System Resilience through Health Monitoring and Reconfiguration

ION MATEI, WIKTOR PIOTROWSKI, ALEXANDRE PEREZ, and JOHAN DE KLEER, PARC, part of SRI International, USA
JORGE TIERNO, WENDY MUNGOVAN, and VANCE TURNEWITSCH, Barnstorm Research Corporation, USA

We demonstrate an end-to-end framework to improve the resilience of man-made systems to unforeseen events. The framework is based on a physics-based digital twin model and three modules tasked with real-time fault diagnosis, prognostics and reconfiguration. The fault diagnosis module uses model-based diagnosis algorithms to detect and isolate faults and generates interventions in the system to disambiguate uncertain diagnosis solutions. We scale up the fault diagnosis algorithm to the required real-time performance through the use of parallelization and surrogate models of the physics-based digital twin. The prognostics module tracks fault progression and trains the online degradation models to compute remaining useful life of system components. In addition, we use the degradation models to assess the impact of the fault progression on the operational requirements. The reconfiguration module uses PDDL-based planning endowed with semantic attachments to adjust the system controls to minimize the fault impact on the system operation. We define a resilience metric and use a fuel system example to demonstrate how the metric improves with our framework.

CCS Concepts: • Computing methodologies → Planning for deterministic actions; Computational control theory; Modeling methodologies; Supervised learning by regression;

Additional Key Words and Phrases: Resilience, diagnosis, prognostics, planning, modeling, machine learning

ACM Reference format:
Ion Matei, Wiktor Piotrowski, Alexandre Perez, Johan de Kleer, Jorge Tierno, Wendy Mungovan, and Vance Turnewitsch. 2024. System Resilience through Health Monitoring and Reconfiguration. ACM Trans. Cyber-Phys. Syst. 8, 1, Article 7 (January 2024), 27 pages.
https://doi.org/10.1145/3631612

1 INTRODUCTION

This article addresses the challenge of greatly improving the resilience of any autonomous cyber-physical system without significant design-time cost. With enough designer effort, any system can be made resilient, but this article demonstrates how to achieve automated resilience in a general...
way with less effort. We illustrate this approach with the task of designing resilient systems for future generations of autonomous naval ships. Most physical systems in use today were designed with three limiting assumptions: (1) the system designer had a severely restricted set of conditions in mind from which the system could recover from (e.g., feedback control systems to maintain stability or direction), (2) human operators are expected to quickly respond to unexpected events, and (3) maintenance technicians are available to repair or modify the system as needed. These limitations made sense given the technologies available to designers at the time. The result of these limitations are designs with limited resilience: lack of ability for the system to adapt in sufficient time to events unexpected by the original designer.

We now live in a world where computation has become exponentially cheaper making it possible to embed sophisticated computation on board. Artificial Intelligence (AI) and Prognostics and Health Management (PHM) algorithms have dramatically increased in capability to the point where we can embed AI systems to perform tasks usually performed by human personnel. To leverage these advances, first, the physical systems must become sensor rich. Second, the physical design must include additional reconfigurable redundancy that the AI can modify, as needed. In our approach, the AI embedded in the system is fully model-based. Given the system is now controlled by an AI, physical systems must be designed differently to take advantage of the new computational capabilities. Some key questions are: (1) where to introduce (or remove) sensors and (2) where to add (or reduce) redundancy. Physical redesign is outside of the scope of this article. However, we envisage a Design Space Exploration approach [37] where both the physical system design and the AI are automatically generated by optimizing a cost function that includes resilience as a metric, as well.

Autonomous systems require the capability to sense discrepancies in their behavior, reason about their causes and adapt to them to reduce the impact on their mission. We focus on system faults as the cause of observed discrepancies. We present an end-to-end framework that uses health monitoring and reconfiguration to detect, reason and adapt to system faults, to ensure system resilience. Autonomous systems are cyber-physical system (CPS) archetypes since they combine hardware components that are controlled by software components, such as the resilience modules we describe in this article. The block diagram of the framework is shown in Figure 1.

