers a certain degradation action (e.g., reduce the speed of the system or remain stationary to prevent collision with surrounding obstacles). Separately, to support recovery, another monitor may be created to look for when the water visibility improves and perform appropriate recovery actions (e.g., dive deeper to regain its visual contact).

This approach, however, has some drawbacks. First, the developer is responsible for designing triggering conditions and response actions, and validating that these actions maintain a desirable level of system utility (e.g., safety); this requires considerable domain knowledge and manual effort on the developer's part. Second, when system requirements evolve, these conditions and response actions may also need to be changed. For example, suppose that the UUV is required to float back up to the surface instead of diving deeper when the visibility falls below a certain safe threshold. Supporting this new requirement would involve designing a new controller (specific to this requirement) that triggers degradation and recovery based on the changes in the environmental condition.

Proposed Approach. To overcome these drawbacks, we take a *requirements-driven* adaptation approach. Given a user-specified requirement, our approach automatically switches between degradation and recovery by weakening and strengthening the requirement, respectively, and adjusting the system behavior to adapt to the modified requirement. In addition, our approach can determine an *optimal* amount of weakening or strengthening that is needed to degrade the system safely or bring it back to a normal operational state.

For example, suppose that the user-specified requirements for the *visibility* and *thruster* features are as follows:

> $R_{visibility}$: *Every time the visual contact with the submarine is lost, regain the contact within the next 5 seconds by diving deeper toward the pipeline.*
> $R_{thruster}$: *The thruster should provide 100 N of thrust to allow it to complete the mission on time.*

During an adverse environmental event, each of the requirements can be weakened to adapt to the changing environment. For example, the *visibility* requirement may be weakened by changing the time to regain contact from within the next 5 second to within the next 15 seconds in the case of severely low water visibility, as follows:

> $R'_{visibility}$: *Every time the visual contact with the submarine is lost, regain the contact within the next 15 seconds by diving deeper toward the pipeline.*

This weakened requirement $R'_{visibility}$ allows the visibility monitor to either delay the action to regain visual contact or descend towards the pipeline at a slower rate to ensure the safety of the vehicle. The weakening of the requirement can, in turn, increase the range of control actions the system can select from, shown in Figure 1 (action space increases from a_{dive} to a'_{dive} after weakening). Then, once the visibility has improved significantly, the vehicle may want to resume normal operation by reducing the time it takes to regain contact, by strengthening $R'_{visibility}$ back to $R_{visibility}$.

Similarly, in the case of an engine failure, the *thruster* requirement can be weakened by changing the required thrust from 100 N to some value less than 100 N, under the premise that it still provides an acceptable level of utility. One such possible weakening is as follows:

> $R'_{thruster}$: *The thruster should provide 50 N of thrust to allow it to complete the mission on time.*

This weakened requirement $R'_{thruster}$ allows certain thrusters to be turned off. Once the engine recovers (through a repair), the framework again identifies the best possible requirement for the *thruster* feature, ensuring that the UUV completes the mission in the most timely manner—by strengthening $R'_{thruster}$ and allowing the system to turn thrusters back on.

Challenges. Generally, given a requirement like $R_{visibility}$, there are numerous ways of weakening or strengthening this requirement (e.g., adjusting the time to regain visibility by different amounts). Weakening a requirement involves temporarily sacrificing a certain utility while strengthening leads to a regain of that utility. One challenge is how to systematically weaken the requirement no more than needed while strengthening the requirement to the maximum extent possible. Another challenge is given a weakened or strengthened requirement, how to reconfigure or adjust the behavior of the system to best fulfill the adjusted requirement. To tackle these challenges, we will describe (1) how requirement weakening and strengthening can be formalized using *parametric signal temporal logic (PSTL)*[3] and (2) how to perform optimal requirement adaptation by encoding it as a MILP problem.

3 PRELIMINARIES

Signals. In our approach, the behavior of CPS is modeled by real-valued continuous-time *signals*. Formally, a signal is a function $\mathbf{s} : T \rightarrow D$ mapping from a time domain, $T \subseteq \mathbb{R}_{\geq 0}$, to a tuple of k real numbers, $D \subseteq \mathbb{R}^k$. The value of a signal $\mathbf{s}(t) = (v_1, \ldots, v_k)$ represents different state variables of the system at time t (e.g., v_1 might represent the altitude of a drone).

Signal temporal logic (STL). STL extends linear temporal logic (LTL) [23] for specifying the time-varying behavior of a system in terms of signals. The basic unit of a formula in STL is a signal predicate in the form of $f(\mathbf{s}(t)) > 0$, where f is a function from D to $\mathbb{R}$; i.e., the predicate is true if and only if $f(\mathbf{s}(t))$ is greater than zero. The syntax of an STL formula φ is defined as:

$$\varphi := f(\mathbf{s}(t)) > 0 \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \mathcal{U}_{[a,b]} \varphi_2$$

where $a, b \in \mathbb{R}$ and $a < b$. The *until* operator $\varphi_1 \mathcal{U}_{[a,b]} \varphi_2$ means that φ_1 must hold until φ_2 becomes true within a time interval $[a, b]$. The until operator can be used to define two other important temporal operators: *eventually* ($\Diamond_{[a,b]}\varphi := True\ \mathcal{U}_{[a,b]}\varphi$) and *always* ($\Box_{[a,b]}\varphi := \neg\Diamond_{[a,b]}\neg\varphi$).

Robustness. Typically, the semantics of temporal logic such as LTL is based on a *binary* notion of satisfaction (i.e., formula φ is either satisfied or violated by the system). Thanks to its signal-based nature, STL also supports a *quantitative* notion of satisfaction, which allows reasoning about how "close" or "far" the system is from satisfying or violating a property. This quantitative measure is called the *robustness* of satisfaction [13].

Informally, the robustness of signal $\mathbf{s}$ with respect to formula φ at time t, denoted by $\rho(\varphi, \mathbf{s}, t)$, represents the smallest difference between the actual signal value and the threshold at which the system violates φ. For example, if the property φ says that "the drone should maintain an altitude of at least 5.0 meters," then $\rho(\varphi, \mathbf{s}, t)$ represents how close to 5.0 meters the drone maintains its altitude. Formally, robustness is defined over STL formulas as follows:

$$\begin{aligned}
\rho(f(\mathbf{s}(t)) > 0, \mathbf{s}, t) &\equiv f(\mathbf{s}(t)) \\
\rho(\neg\varphi, \mathbf{s}, t) &\equiv -\rho(\varphi, \mathbf{s}, t) \\
\rho(\varphi_1 \wedge \varphi_2, s, t) &\equiv \min\{\rho(\varphi_1, \mathbf{s}, t), \rho(\varphi_2, \mathbf{s}, t)\} \\
\rho(\Diamond_{[a,b]}\varphi, \mathbf{s}, t) &\equiv \sup_{t_1 \in [t+a, t+b]} \rho(\varphi, \mathbf{s}, t_1) \\
\rho(\Box_{[a,b]}\varphi, \mathbf{s}, t) &\equiv \inf_{t_1 \in [t+a, t+b]} \rho(\varphi, \mathbf{s}, t_1)
\end{aligned}$$

where $\inf_{x \in X} f(x)$ is the greatest lower bound of some function $f : X \rightarrow \mathbb{R}$ (and sup the least upper bound). The robustness of satisfying predicate $f(\mathbf{s}(t)) > 0$ captures how close signal $\mathbf{s}$ at time t is above zero.

Parametric signal temporal logic (PSTL). PSTL [3] is a logic obtained by replacing the constants in an STL formula with parameters. The syntax of PSTL is as follows:

$$\varphi \ := f(\mathbf{s}(t)) > a \ \mid \ \neg\varphi \ \mid \ \varphi \wedge \psi \ \mid \ \varphi \ \mathcal{U}_I \ \psi$$

Note that the syntax is similar to that of STL except both a and the time interval I can either be a parameter or a constant. In addition, there are two types of parameters in PSTL formulas: a represents the value parameter for the atomic proposition $f(\mathbf{s}(t)) > a$, whereas I represents the time parameters $[\tau_1, \tau_2]$ (where $\tau_1 < \tau_2$). A PSTL formula is denoted as $\varphi(\mathbf{p})$, where $\mathbf{p} = (p_1, ..., p_m) \in \mathcal{P}$ is the tuple of parameters appearing in the PSTL formula.

To instantiate a PSTL formula into an STL formula, a *valuation function* v is needed to map parameters to their corresponding concrete values. For example, it maps value parameters to the signal domain D, and time parameters to the time domain T. A PSTL formula combined with a valuation function v for $\mathbf{p}$ defines an STL formula $\varphi(v(\mathbf{p}))$. We say that a PSTL formula φ is satisfiable with respect to signal trace $\mathbf{s}$ if there exists an instantiated STL formula $\varphi(v(\mathbf{p}))$ such that it is satisfiable. It is formally denoted as follows:

$$(\mathbf{s}, t) \models \varphi \Leftrightarrow \exists \ \varphi(v(\mathbf{p})) \bullet (\mathbf{s}, t) \models \varphi(v(\mathbf{p})) \qquad (1)$$

For example, consider PSTL formula $\Diamond_{[\tau_1,\tau_2]}(f(\mathbf{s}(t)) > a)$. Instantiating it with the valuation function $v = \{\tau_1 \mapsto 0, \tau_2 \mapsto 5, a \mapsto 30\}$ results in the STL formula $\varphi(v(\mathbf{p})) \equiv \Diamond_{[0,5]}(f(\mathbf{s}(t) > 30)$. Given signal $\mathbf{s}(t) = (20, 30, 40...)$, PSTL formula φ is satisfiable with respect to signal $\mathbf{s}$ because the instantiation $\varphi(v(\mathbf{p}))$ is satisfiable with respect to $\mathbf{s}$.

Validity Domain. The validity domain of a PSTL formula is the set of all the parameter values that yield satisfaction for an arbitrary signal $\mathbf{s}$. We will be extending this concept in Section 5 for our adaptation framework.

4 RUNTIME ADAPTATION ARCHITECTURE

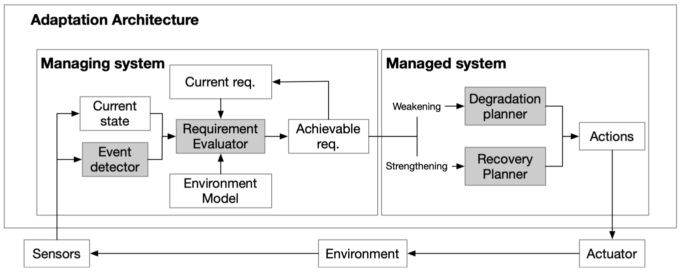

Figure 2: Proposed adaptation framework

We present an overview of our proposed runtime adaptation architecture in Figure 2. The adaptation framework periodically observes the state of the environment, determines whether adaptation is needed, and generates actions based on requirements to affect the system and environmental states.

There are three major components in the proposed adaptation architecture. First, the *event detector* looks for degradation or restoration events in the environment. Second, the *requirement evaluator* determines the achievable requirement based on the current environmental conditions. Third, the *degradation* and *recovery planner* plan future system actions based on the changing requirements. For example, if visibility decreases below a certain threshold and the weakened requirement states that "regain contact within the next 5 seconds", the degradation planner may institute a *wait* action instead of diving directly or diving with a lower speed.

When activated by the *event detector*, the *requirement evaluator* takes various inputs: (1) an environmental event, (2) the current system requirement, (3) the current state of the system, and (4) the *environmental model*, which captures how the state of the environment changes based on system actions. Using a *model-predictive control (MPC)* approach [24], the evaluator finds an optimal requirement either via strengthening or weakening of the current requirement and produces the corresponding control actions through the planner.

In the following, we describe (A) the environmental model in more detail, (B) how degradation or adaptation is triggered, and (C) building on these, a precise formulation of the requirements-driven adaptation problem.

4.1 Environment Model

Given the current requirement φ_0, the goal of the evaluator is to search for an alternative requirement φ_1 that is satisfiable. Evaluating the satisfiability of an STL formula requires knowledge of certain future steps $\mathbf{s}$, and the environmental model enables the generation of the predictive signals for these particular evaluations.

Formally, the environment model is represented as transition system $T = (Q, \mathcal{A}, \delta, Q_i)$, where:

- $Q \subseteq \mathbb{R}^k$ is the set of environment states. Each state is a combination of values for signal variables, represented as a k-dimensional tuple; $q = (v_1, ...v_k) \in Q$.
- $\mathcal{A}$ is the set of all actions, $\mathcal{A} = \mathcal{A}_{sys} \cup \mathcal{A}_{env}$, where $\mathcal{A}_{sys}$ is the set of actuator actions and $\mathcal{A}_{env}$ is the set of environmental actions. Note that $\mathcal{A}_{sys}$ and $\mathcal{A}_{env}$ are disjoint, meaning that $\mathcal{A}_{sys} \cap \mathcal{A}_{env} = \emptyset$.
- $\delta : Q \times \mathcal{A} \to Q$ is the transition function that captures how the system moves from one state to another by performing an action.
- Q_i is the set of initial states.

For example, the environment model for the pipeline inspection case study captures the current location and the velocity of the vehicle, the relative location for the pipeline, the thrust of the engine, and how the engine configuration affects the thrust. The location of the vehicle changes during each transition depending on the current velocity, which, in turn, may be modified by a system action that accelerates or decelerates the vehicle (represented by a velocity vector). Suppose the state q is represented as a tuple (vel_x, vel_y, vel_z). The next state can be computed using the previous state q, action a, and transition function δ, such that $q' = \delta(q, a)$ The example below captures the environmental model for setting the velocity of the vehicle based on the acceleration provided by the engine thrust:

$$q'.vel = (q.vel_x + q.acc_x, q.vel_y + q.acc_y, q.vel_z + q.acc_z)$$

Then, given a sequence of actions $a_1, a_2, ..., a_n$ and current state q, the environment model can be executed over these actions to generate a corresponding state sequence, $q_0, q_1, ...q_n$, which is then formed into predictive signal $\mathbf{s}$.

We do not impose restrictions on one particular notation for specifying an environment model, as long as it can be used to generate signals in the format illustrated above. With a more powerful backend like Gurobi [26], one can encode more complex environmental models like non-linear dynamics. For our implementation, we use the MiniZinc modeling language [21], which provides declarative constraints for specifying relationships between different variables of a system.