Fig. 1. Block diagram of the automatically created embedded AI to ensure resilience.

In our conception of a resilient system, algorithms supporting diagnosis, prognosis and reconfiguration must continually operate as the physical system such as a ship goes about its mission. To maximize resiliency we need those algorithms to be general and not be tied to the specifics of the ship itself. We want to avoid the fatal flaw of handcrafted resilience where the tacit assumptions of the designer limit resiliency. In our framework, at design-time, we are given only the model of the system expressed in a language like Modelica [28]. Our software then constructs all the data structures needed by the run time algorithms to achieve resilience: computational models to encode the Modelica models, surrogate models to improve efficiency, and PDDL models for reconfiguration [27].

At run-time, the system resilience is ensured by three key modules. Health monitoring is implemented through the pervasive diagnosis and prognostics modules. The reconfiguration module is responsible for adapting the system in response to faults. The pervasive diagnosis module uses
sensor measurements and a model of the system to detect and isolate faults. We use a **model-based diagnosis** (MBD) approach, where a model of the system is coupled with an optimization algorithm to reason about system faults. The optimization algorithm estimates system parameters and compares them against their nominal values. A significant challenge in automated diagnosis is ambiguity in diagnosis solutions. We minimize diagnosis uncertainty by designing limited interventions (i.e., sequence of control inputs) that drive the system through a sequence of states that makes the true diagnosis more evident. We ensure scalability to real-time by replacing the physics-based model with a surrogate model endowed with automatic-differentiation constructs. The prognostics module enables the prediction of fault evolution and the estimation of **remaining useful life** (RUL) of components from models and data. The Prognostics algorithm uses data-driven models trained online based on data generated by the diagnosis module to make predictions about the fault progression. Such fault progressions are used to evaluate their impact on the mission objectives. The reconfiguration module computes actuation configurations that respond to fault progressions. We employ PDDL-based planning [27] enhanced with semantic attachments [33] to efficiently compute optimal system configurations. The reconfigurator will mitigate the impact of a fault on the mission objectives if the system is designed with sufficient actuation capacity. We use tool-independent computational representations, namely **Functional Mockup Units** (FMUs) [13] that are integrated into the three modules. FMUs are computational representations of system models developed for simulation purposes.

Since the framework is intended to be deployed on an autonomous ship, scalability of the framework to real-time execution is of utmost importance. Our contributions to address this challenge are: (1) employ parallel execution of processes that compute diagnosis solutions, (2) use surrogate models and optimal control to efficiently compute interventions to reduce diagnosis uncertainty, and (3) improve the performance of the planner via model-specific heuristics, and reducing the complexity of the mixed discrete/continuous planning problem via temporal discretization with a finite time horizon.

**Notations:** We use upper-case to denote random variables \(X\) and lower-case to denote a realization of a random variable \(x\). We use bold letters to denote vectors \(x\). We mark the continuous time dependency by using the notation \(X(t)\) and \(x(t)\) for random processes and time-varying variables, respectively. To represent discrete time dependency, we use the notation \(x(t_k) = x_k\), for time instants \(t_k\). A sequence of variables over time \(\{x_k\}_{k=0}^K\) is denoted by \(x_{0:K}\). We denote the probability distribution function (p.d.f.) of a random variable \(X\) by \(f_X(x)\). We represent the conditional p.d.f. of \(X|Y\) by \(f_{X|Y}(x|y)\). When there is no loss of clarity, to simplify the notation, we will omit the subscript notation of \(f_{X|Y}(x|y)\), that is, we will use \(f(x|y)\). We denote the expectation of the random variable \(X\) by \(E[X]\). Let \(S = \{s_i\}_{i=1}^n\) denote a set of elements. We denote by \(s_{-i}\) the set \(S - \{s_i\}\).

### 2 PROBLEM SETUP

In this section, we describe the model of the physical system to which we apply our approach to resilience, i.e., a fuel system model. In addition, we formalize the diagnosis, prognostics and reconfiguration problems. We conclude with our definition of resilience.