4.2 Adaptation Trigger

A key decision in our adaptation process is determining when degradation or recovery should be triggered. Degradation takes place when the system is no longer capable of satisfying the given requirement φ in the current environment. We state this condition more formally as follows:

$$\exists \mathbf{a}_{env} : Seq(\mathcal{A}_{env}) \bullet \forall \mathbf{a}_{sys} : Seq(\mathcal{A}_{sys}) \bullet$$
$$\forall \mathbf{s} : \mathbf{S} \bullet \mathbf{s} = \delta^*(q_0, \mathbf{a}_{env} \oplus \mathbf{a}_{sys}) \implies (\mathbf{s}, 0) \not\models \varphi$$

where $\mathbf{S}$ is the set of all signals, q_0 is the current state of the system, $\oplus$ is an interleaving of two sequences of actions. In other words, the above statement says that the environment can behave in a certain way (through some sequence of actions, $\mathbf{a}_{env}$) such that no matter how the system responds, it is unable to satisfy the property φ. The idea is then to carry out degradation to find and satisfy a weaker version of φ.

Similarly, the triggering condition for recovery is stated as follows:

$$\forall \mathbf{a}_{env} : Seq(\mathcal{A}_{env}) \bullet \exists \mathbf{a}_{sys} : Seq(\mathcal{A}_{sys}) \bullet$$
$$\forall \mathbf{s} : \mathbf{S} \bullet \mathbf{s} = \delta^*(q_0, \mathbf{a}_{env} \oplus \mathbf{a}_{sys}) \land (\mathbf{s}, 0) \models \varphi$$

In other words, in the current environment, the system can guarantee the satisfaction of φ, no matter how the environment behaves. Since the environment is in such an agreeable condition, the system may then attempt to improve its utility by satisfying a stronger version of φ.

Evaluating the above conditions involves generating a predictive signal ($\mathbf{s}$) that describes how the environment evolves given a sequence of system actions. However, carrying this out frequently may incur significant runtime overhead and possibly interfere with the system operations. Thus, instead of evaluating these conditions, our *event detector* looks for designated *degradation* and *restoration* events ($A_{degrade}$ and $A_{restore}$) and uses these as proxy triggers for degradation and recovery, respectively. Degradation events are abnormal events that occur due to an unexpected change in the environment, such as a thruster failure or unusually low water visibility. On the other hand, a restoration event indicates the environment returning to its previous state, such as an improvement in visibility or recovery of an engine thruster.

4.3 Adaptation Problems

We provide a precise statement of our requirements-driven adaptation problems:

Problem 4.1. ***Graceful Degradation Problem***. *Given degradation event $a_0 \in \mathcal{A}_{degrade}$ and current requirement φ_{curr}, find φ'_{curr} and action sequence $\boldsymbol{a}$ such that*

$$\forall \boldsymbol{s} : \boldsymbol{S} \bullet (\boldsymbol{s}, 0) \models \varphi_{curr} \Rightarrow (\boldsymbol{s}, 0) \models \varphi'_{curr} \land \tag{2}$$
$$\exists \boldsymbol{s}_{pred} : \boldsymbol{S} \cdot (\boldsymbol{s}_{curr} \frown \boldsymbol{s}_{pred}, 0) \models \varphi'_{curr} \land \tag{3}$$
$$\boldsymbol{s}_{pred} = \delta^*(q_0, \langle a_0 \rangle \frown \boldsymbol{a}) \tag{4}$$

where q_0 is the current state of the system (encoded in $\mathbf{s}_{curr}$) and $\frown$ is the concatenation operator that is used to link two sequences together (i.e., $\langle s_1, s_2 \rangle \frown \langle s_3, s_4 \rangle$ results in $\langle s_1, s_2, s_3, s_4 \rangle$). Degradation actions $\mathcal{A}_{degrade}$ is a subset of all environmental actions $\mathcal{A}_{env}$ (i.e., $\mathcal{A}_{degrade} \subseteq \mathcal{A}_{env}$) defined in Section 4. $\mathbf{s}_{pred}$ represents the future signal generated by the sequence of actions $\mathbf{a}$, and $\mathbf{s}_{curr}$ represents the signal that encompasses the current state, which is monitored by the sensor. Note also that the new requirement needs to be weaker than the current requirement to achieve degradation, using the definition in Eq. (2), which will be formally defined in Section 5.

Informally, this problem involves given a degradation event, how to find control actions that can satisfy an alternative requirement such that it still provides an acceptable level of system utility.

Problem 4.2. ***Recovery Problem***. *Given restoration event $a_0 \in \mathcal{A}_{restore}$ and current requirement φ_{curr}, find φ''_{curr} and $\boldsymbol{a}$ such that*

$$\forall \boldsymbol{s} : \boldsymbol{S} \bullet (\boldsymbol{s}, 0) \models \varphi''_{curr} \Rightarrow (\boldsymbol{s}, 0) \models \varphi_{curr} \land \tag{5}$$
$$\exists \boldsymbol{s}_{pred} : \boldsymbol{S} \cdot (\boldsymbol{s}_{curr} \frown \boldsymbol{s}_{pred}, 0) \models \varphi''_{curr} \land \tag{6}$$
$$\boldsymbol{s}_{pred} = \delta^*(q_0, \langle a_0 \rangle \frown \boldsymbol{a}) \tag{7}$$

Note that in the case of recovery, the new requirement needs to be stronger than the current requirement to achieve recovery, as defined in Eq. (5). The formalism for strengthening will be formally defined in Section 5. Informally, the recovery problem is framed as given a restoration event, how to find control actions that can satisfy an alternative requirement such that it can provide an improved level of system utility.

5 REQUIREMENT WEAKENING AND STRENGTHENING

In this section, we present an extension to PSTL to (1) incorporate the concept of requirement weakening and strengthening, and (2) restrict the search space of alternative requirements by providing upper and lower bounds for the validity domain of PSTL formulas. We also propose metrics to compare multiple PSTL instantiations. These metrics are then used by the MILP solver to find an optimal requirement; i.e., a minimally weakened or maximally strengthened version of the current requirement.

5.1 Minimal, Optimal and Current requirements

To enable requirement adaptation, we introduce three new concepts that guide and restrict the range of the requirement space. The *minimal requirement*, φ_{min}, represents the lower bound of the PSTL requirement, allowing for the loosening of constraints when necessary for system adaptability. Conversely, the optimal requirement, φ_{opt}, signifies the upper bound of the PSTL requirement, enabling the strengthening of the reference requirement. We assume that any requirements that are stronger than the optimal requirement do not provide additional utility for the system, and on the contrary, any requirement weaker than the minimal requirement will result in behavior that is deemed unacceptable to stakeholders.

The current requirement, denoted as φ_0, represents the requirement that the system is trying to achieve. Note that the current requirement should always be weaker than (or equal to) the optimal requirement, while always being stronger than (or equal to) the minimal requirement.

Recall the *visibility* requirement for the UUV example in Section 2. The optimal requirement, in this case, is φ_{opt}: *Every time the visual contact with the submarine is lost, regain the visual contact within the next 5 seconds*, formally denoted as $\texttt{visibility} < 20 \Rightarrow \diamond_{[0,5]}(\texttt{distance\_to\_pipe} < 10)$, assuming that visibility of 20 meters is the threshold that determines whether visual contact is maintained.

Suppose that the UUV designer is willing to accept a weakening of the time interval to regain visual contact to 15 seconds; any time above that threshold would be considered unacceptable. Thus, the requirement $\texttt{visibility} < 20 \Rightarrow \diamond_{[0,15]}(\texttt{distance\_to\_pipe} < 10)$ is designated as a minimal requirement, and any time in between can be set as the current requirement (for example, regaining visual contact within 7 seconds).

5.2 Extension to PSTL

Strengthening and weakening. Suppose φ_1 and φ_2 are two distinct instantiations of the PSTL formula φ, such that $\varphi_1 = \varphi(v_1(\mathbf{p}))$, $\varphi_2 = \varphi(v_2(\mathbf{p}))$, and $\varphi_1 \neq \varphi_2$. We say that φ_1 is weaker than φ_2 if and only if Eq. 9 is true, denoted as $\varphi_1 \ll \varphi_2$; conversely, φ_1 is stronger than φ_2 if and only if Eq. 8 holds, denoted as $\varphi_1 \gg \varphi_2$.

$$\forall \mathbf{s} \in T \to D \cdot (\mathbf{s}, 0) \models \varphi_1 \Rightarrow (\mathbf{s}, 0) \models \varphi_2 \tag{8}$$

$$\forall \mathbf{s} \in T \to D \cdot (\mathbf{s}, 0) \models \varphi_2 \Rightarrow (\mathbf{s}, 0) \models \varphi_1 \tag{9}$$

Suppose we are given a PSTL formula φ, its reference requirement φ_0, minimal requirement φ_{min} and optimal requirement φ_{opt}. We define *strong valuation* as a set of valuation functions v_s such that $\varphi_0 \ll \varphi(v_s(\mathbf{p})) \ll \varphi_{opt}$; and *weak valuation* as a set of valuation functions v_w, such that $\varphi_{min} \ll \varphi(v_w(\mathbf{p})) \ll \varphi_0$. The set of all instantiated STL formulas as a result of a strong valuation $\varphi(v_s(\mathbf{p}))$ are referred to as *strengthened formulas*; and all instantiated STL formulas as a result of a weak valuation $\varphi(v_w(\mathbf{p}))$ are referred to as *weakened formulas.*

Bounded Validity Domain. The *validity domain* of a PSTL formula [3], evaluated with respect to signal trace $\mathbf{s}$, is the set of all the parameter values that yield satisfaction for the trace $\mathbf{s}$. We extend the validity domain concept so that it is bounded by minimal and optimal requirements that we defined above in Section 5.1.

We first define the $\preceq$ operator: $\mathbf{v} \preceq \mathbf{v}'$ if and only if $\forall j \bullet \mathbf{v}_j \leq \mathbf{v}'_j$. The $\oplus$ operator in this context is used for time interval concatenation (i.e., $t \oplus [t_1, t_2]$ is equivalent to $[t + t_1, t + t_2]$). Additionally, $\mathbf{v}_{min}$ and $\mathbf{v}_{opt}$ correspond to the valuations that instantiate the minimal and optimal requirements φ_{min} and φ_{opt}, respectively. Then, the validity domain of PSTL formula φ with respect to a signal $\mathbf{s}$, bounded by $\mathbf{v}_{min}$ and $\mathbf{v}_{opt}$, is denoted as $d(\mathbf{s}, \varphi)$ and defined in the following way:

$$\begin{aligned}
d(\mathbf{s}, f(\mathbf{s}(t)) > a) \quad &= \{(t, \mathbf{v}) : f(\mathbf{s}(t)) < a_{\mathbf{v}} \wedge \\
&\quad (\mathbf{v}_{min} \preceq \mathbf{v} \preceq \mathbf{v}_{opt})\} \\
d(\mathbf{s}, \varphi \wedge \psi) \quad &= d(\mathbf{s}, \varphi) \cap d(\mathbf{s}, \psi) \\
d(\mathbf{s}, \neg\varphi) \quad &= d(\mathbf{s}, \varphi) \\
d(\mathbf{s}, \varphi\, \mathcal{U}_I\, \psi) \quad &= \{(t, \mathbf{v}) : \exists t' \in t \oplus I_{\mathbf{v}} \bullet (t', \mathbf{v}) \in d(\mathbf{s}, \psi) \\
&\quad \wedge \forall t'' \in [t, t'](t'', \mathbf{v}) \in d(\mathbf{s}, \varphi) \\
&\quad \wedge\ (\mathbf{v}_{min} \preceq \mathbf{v} \preceq \mathbf{v}_{opt})\}
\end{aligned}$$

Note that $a_{\mathbf{v}}$ and $I_{\mathbf{v}}$ here either represent constants or the concrete value assignment of the parameter a and I using valuation tuple $\mathbf{v}$. The resulting set, $d(\mathbf{s}, \varphi)$, is a set of all 2-element tuples of form $(t, v(\mathbf{p}))$, where $(\mathbf{s}, t) \models \varphi(v(\mathbf{p}))$.

Degree of Weakening and Strengthening. To quantitatively measure and compare the relative restrictiveness between two instantiated formulas (i.e., $\varphi(v_1(\mathbf{p}))$ and $\varphi(v_2(\mathbf{p}))$, which will be abbreviated to φ_1 and φ_2 below), we introduce two metrics: *degree of weakening* (Eq. 10), and its inverse metric, *degree of strengthening* (Eq. 11). These metrics are defined based on the robustness of satisfaction, as follows:

$$\Delta_{weak}(\varphi_1, \varphi_2, \mathbf{s}, t) = \rho(\varphi_2, \mathbf{s}, t) - \rho(\varphi_1, \mathbf{s}, t) \tag{10}$$

$$\Delta_{strong}(\varphi_1, \varphi_2, \mathbf{s}, t) = \rho(\varphi_1, \mathbf{s}, t) - \rho(\varphi_2, \mathbf{s}, t) \tag{11}$$

Note that for *degree of weakening* to be a positive number, φ_2 must be weaker than φ_1, and vice versa for *degree of strengthening.* We will present an example below to illustrate how these metrics are used.

Example. Recall the example in Section 2, with the requirement for the *thruster* feature: The thruster should provide 100N of thrust within the next second. The parameter 100N is subject to change. This requirement can be formalized in PSTL as $\varphi \equiv \square_{[0,1]}(\texttt{thrust} > p_1)$. The original requirement is $\varphi_{origin} \equiv \varphi(p_1 \mapsto 100) \equiv \square_{[0,1]}(\texttt{thrust} > 100)$, while one possible weakened version is $\varphi_{weak} \equiv \varphi(p_1 \mapsto 70) \equiv \square_{[0,1]}(\texttt{thrust} > 70)$.

Suppose the system evolves to generate signal $\mathbf{s}$ with a thrust of 110 and 80, at time 0 and 1 second, respectively; then $\rho(\varphi_{origin}, \mathbf{s}, 0) = -20$ (meaning φ_{origin} is violated), while $\rho(\varphi_{weak}, \mathbf{s}, 0) = 10$ (meaning φ_{weak} is satisfied. Thus, the *degree of weakening* is measured as $\Delta_{weak}(\varphi_{origin}, \varphi_{weak}, \mathbf{s}, 0) = 30$ (meaning requirement weakening from φ_{weak} to φ_{origin}), while *degree of strengthening* is measured as $\Delta_{strong}(\varphi_{weak}, \varphi_{origin}, \mathbf{s}, 0) = 30$ (meaning strengthening from φ_{weak} to φ_{origin}).

5.3 Runtime Adaptation as a MILP Formulation

The requirement adaptation process can be performed by reducing it to an instance of MILP, where the problem consists of solving for a set of decision variables that optimize certain objectives while fulfilling a set of hard constraints. We next show how minimizing the degree of weakening and maximizing the degree of strengthening can be formulated as MLIP objectives.