#### 2.1 System Model

The health monitoring and reconfiguration modules use physics-based models to reason about the state of the system. The types of systems we encounter include nonlinearities, discrete and algebraic constraints. The mathematical model describing the behavior of the physical system is given by a (hybrid) **differential algebraic equation** (DAE) of the form

\[
0 = g(X, X, U; P), \quad X(0) = X_0, \tag{1}
\]

\[
Y = h(X, U; P) + V, \tag{2}
\]
where \( X \) is the (stochastic) state of the system, \( U \) is the vector of inputs, \( P \) is the vector of model parameters, and \( Y \) is the vector of output measurements. The outputs are affected by the independent and identically distributed (i.i.d.), additive noise \( V \), assumed Gaussian with zero mean and covariance matrix \( \Sigma_v \). The initial state \( X_0 \) and the vector of parameters \( P \) are vector-valued random variables with known prior distribution \( f_{X_0} \) and \( f_P \), respectively. For example, the p.d.f. of the vector of parameters can be Gaussian, with mean \( \bar{p} \) and covariance matrix \( \Sigma_p \). The vector \( \bar{p} \) can be interpreted as nominal parameters, and matrix \( \Sigma_p \) reflects the uncertainty in the nominal value.

To illustrate the end-to-end health monitoring and reconfiguration framework we constructed a fuel system model, expressed in the Modelica language [28]. The model describes the fuel supply from two tanks to two engines through a series of pipes and valves. This model is a proxy for the fuel system model used in an autonomous ship. We present the Modelica block diagram of the fuel system in Figure 2. There are four valve components for each line that control the mass flow rate of fuel. The two main lines are connected through three cross-line valves, along the fuel lines. They ensure that the fuel can be re-routed from one line to another, as needed. Each valve is controlled by a continuous actuation signal, taking values in the interval \([0, 1]\): a 0 value means that the valve is completely closed, a 1 value means that the valve is completely open. There are a total of 11 control signals operating on the valves. Two fuel pumps move fuel from the tanks to the engines. The volumetric flow rate of each pump is determined by a control signal that sets its angular velocity. We can measure fuel mass flow rates at 8 location along the fuel lines (see Figure 2), and pressure at 4 locations.

The fuel system shown in Figure 2 is affected by two types of faults: leaks and stuck valves. To include faulty behavior in the model, we augment the model with parameterized components that enable fault injection. This approach to fault modeling builds on [59]. The leaks are modeled through specialized components that use (fault) parameters to set the severity of the faults. A 0 value means there is no leak, while a 1 value means the highest leak severity. We have included 4 leak points on each line. The leak components are implemented using a valve connected to a sink component that sets ambient pressure. The fault parameter sets the opening of the valve. The
stuck valve is modeled by setting the valve control signal to a constant value determined by the associated fault parameter. This model is a good use case for three reasons. First, it represents all the main components of a medium size vessel fuel system. It models the flow of fuel between the day tank and the engines, the part of the overall fuel system most likely to cause trouble while underway. (We did omit a set of filters located before feeding the fuel to the engines). Second, it models the typical port/starboard redundancy and crossovers; and the set of valves needed to isolate faulty regions used to mitigate the fault effects until mechanics can access the ship. Finally, this model is indeed representative because it contains some of the most difficult components to model, i.e., fluid dynamics components, which are notoriously brittle. We also tested our approach on the electro-mechanical power system, which included generators, batteries and solar panels under variable load and diverse faults. We found that to be a much easier problem. Our use-case has a number of faults similar to the diagnosis literature. A widely used testbed for fault detection and isolation is the three-tank diagnosis example [62], which incorporates 3 sensor faults and 2 actuator faults. More intricate systems, such as a wind turbine [71], involve 8 faults. In our case, we address 8 leak faults and 10 stuck valve faults. The complexity of our system is significantly larger. Even after the model is translated, meaning it is flattened and simplified, it still comprises 366 time-varying variables, 34 linear system equations, and 85 nonlinear systems of equations.