Problem 5.1. ***Graceful Degradation as MILP****. Given the current requirement $\varphi(v(\boldsymbol{p}))$ and its weaker version to be found, $\varphi(v'(\boldsymbol{p}))$, transition system $T = (Q, \mathcal{A}, \delta, Q_i)$, and state sequence $q^t = q_0, \ldots, q_t$*

representing the signal observed from the environment so far, compute:

$$argmin_{v'(\boldsymbol{p}),\boldsymbol{a}} \ \Delta_{weak}(\varphi(v(\boldsymbol{p})), \varphi(v'(\boldsymbol{p})), \mathbf{s}, 0) \quad (12)$$
$$s.t. \ \ \mathbf{s}_i = q_i \ for \ i \leq t \ \wedge \quad (13)$$
$$\mathbf{s}_i = \delta(\mathbf{s}_{i-1}, \mathbf{a}_{i-1}) \ for \ t < i \leq t+N \ \wedge \quad (14)$$
$$(0, v'(\boldsymbol{p})) \in d(\mathbf{s}, \varphi) \quad (15)$$

There are two decision variables to be solved in this MILP problem: parameters $v'(\mathbf{p})$ and control action sequence $\mathbf{a}$. Note that in Eq. (13), the past signal is defined up until the current time. Eq. (14) involves the prediction of a future signal from the next time step to time $t+N$ using the transition system T that encodes an environment model. $N \in \mathbb{N}$ is a finite predictive horizon inferred from the STL requirement φ. Eq. (15) guarantees that the new requirement is within the defined range of requirements by restricting the parameters of the PSTL formula to the validity domain.

Finally, by minimizing the degree of weakening from $\varphi(v(\mathbf{p}))$ to $\varphi(v'(\mathbf{p}))$ in Eq. (12), we guarantee that the system utility associated with the requirement is degraded by a minimal necessary amount.

The formulation of recovery as MILP mirrors that for degradation:

Problem 5.2. ***Recovery MILP Formulation****. Given the current requirement $\varphi(v(\boldsymbol{p}))$ and it strengthened version to be found, $\varphi(v'(\boldsymbol{p}))$, transition system $T = (Q, \mathcal{A}, \delta, Q_i)$, and state sequence $q^t = q_0, \ldots, q_t$ representing the signal observed from the environment so far, compute:*

$$argmax_{v'(\boldsymbol{p}),\mathbf{a}} \ \Delta_{strong}(\varphi(v(\boldsymbol{p})), \varphi(v'(\boldsymbol{p})), \mathbf{s}, 0) \quad (16)$$
$$s.t. \ \ \mathbf{s}_i = q_i \ for \ i \leq t \ \wedge \quad (17)$$
$$\mathbf{s}_i = \delta(\mathbf{s}_{i-1}, \mathbf{a}_{i-1}) \ for \ t < i \leq t+N \ \wedge \quad (18)$$
$$(0, v'(\boldsymbol{p})) \in d(\mathbf{s}, \varphi) \quad (19)$$

In comparison to the MILP formulation for the degradation problem, the objective here is to *maximize* the degree of strengthening between $\varphi(v(\mathbf{p}))$ and $\varphi(v'(\mathbf{p}))$, as shown in Eq. (16). This allows searching for the best possible requirement that allows the system to regain utility that was temporarily sacrificed during an earlier degradation step.

6 IMPLEMENTATION

6.1 Simulator

To illustrate our proposed requirement adaptation approach, we have developed a prototype implementation[1] based on SUAVE [29], an unmanned underwater vehicle that supports the customization of self-adaptation logic. It uses a ROS2-based platform and implements a pre-defined mission that detects, follows, and inspects a pipeline on the seabed. The backend of the simulator uses ArduSub [34], which provides various controller APIs to control the trajectory of the underwater vehicle. The vehicle trajectory and environmental setup can be visualized through the Gazebo simulator [20] with the UI shown in Fig. 3 where the seabed pipe is indicated with the yellow cylinder, while the UUV is illustrated by the rectangular box to the left side of the pipe. Various environmental entities (i.e., lighting, terrain) are listed on the panel to the right, and can be adjusted as needed.

[1]All the code, models, and experimental data are available at https://github.com/sychoo/cps-degradation-recovery

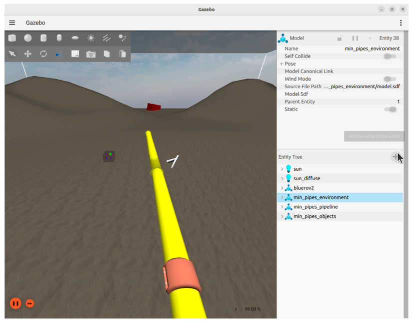

Figure 3: A screenshot of the Gazebo UUV simulator.

Features. For evaluation, we implemented two mission-related features in the UUV. When these features are activated, they generate propulsion actions (in the form of velocity vectors) or system reconfiguration actions that override those from the path planner. The implemented features are as follows:

- *Visibility Monitor* ensures that the UUV maintains sufficient visibility to the pipelines. To achieve this, it generates actions to enforce the UUV to close in on the pipelines when the visibility is below a safety threshold.
- *Thrust Monitor* ensures that the UUV maintains enough thrust to support the timely completion of the mission. The thrust monitor may change the system configuration dynamically (i.e., turning on additional thrusters when the actual thrust falls below the expected thrust.)

Environmental Anomalies. Next, we randomly injected abnormal environmental events throughout the operation of the UUV. We list two failures that we investigated:

- *Loss of visual contact with the underwater pipeline*: The low water visibility makes it difficult for the UUV to detect and follow the underwater pipeline. When it occurs during the pipeline inspection, it may result in the UUV losing sight of the pipeline and requiring it to dive deeper in the short term, regain the visual line of sight, and continue the inspection progress. The change in the water visibility is introduced randomly during the simulation.
- *Thruster failure*: A thruster failure causes the engine of the UUV to provide partial propulsion, move erratically, or shut down completely. Similarly, thruster failures are modeled and injected in a stochastic manner.

6.2 Environment Model

Our framework leverages a model of the environment during the adaptation process. For the UUV system, the environmental model captures the (1) physical dynamics of the system (2) thruster estimation from the engine, and (3) the estimated coordinates of the seabed pipeline. The dynamics model is used to estimate the velocity of the UUV based on the accelerating or decelerating actions. The thruster estimation is based on the configuration of the thrusters (i.e., which ones are turned on or off). Lastly, the estimated coordinates of the seabed pipeline are based on information received via onboard sensors. The coordinate information is also used to keep track of the inspection progress and guide the diving operation toward the pipeline.

All aspects of the environmental model are specified in the MiniZinc modeling language, translated to linear constraints and solved using MILP during the adaptation process.

7 EVALUATION

This section describes the evaluation of our proposed approach. We present the following:

- **RQ1**: Does our approach achieve a higher overall system utility than existing state-of-the-art approaches?
- **RQ2**: What is the runtime overhead of the proposed approach? Does it interfere with system operations?

7.1 Experimental Design

We conducted a set of experiments to evaluate the proposed adaptation approach through comparison against with state-of-the-art self-adaptation method in TOMASys [5].To ensure that our adaptation approach generalizes across various scenarios, we randomly generated 100 different system setups and failure scenarios, including the duration of the simulation, times when failures are injected, changes in the water visibility, number of usable thrusters, the initial position of the UUV, and the location of the underwater pipeline.

To compare our proposed method and the baseline approach, we use the *cumulative utility* as the metric, which is measured by calculating the robustness of the minimal requirement for the signal collected during the degradation and recovery process. The reason why we chose to measure the satisfaction of the minimal requirement is that it is the lower bound of the instantiated STL formula under which the system feature ceases to function and provides useful utility. We assume that the robustness metric is a suitable medium to reflect the desirability of the system behavior. The robustness is measured upon the start of the degradation events and ends upon the satisfaction of the optimal requirement (φ_{opt}) which indicates the end of the recovery. If the optimal requirement is never reached due to persistent environmental disruptions, the cumulative robustness measurement will continue until the end of the simulation life cycle.