2.2 Diagnosis

Broadly speaking there are two approaches to the diagnosis problem: MBD and Machine Learning (ML). MBD methods require models and parameters while ML approaches need a large amount of training data and extensive feature engineering. Since in our case we do have a system model but no failure data, we use MBD. In MBD we use a model of the system, nominal values of the parameters of the model and values of some of its inputs and outputs. The diagnosis engine determines from only this information the presence of a fault and isolates it. MBD approaches were proposed independently by the AI [22] and control [34, 53] communities. Traditional MBD approaches in the control communities include filters (e.g., Kalman filter [36], particle filter [5]), or optimization based-techniques that estimate parameters whose deviation from their nominal values indicate the presence of a fault. Both these methods rely on model simulations.

We define $\mathcal{F} = \{1, 2, \ldots, N\}$ as a set of ordered indices for the fault modes, with associated scalar parameters $p_i$. We denote the vector of all fault parameters by $p = [p_1, \ldots, p_N]$, and we assume it has a nominal value $\bar{p}$. Under the single fault mode scenario, we model a fault mode as an event $\omega_i$ defined as a discrepancy between a model parameter and its nominal value. In particular, $\omega_i = \{ |p_i - \bar{p}_i| > \epsilon_i, \bar{p}_i = \bar{p}_{-i} \}$, where $\epsilon_i$ is a positive scalar. The scalar $\epsilon_i$ depends on the measurement noise and the sensitivity of the behavior of the system to changes in the parameter $p_i$. The fault magnitude is determined by estimating the value of the system parameter $p_i$. Given a sequence of input and output measurements over the time horizon $\tau$, the diagnosis problem consists of computing the conditional probability $P(\omega_i|y_{0:\tau}, u_{0:\tau})$, together with the estimation of the parameter $p_i$.

The fault parameter estimation is implemented using optimization algorithms that use model simulations to update the fault parameters so that the simulated outputs match the measured outputs. We segment the output time series into non-overlapping windows, and for each window we update the fault parameters by solving least square problems. The window size depends on the time constants of the system dynamics. The size of the window must be correlated with the time needed by the optimization algorithm to generate a solution. If such a time is larger than the window size, we incur delays in generating diagnosis solutions. As an alternative to the optimization-based approach, we could use filtering-based techniques, by considering the fault parameters as
states. Since the fuel system model is nonlinear, the linear Kalman filter cannot be applied directly. The extended Kalman filter [47] requires artifacts not readily available such as the Jacobians of the state and measurements maps. Sampling-based filtering techniques, such as the particle filter [5], are computationally intensive since they require many sample points to propagate an accurate distribution of the state. The unscented Kalman filter [35] is a compromise between accuracy and computational effort and uses a set of sigma-points to approximate the distribution of the state. Qualitative diagnosis algorithms, such as analytical redundant relations (ARRs) [60] can also be applied. They have the advantage that they do not require fault models, but they typically need more sensors to generate unambiguous diagnosis solutions. Since in the case of the fuel system the physics behind leaks is well understood, we opted for a diagnosis solution based on fault models.

2.3 Prognostics

Prognostics is the science of predicting the health condition of a system or component using information about the past usage, current state, and future conditions. The time until a component reaches a failure beyond which the system can no longer be used to meet desired performance is defined as RUL. Similar to the diagnosis problem, data-driven and model-based approaches are used for prognostics. Data-driven prognostics use pattern recognition and ML algorithms and models to detect changes in system states. Data driven models used for prognostics include autoregressive (AR) models, various types of neural networks (NNs) or neural fuzzy (NF) systems.

Model-based approaches use physics-based degradation models such as braking system wear based on the Archer’s law [2]), fatigue life model for ball bearings [70], crack growth model [51], or stochastic defect-propagation model [40]. While physics-based degradation models exist for a number of applications, there is no formal approach to determine such models for all systems. More details about the state-of-the-art in prognostics can be found in Section 7. We combine a physics-based model of a system with a data-driven model that learns online the degradation model and extrapolate the RUL. The data-driven model is trained based on the outputs generated by the diagnosis module. The diagnosis algorithm produces a sequence of parameter predictions \( \{\hat{p}_{k}\}_{k=0}^{\tau} \), where \( \tau \) denotes the present time. We use this sequence to learn a dynamical model,

\[
\hat{p}_{k} = G(\hat{p}_{k-1}, \ldots, \hat{p}_{k-n}, e_{k}, \ldots, e_{k-m}),
\]

for positive integers \( n, m \), where \( e_{k} \) stands for uncertainties and noise. This dynamics represents the degradation model. The RUL metric is defined in terms of component parameters or component variables. A component is declared failed when its parameter \( p_{i} \) exceeds a range. For some components only the maximum or minimum of the range matters. In the case where a fault occurs if some maximum is passed, we define the RUL for component \( i \) as the random variable \( K_{RUL}^{i} = \min_{k}(\hat{p}_{k,i} > p_{\text{max},i}) \), where \( \hat{p}_{k,i} \) follows the stochastic dynamics predicted by the degradation model (3).

2.4 Reconfiguration

The Reconfiguration module is tasked with mitigating the fault’s impact on the mission objectives, and is achieved by adjusting the system’s actuation controls. Reconfiguration uses as inputs information from the Diagnosis and Prognostics modules. Given mission specification, the reconfiguration module makes strategic adjustments based on the system’s current and future condition, as well as its objectives and capabilities. Our approach to reconfiguration is AI Planning that poses the problem as a combinatorial search task.

Planning models are typically defined using the Planning Domain Definition Language (PDDL) [46], a standardized modeling paradigm. The characteristics of the modeled system dictate the appropriate level of PDDL needed to accurately capture important features. PDDL+ [26]
was designed specifically to represent hybrid (mixed discrete-continuous) systems and introduced the concepts of continuous flows (processes) and discrete mode switches (events) to planning domains. As a generic AI planning paradigm, various reconfiguration scenarios can be encoded in PDDL+ and general-purpose, off-the-shelf PDDL+ planners can be used to solve the resulting problems. Various approaches to solving problems set in hybrid domains have been proposed in planning [17, 19], as well as control and model-checking [14, 18]. However, all of these techniques are limited in scale, model features, or dynamics. To date, the only viable approach to planning in realistic hybrid domains is planning via discretization [23, 55]. More formally, planning via discretization paradigm sees PDDL+ models translated into Finite State Temporal System (FSTS), an automation which defines the state space and state transitions. An FSTS is a tuple $(S, s_0, \mathcal{A}, \mathcal{D}, F, T)$ where $S$ is a finite set of states (i.e., configurations the system can be in), $s_0 \in S$ the initial state, $\mathcal{A}$ is the set of actions that modify the states, $\mathcal{D} = \{0, \Delta t\}$ where $\Delta t$ is the discretized time step, $F : S \times \mathcal{A} \times \mathcal{D} \rightarrow S$ is the transition function generating successor states $s'$ (i.e., $F(s, a, d) = s'$), and $T$ is the finite temporal horizon. A planning task focuses on finding a state $s_G$ that satisfies goal conditions, within the FSTS. A solution to a PDDL+ planning problem in an FSTS is a plan, i.e., a trajectory or sequence of state, action, and duration tuples (i.e., $(s_i, a_i, d_i)$) that starts with the initial state $s_0$ and ends with the goal state $s_G$. The vast majority of real-world scenarios are inherently hybrid in nature. We cast reconfiguration as a hybrid planning problem, encode the system in PDDL+, and use a general-purpose planner to generate a reconfiguration plan.