Before we conducted the experiments for the case studies, we created the following hypotheses to be tested:

- **H1**: Our approach results in a higher cumulative utility throughout degradation and recovery processes than the existing method.
- **H2**: Our approach results in a higher run-time overhead but it does not disrupt normal system operations.

All our experiments were run on a Ubuntu desktop machine with 16 GB RAM, 6-core Intel Core i5, and a NVIDIA GeForce RTX 3060 graphics card.

7.2 Seabed Pipe Inspection Case Study

Recall the example mentioned in Section 2, where a UUV is conducting a mission to inspect underwater pipes on the seabed. The optimal STL formula φ_{opt} and the minimal requirement φ_{min} for the water visibility and thrust requirement are specified as follows:

$$\varphi_{visibility\_opt} : \square_{[0,1]}(\texttt{visibility} < 20 \Rightarrow \Diamond_{[0,5]}\texttt{distance\_to\_pipeline} < 10) \tag{20}$$

$$\varphi_{visibility\_min} : \square_{[0,1]}(\texttt{visibility} < 20 \Rightarrow \Diamond_{[0,15]}\texttt{distance\_to\_pipeline} < 10) \tag{21}$$

$$\varphi_{thruster\_opt} : \square_{[0,1]}(\texttt{thrust} > 100) \tag{22}$$

$$\varphi_{thruster\_min} : \square_{[0,1]}(\texttt{thrust} > 50) \tag{23}$$

The visibility requirement $\varphi_{visibility}$ ensures that the UUV can closely observe the pipelines upon the visibility falling below the safety threshold 20. The thrust requirements allow the system to continuously provide adequate propulsion to ensure a timely mission completion.

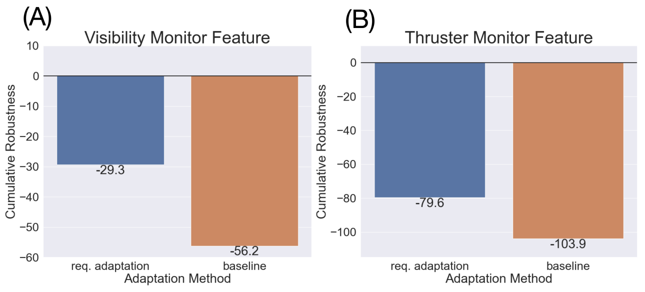

Figure 4: Cumulative robustness for visibility and thruster monitor features for the pipeline inspection case study

7.3 Experiment Results

In Fig 4, charts (A) and (B) show the cumulative robustness values for both the visibility and thruster monitor feature while the system is encountering thruster failures or low water visibility events. The results show that our adaptation approach achieves a higher cumulative robustness value than the baseline self-adaptive approach. Specifically, in the case of the water visibility monitor, it achieves an approximately 2-fold increase in cumulative robustness (with a 26.9 increase in robustness). In the case of the thruster monitor feature, our approach has a 24.3 lead in cumulative robustness.

The SUAVE artifact [29] also comes with a set of mission-related metrics that measure the quality of the mission completion. We reused some of the metrics, namely, the distance of the pipeline inspected. We discovered that our approach, on average, can inspect 86.10 meters of the pipeline while the baseline approach can only inspect 49.63 meters, an increase of about 74%. However, the standard deviation of our approach (36.68) is slightly worse than the baseline approach (23.30), meaning that the actual performance fluctuates more across various failure scenarios in the 100 experiment runs.

There are two main reasons for this increase in the cumulative utility and the quality of the mission completion. First, it is the dynamic nature of the action planning based on the specific scenario or context. For instance, TOMASys relies on a fixed set of actions (determined at design time) to address each scenario. To satisfy the objective of regaining contact with the pipeline after losing visuals, TOMASys simply has a predefined action to dive several meters down the seabed, whereas our approach predicts all possible signals given available actions using an environmental model. Then it chooses the actions that result in the most desirable signals. Second, it is the ability to flexibly adapt goals in different situations—namely, adjusting the requirements when they are not attainable or have the potential for improvements. The dynamic action planning and the requirement adjustments result in a more measurable system utility, and therefore, more robust and desirable system actions.

7.4 Performance Overhead

Since the requirement adaptation approach requires the use of MILP to search for requirements and plan actions at run time, it has a higher overhead than the baseline approach that uses actions that are defined at design time. We measure the overhead as the average additional time the system uses per control cycle as a result of deploying our adaptation approach compared to the baseline approach using TOMASys. Consequently, we measured the average overhead across both the visibility and thrust monitor features as 0.35 seconds. We have not observed noticeable delays or disruptions to our UUV during the operation, as the UUV controller was running at 2Hz, resulting in a window of 0.5 seconds for each cycle of the controller update.

Furthermore, the performance overhead is subject to the complexity of both the feature requirements and the environmental model. For example, the visibility monitor feature incurs a higher overhead because the encoding of the STL formula results in a more complex set of constraints for the MILP solver at runtime.

7.5 Threats to Validity

Internal Validity: The sampled configuration space is partial and may not capture all exceptional scenarios exhaustively. However, we have mitigated selection biases by randomizing the configuration parameters. For example, at which point failures occur, the initial location of the UUV, the setup of the underwater pipelines, etc. In addition, we are using a deterministic environmental model that captures simple physical dynamics. This is to simplify the scope of the problem and reduce the search space for the changed requirements and corresponding actions. However, the model may not capture the dynamic of the real world caused by external factors (i.e., UUVs may deviate from the direction it is accelerating towards due to water currents). We attempted to mitigate this discrepancy by manually inspecting the source code of the ArduSub simulator to ensure the model largely captures the logic of the simulator.

External Validity: The overhead of the simulation is application-specific and hardware-dependent. More restrictive hardware or outdated MILP solver may increase the runtime overhead of our framework and therefore require more performance tuning than our existing software artifacts.

Furthermore, the result of the case study may not be generalized to all applications of CPS. Depending on the specific applications (i.e., electricity grids, UAVs), the system requirement may be expressed differently, and so does the way to degrade the systems. However, we believe the concept of using requirement weakening and strengthening to perform degradation and recovery is requirement-agnostic, and that it can be generalized across different domains.

8 RELATED WORK

Graceful Degradation and Recovery. Graceful degradation is the ability of a system to maintain an acceptable level of functionality even when a significant portion of the system is rendered inoperable due to environmental disturbances. Recovery refers to the ability of the system to restore to a safe or desired state after a failure. Both areas are well-studied in control systems and human-machine interfaces. Many degradation frameworks have been developed for different applications and domains, such as adaptive cruise control [22], autonomous drones [10, 35], software user interfaces [14], industrial control systems [6, 12], and design patterns for degradation[27], respectively. There is also a large body of work on system recovery. [9] proposed an STL-based resilient framework that enables analysis of trade-offs between time-to-recovery and durability in a cruise control system using multi-objective optimization, while [17] and [31] propose checkpointing techniques (i.e., storing system traces with a finite horizon leading up to the current state). Lastly, [16] and [1] use system restart and reset to recover the system from a failure. As far as we know, no prior work provides a coordination mechanism to facilitate these two processes in a single system.