### 2.5 Resilience

A qualitative definition of resilience inspired from power systems [50, 68] is: a system is resilient if it is able to withstand, adapt and recover from unforeseen events. A more general, quantitative list of resilience metrics is proposed in Ref. [49], where resilience is defined with respect to abilities of resilience: anticipation, absorption, adaptation, and rapid recovery. The simplest one is defined as the fraction between the true functionality as the system goes through the four phases, and the nominal functionality, i.e., $(\int_0^{T_f} F(t) dt) / (\int_0^{T_f} TF(t) dt)$, where $TF(t)$ represents the expected normal functionality of the system, $F(t)$ is the true functionality of the system. A challenge with this type of metric is that it is defined with respect to the time $T_f$ at which the system recovers its functionality. However, a system may never recover its full functionality, yet still be able to perform its mission. In addition, the system may be in a continuous adaptation phase, as it continuously adapts and responds to faults; faults that evolve over time, as well.

Remaining in the same spirit of resilience with respect to functionality, we define the system resilience with respect to a set of system requirements. In the case of the fuel system, the system requirements are (1) to ensure a prescribed mass flow rate at each of the two engines, and (2) to minimize the fuel losses, over some time horizon. The system should withstand the effect of faults, by adapting its operation (i.e., acting on the pumps and valves) to recover from disruptions in supplying fuel to the engines. Let $y^d$ = $[y^{d}_1, y^{d}_2]$ define the required mass flow rates at the engines, and let $y^{in}_k$ = $[y_{i,n,k}^{in}, y_{k,n}^{in}]$, $y^{out}_k$ = $[y_{k,u,k}^{out}, y_{k,o,k}^{out}]$ define the measured fuel mass flow rates at the tanks and engines, respectively. We bound the amount of fuel the tanks can supply, i.e., $\sum_{k=0}^{T} y^{in}_{i,k} \leq \beta_i$, where index $i$ designates tank $i$, and $\beta_i$ are positive scalars proportional to the tank volumes. We quantitatively define the fuel system resilience over a time horizon $T$ as the convex combination of two metrics that measure the deviation from the system requirements, i.e., $R_T = 1 - \alpha_1 J^1_T - \alpha_2 J^2_T$, where $\alpha_1 \geq 0$ and $\sum \alpha_i = 1$, with $J^1_T = \frac{1}{(T+1)(y^d_1+y^d_2)} \sum_{i,k=0}^{T} |y^d_i - y^{out}_{i,k}|$, and $J^2_T = \frac{1}{\beta_1 + \beta_2} \sum_{i,k=0}^{T} |y^{out}_{i,k} - y^{in}_{i,k}|$. The term $J^1_T$ measures the deviations from the required mass flow rates, while $J^2_T$ measures the fuel losses during the system operation over the time horizon $T$. Scalars $\alpha_1$ and $\alpha_2$ shift focus between the two requirements. In nominal conditions, the terms
$J_T^1$ and $J_T^2$ are zero, hence the system resilience is $R_T = 1$. In the worst case, all fuel is lost and the system resilience is zero, i.e., $R_T = 0$. In Section 6, we will evaluate the resilience metric under different scenarios that demonstrate the effect of our proposed framework on the system resilience.

3 PERVERSIVE DIAGNOSIS

In this section, we describe our optimization-based approach to MBD that includes the maximum likelihood estimation, the numerical optimization approach, and the optimal control for disambiguation.

3.1 Maximum Likelihood Estimator

We represent the system model (1)-(2) as an input-output model $Y_k = h(U_k; P) + V_k$, where we use upper cases to emphasize the stochastic nature of the system model. Given a sequence of output measurements $\{y\}_{k=0}^{\tau}$ and inputs $\{u\}_{k=0}^{\tau}$, we define the diagnosis problem as the computation of the p.d.f. $f(p|y_{0:\tau}, u_{0:\tau})$, and the optimal estimator is given by $\hat{p}_\tau = E[P|Y_{0:\tau}, U_{0:\tau}]$. The maximum likelihood estimator is the solution of the optimization problem $\max_p f(p|y_{0:\tau}, u_{0:\tau})$, whose solution is $\hat{p}_\tau$, when $f(p|y_{0:\tau}, u_{0:\tau})$ is a Gaussian distribution. Using a Bayesian approach, the $f(p|y_{0:\tau}, u_{0:\tau})$ can be expressed in terms of $f(p|y_{0:\tau}, u_{0:\tau})$ and $f(p)$, the prior distribution of the vector of parameters. Under the input-output dynamics, the conditional p.d.f. $f(y_k|u_k, p)$ is a Gaussian p.d.f. with mean $\hat{y}_k = h(u_k; p)$ and covariance matrix $\Sigma_v$. The mean vector $\hat{y}_k$ is generated by simulating the system model, given the input $u_k$ and the vector of parameters $p$. Under the additive Gaussian measurement noise assumption, we compute the solution of the maximum likelihood estimator by solving the following optimization problem:

$$\min_p \sum_{k=0}^{\tau} (y_k - \hat{y}_k)^T \Sigma_v^{-1} (y_k - \hat{y}_k) - \log f(p). \quad (4)$$

There is no guarantee that problem (4) has a unique solution. The exact type of solution depends on the number of sensors and their placement. To address the solution non-uniqueness efficiently, we use the single fault scenario (even if we know there are multiple faults): we assume that at the diagnosis time only one fault is responsible for the abnormal observation. Thus, we compute $N$ parameter estimates corresponding to each fault scenario, by solving the problem (4), where $p$ corresponds to a single parameter $p_i$.

The optimization problem (4) can be further simplified by assuming high uncertainty in the parameter vector $p_i$ that can be expressed through a uniform distribution. Consequently, the term $\log f(p_i)$ in the loss function of (4) can be neglected. For non-linear systems, computing the closed-form solution of $f(p_i|y_{0:\tau}, u_{0:\tau})$ is intractable due to the denominator. We can approximate the p.d.f. $f(p_i|y_{0:\tau}, u_{0:\tau})$ by taking advantage of the parameter samples that are generated as part of the optimization problem (4). At each iteration of the algorithm, the model is simulated at the current parameter value, and the model outputs are generated. Then, we have the following approximation:

$$f(p_i|y_{0:\tau}, u_{0:\tau}) \approx \frac{\sum_{m=1}^{M} \delta(p_i - p_{i}^{m}) \prod_{k=0}^{\tau} f(y_k|u_k, p_{i}^{m})}{\sum_{m=1}^{M} \prod_{k=0}^{\tau} f(y_k|u_k, p_{i}^{m})},$$

and the fault probability $\tilde{q}_i = P(|p_i - \hat{p}_i| > \epsilon_i)$ is given by $\tilde{q}_i = 1 - (\sum_{m=1}^{M} \prod_{k=0}^{\tau} f(y_k|u_k, p_{i}^{m}))/ (\sum_{m=1}^{M} \prod_{k=0}^{\tau} f(y_k|u_k, p_{i}^{m})), where $M_i = \{m|p_{i}^{m} \in [\hat{p}_i - \epsilon_i, \hat{p}_i + \epsilon_i]\}$. We compute the normalized fault probability as $q_i = \tilde{q}_i/(\sum_{j=1}^{M} \tilde{q}_j)$.

Note that if the measurement noise is not Gaussian, the loss function in Equation (4) would not be a mean square error (MSE) loss but would depend on the particular distribution of the noise.
In turn, this would prevent the use a nonlinear least square algorithms that require the MSE loss. However, we can still use the MSE, but we would lose the maximum likelihood interpretation of the solution, resulting in suboptimal parameter estimates.