Self-adaptive Systems. Self-adaptive systems aim to design systems that are capable of adjusting to changes in the environment. The most influential reference control model in autonomic and self-adaptive systems is the MAPE-K [2] architecture, which stands for Monitor-Analyze-Plan-Execute over a shared Knowledge, on which our resolution architecture is based. Self-healing systems [28] have been proposed by Shaw, et al. to address the problem of coping with fluctuations and uncertainties in the environment. The idea is that the system always keeps a background maintenance process running regardless of the current state of the system and the environment (i.e. garbage collection, optimized network routing, etc.). This way, the system will eventually maintain internal stability and continuous improvement despite external changes. This state is also known as a homeostasis state. Our approach draws inspiration from this idea, in that we attempt to ensure that the system aims to achieve the most desirable requirement relative to the current requirement.

The concept of *meta-control* has been used for design-time analysis. For example, OpenODD [25] has been used as a scenario-based analysis framework for autonomous systems, especially in self-driving cars. TOMASys [5] provides a comprehensive set of reconfiguration strategies ranging from architectural reconfiguration, to monitor construction, and then to scenario-based definition of adaptation tactics.

Requirement Adaptation. Requirement adaptation has been investigated in the context of self-adaptive systems. The idea of weakening a requirement has been studied for feature interaction resolution [10], handling a violation of environmental assumptions and safety properties [6, 35], controller synthesis [7, 8], and goal adjustment for security-critical systems [30]. As far as we know, however, these existing works focus on how to gracefully degrade the system using requirement relaxation, instead of combining both degradation and recovery stages or attempting to coordinate them using requirement adaptation.

RELAX [32] is designed to support expressing requirements that explicitly capture uncertainties about runtime system behavior based on fuzzy temporal logic. For example, with RELAX, users can specify requirements like, "Once a user request is sent, it should be processed as early as possible". Although our approach differs in that it relies on STL as the underlying specification formalism, an interesting future work would be to investigate the utility of RELAX for the type of degradation-recovery loop that we have explored.

Both [33] and [18] study goal adaptation in the context of changing stakeholder goals or user preferences. However, their proposed methods are designed to be dependent on human inputs, not for automatically adapting to changing environments. [4] focuses on requirement adaptation based on resource constraints and proposes a 2-tier adaptation framework that (1) first tries to fulfill goals by finding alternative resources and (2) if needed, deviates the goal slightly from the original one to compensate for limited resources. Our approach differs from these approaches in that our utility is represented as the desirability of requirements over run-time signals, instead of adequate resources being allocated.

9 LIMITATIONS AND FUTURE WORK

We propose a requirement-driven runtime adaptation framework that coordinates between grace degradation and recovery. Through the case study on UUVs, we have demonstrated our proposed approach can result in more desirable system behaviors during periods of degradation and recovery.

However, our approach makes several assumptions about the system that may limit its applicability. First, our approach tackles the class of requirements can be specified using STL [19]. These requirements are specific to cyber-physical systems, where the system is time-sensitive, and sensor inputs can be monitored and transformed into continuous signals and quantified using state predicates. Thus, the framework is currently not designed to handle other classes of requirements (e.g., discrete behavioral specifications in LTL or stochastic ones in PCTL[11]).

Secondly, we assume the system has a meaningful way of quantifying the satisfaction of its requirements, these requirements can be weakened, and the user is willing to tolerate temporary degradation in the associated system utility. Thus, this framework is not well-suited for safety-critical requirements, which tend to be hard constraints and cannot be negotiated (i.e., a collision avoidance system for smart vehicles).

Thirdly, we assume that interactions between the system and the external environment (i.e. how a UUV move around the water based on an actuator command) can be predicted by a deterministic *environmental model*. However, the estimation from the environmental model may deviate from reality, due to environmental disturbance and other uncertainties. To address this, we used model predictive control [24], which involves repeated prediction and planning on a short horizon to reduce the effect of deviation in the estimation.

Another limitation of our approach is that a MILP solver may fail to find a satisfiable solution. This can be due to an overly restrictive environmental model, a limited range of alternative requirements that the solver can search for, or an unresolvable system state at the time of the adaptation. While this happened rarely in our case studies (less than 2% of adaptation cycles), we can cope with the problem through some minor tuning of the adaptation frameworks like reducing constraints to the environmental model, increasing the ranges of requirements available, or defining some simple fallback adaptation approaches, etc.

In future work, we also plan to investigate the applicability of our approach in our domains, such as robotics and cyber-security, where the ability to gracefully degrade and recover is critical for system autonomy and resilience.

10 ACKNOWLEDGEMENT

This work was supported in part by Award N00014172899 from the Office of Naval Research, Award No. H98230-23-C-0274 from the National Security Agency, and the National Science Foundation award CCF-2144860. Any views, opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding agencies.

REFERENCES

[1] Fardin Abdi Taghi Abad, Renato Mancuso, Stanley Bak, Or Dantsker, and Marco Caccamo. 2016. Reset-based recovery for real-time cyber-physical systems with temporal safety constraints. In *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*. 1–8. https://doi.org/10.1109/ETFA.2016.7733561

[2] Paolo Arcaini, Elvinia Riccobene, and Patrizia Scandurra. 2015. Modeling and Analyzing MAPE-K Feedback Loops for Self-Adaptation. In *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. 13–23. https://doi.org/10.1109/SEAMS.2015.10

[3] Eugene Asarin, Alexandre Donzé, Oded Maler, and Dejan Nickovic. 2012. Parametric Identification of Temporal Properties. In *Runtime Verification*, Sarfraz Khurshid and Koushik Sen (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 147–160.

[4] Amel Bennaceur, Andrea Zisman, Ciaran McCormick, Danny Barthaud, and Bashar Nuseibeh. 2019. Won't Take No for an Answer: Resource-Driven Requirements Adaptation. In *2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. 77–88. https://doi.org/10.1109/SEAMS.2019.00019

[5] Julita Bermejo-Alonso, Carlos Hernández, and Ricardo Sanz. 2016. Model-based engineering of autonomous systems using ontologies and metamodels. In *2016 IEEE International Symposium on Systems Engineering (ISSE)*. 1–8. https://doi.org/10.1109/SysEng.2016.7753185

[6] Titus Buckworth, Dalal Alrajeh, Jeff Kramer, and Sebastian Uchitel. 2023. Adapting Specifications for Reactive Controllers. In *2023 IEEE/ACM 18th Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. 1–12. https://doi.org/10.1109/SEAMS59076.2023.00012

[7] Ali Tevfik Buyukkocak and Derya Aksaray. 2022. Temporal Relaxation of Signal Temporal Logic Specifications for Resilient Control Synthesis. In *2022 IEEE 61st Conference on Decision and Control (CDC)*. 2890–2896. https://doi.org/10.1109/CDC51059.2022.9992914

[8] Davide G. Cavezza, Dalal Alrajeh, and András György. 2018. A Weakness Measure for GR(1) Formulae. *Formal Aspects of Computing* 33 (2018), 27 – 63. https://api.semanticscholar.org/CorpusID:13688783

[9] Hongkai Chen, Scott A. Smolka, Nicola Paoletti, and Shan Lin. 2022. An STL-based Approach to Resilient Control for Cyber-Physical Systems. https://doi.org/10.48550/ARXIV.2211.02794

[10] Simon Chu, Emma Shedden, Changjian Zhang, Rômulo Meira-Góes, Gabriel A. Moreno, David Garlan, and Eunsuk Kang. 2023. Runtime Resolution of Feature Interactions through Adaptive Requirement Weakening. In *2023 IEEE/ACM 18th Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. 115–125. https://doi.org/10.1109/SEAMS59076.2023.00025

[11] Frank Ciesinski and Marcus Größer. 2004. On Probabilistic Computation Tree Logic. In *Validation of Stochastic Systems*. https://api.semanticscholar.org/CorpusID:30548386

[12] Nicolas D'Ippolito, Víctor Braberman, Jeff Kramer, Jeff Magee, Daniel Sykes, and Sebastian Uchitel. 2014. Hope for the best, prepare for the worst: multi-tier control for adaptive systems. In *Proceedings of the 36th International Conference on Software Engineering* (Hyderabad, India) *(ICSE 2014)*. Association for Computing Machinery, New York, NY, USA, 688–699. https://doi.org/10.1145/2568225.2568264

[13] Georgios E. Fainekos and George J. Pappas. 2009. Robustness of temporal logic specifications for continuous-time signals. *Theor. Comput. Sci.* 410, 42 (2009), 4262–4291.

[14] Murielle Florins and Jean Vanderdonckt. 2004. Graceful Degradation of User Interfaces as a Design Method for Multiplatform Systems. In *Proceedings of the 9th International Conference on Intelligent User Interfaces* (Funchal, Madeira, Portugal) *(IUI '04)*. Association for Computing Machinery, New York, NY, USA, 140–147. https://doi.org/10.1145/964442.964469

[15] Ivo Friedberg, Kieran Mclaughlin, Paul Smith, and Markus Wurzenberger. 2016. Towards a Resilience Metric Framework for Cyber-Physical Systems. In *International Symposium for ICS & SCADA Cyber Security Research*. https://api.semanticscholar.org/CorpusID:8835415

[16] Pushpak Jagtap, Fardin Abdi, Matthias Rungger, Majid Zamani, and Marco Caccamo. 2020. Software Fault Tolerance for Cyber-Physical Systems via Full System Restart. *ACM Trans. Cyber-Phys. Syst.* 4, 4, Article 47 (aug 2020), 20 pages. https://doi.org/10.1145/3407183

[17] Fanxin Kong, Meng Xu, James Weimer, Oleg Sokolsky, and Insup Lee. 2018. Cyber-Physical System Checkpointing and Recovery. In *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*. 22–31. https://doi.org/10.1109/ICCPS.2018.00011

[18] Nianyu Li, Mingyue Zhang, Jialong Li, Eunsuk Kang, and Kenji Tei. 2023. Preference Adaptation: user satisfaction is all you need!. In *2023 IEEE/ACM 18th Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. 133–144. https://doi.org/10.1109/SEAMS59076.2023.00027

[19] Oded Maler and Dejan Nickovic. 2004. Monitoring Temporal Properties of Continuous Signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, Yassine Lakhnech and Sergio Yovine (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 152–166.

[20] Riccardo Mengacci, Grazia Zambella, Giorgio Grioli, Danilo Caporale, Manuel Catalano, and Antonio Bicchi. 2021. An Open-Source ROS-Gazebo Toolbox for Simulating Robots With Compliant Actuators. *Frontiers in Robotics and AI* 8 (08 2021), 713083. https://doi.org/10.3389/frobt.2021.713083

[21] Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. 2007. MiniZinc: Towards a Standard CP Modelling Language. In *Principles and Practice of Constraint Programming – CP 2007*, Christian Bessière (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 529–543.

[22] Jeroen Ploeg, Elham Semsar-Kazerooni, Guido Lijster, Nathan van de Wouw, and Henk Nijmeijer. 2015. Graceful Degradation of Cooperative Adaptive Cruise Control. *IEEE Transactions on Intelligent Transportation Systems* 16, 1 (2015), 488–497. https://doi.org/10.1109/TITS.2014.2349498

[23] Amir Pnueli. 1977. The Temporal Logic of Programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*. 46–57.

[24] J.B. Rawlings and D.Q. Mayne. 2009. *Model Predictive Control: Theory and Design*. Nob Hill Pub.

[25] Jan Reich, Daniel Hillen, Joshua Frey, Nishanth Laxman, Takehito Ogata, Donato Di Paola, Satoshi Otsuka, and Natsumi Watanabe. 2023. Concept and Metamodel to Support Cross-Domain Safety Analysis for ODD Expansion of Autonomous Systems. In *Computer Safety, Reliability, and Security: 42nd International Conference, SAFECOMP 2023, Toulouse, France, September 20–22, 2023, Proceedings* (Toulouse, France). Springer-Verlag, Berlin, Heidelberg, 165–178. https://doi.org/10.1007/978-3-031-40923-3_13

[26] Divya Aggarwal Rimmi Anand and Vijay Kumar. 2017. A comparative analysis of optimization solvers. *Journal of Statistics and Management Systems* 20, 4 (2017), 623–635. https://doi.org/10.1080/09720510.2017.1395182 arXiv:https://doi.org/10.1080/09720510.2017.1395182

[27] Titos Saridakis. 2009. *Design Patterns for Graceful Degradation*. Springer Berlin Heidelberg, Berlin, Heidelberg, 67–93. https://doi.org/10.1007/978-3-642-10832-7_3

[28] Mary Shaw. 2002. "Self-Healing": Softening Precision to Avoid Brittleness: Position Paper for WOSS '02: Workshop on Self-Healing Systems. In *Proceedings of the First Workshop on Self-Healing Systems* (Charleston, South Carolina) *(WOSS '02)*. Association for Computing Machinery, New York, NY, USA, 111–114. https://doi.org/10.1145/582128.582152

[29] Gustavo Rezende Silva, Juliane Päßler, Jeroen Zwanepol, Elvin Alberts, S. Lizeth Tapia Tarifa, Ilias Gerostathopoulos, Einar Broch Johnsen, and Carlos Hernández Corbato. 2023. SUAVE: An Exemplar for Self-Adaptive Underwater Vehicles. arXiv:2303.09220 [cs.RO]

[30] Thein Than Tun, Amel Bennaceur, and Bashar Nuseibeh. 2021. OASIS: Weakening user obligations for security-critical systems. In *IEEE International Conference on Requirements Engineering*. 113–124.

[31] Priya Venkitakrishnan. 2002. Rollback and Recovery Mechanisms In Distributed Systems. https://api.semanticscholar.org/CorpusID:15179795

[32] Jon Whittle, Pete Sawyer, Nelly Bencomo, Betty Cheng, and Jean-Michel Bruel. 2010. RELAX: A language to address uncertainty in self-adaptive systems requirement. *Requir. Eng.* 15 (06 2010), 177–196. https://doi.org/10.1007/s00766-010-0101-0

[33] Rebekka Wohlrab, Rômulo Meira-góes, and Michael Vierhauser. 2022. Run-Time Adaptation of Quality Attributes for Automated Planning. In *2022 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. 98–105. https://doi.org/10.1145/3524844.3528063

[34] Yuming Zhao, Zhen He, Gelun Li, Yabiao Wang, and Zhigang Li. 2020. Design and Application of a Small ROV Control System Based on ArduSub System. In *2020 IEEE 2nd International Conference on Civil Aviation Safety and Information Technology (ICCASIT*. 585–589. https://doi.org/10.1109/ICCASIT50869.2020.9368667

[35] Sebastián A. Zudaire, Leandro Nahabedian, and Sebastián Uchitel. 2022. Assured Mission Adaptation of UAVs. *ACM Trans. Auton. Adapt. Syst.* 16, 3–4, Article 7 (jul 2022), 27 pages. https://doi.org/10.1145/3513091