### 3.2 Optimization-Based System Diagnosis

We use optimization algorithms to find an assignment to the vector of fault parameters $p$ such that the simulated output is consistent with the observations. In a single fault scenario only one fault can happen at a time. As such at most one off-nominal fault value $p_i$ is estimated in $p = [\bar{p}_1, \ldots, \bar{p}_{i-1}, \bar{p}_i, \bar{p}_{i+1}, \ldots, \bar{p}_N]$. (In the multiple fault scenario, multiple $p_i$ are estimated). We separately compute the most consistent value for every $i$th fault parameter:

$$l_i = \min_{p_i} \sum_{k=0}^{r} \left( y_k - \hat{y}_k^i \right)^2,$$

where $\hat{y}_k^i$ is the output of the system model simulation using the off-nominal fault value $p_i$. As Equation (5) depicts, we use the mean squared error between observation and simulation as the loss function for optimization. Such a cost function results from Equation (4) when assuming independent measurement noise, with identical variances. We only report $p_i$ as a diagnosis if $|p_i - \bar{p}_i| > \varepsilon_i$.

The diagnosis are ranked based on the empirical probabilities $q_i$ that are a function of $l_i$, and the most likely diagnosis is $i^* = \arg\max_i q_i$.

Our system model simulation pipeline adheres to the Functional Mockup Interface standard [13]. Specifically, our diagnosis engine takes as input FMUs, which are self-contained, natively executable containers allowing for the exchange and simulation of dynamic models. Multiple systems modeling environments—such as OpenModelica, Dymola, among others—support exporting models FMU objects. While forward simulations of FMUs are very efficient as these objects can natively target a specific computing architecture, they do not natively posses constructs enabling AD. As such, the black-box\(^1\) nature of our simulation setup limits us to using only gradient-free optimization approaches to estimate the fault vector. Our implementation leverages Scipy’s [67] gradient-free optimization algorithms, namely Powell.

The single fault scenario allows us to parallelize the execution of Equation (5) across available computational resources, as each fault parameter estimation procedure is independent from each other. We implemented a multiprocessing architecture capable of running multiple optimization procedures across a shared pool of processes, effectively speeding up diagnosis. Each optimization procedure contains its own FMU object handle which it uses for simulation and loss function computation.

We applied the parameter estimation based diagnosis algorithm for detecting leaks in the transportation of fuel from the tanks to the engines. We considered two leak magnitudes medium and high, which correspond to the fault parameter values 0.5 and 0.75, respectively (a fault value 1 means that all fluid is lost). We tested the diagnosis algorithm for detecting leaks at eight locations, and computed the fault probability of each leak location. We used the Modelica model to generate data for each possible leak location and for the two fault magnitudes. We kept the same exogenous inputs (i.e., pump reference signals, valve openings) for each of the fault scenario. We recorded the mass flow rates at the eight measurement locations, to which measurement noise was added. We use these output measurements in the diagnosis algorithm. The diagnosis results are shown in Tables 1 and 2. The table entries are fault probability, parameter estimate tuples. A perfect diagnosis result corresponds to the probability one on the main diagonal and corresponding correct parameter estimate. For example, for Table 1, a perfect result is given by (1, 0.5) on the main diagonal. For

\(^1\)The simulator does not have direct access to the model equations. It can only query the FMU.
In a perfect diagnosis the diagonal values should correspond to (1,0.5).

medium leaks, while there is some uncertainty in the fault likelihood, the probabilities corresponding to the ground truth are indeed dominant: hence, correct diagnosis are made. For the high leak case, we can no longer distinguish between leak_fault_3 and leak_fault_5, and leak_fault_4 and leak_fault_6. This result is not surprising: since due to the magnitude of the leak, all fuel is lost at the leak location. But since, there is no sensor in between the leak_fault_3 and leak_fault_5, the diagnosis algorithm cannot pinpoint the exact location. Hence, the diagnosis is ambiguous. In the next section, we show how we can solve the diagnosis disambiguation problem. The parameter estimates must be read in concert with the fault probabilities, i.e., we are interested in the parameter estimates for faults that have high probabilities. On a Dell Precision 3640 Tower (Intel Xeon CPU @ 4.1 GHz, 6 cores, 64 GB), the average time to generate a diagnosis is 1.5 sec.

### 3.3 Diagnosis Disambiguation

In an ideal scenario, the sensor measurements include sufficient information to distinguish among the fault modes. However, this is not always the case. Given the two mass flow rate sensors (massFlowSensor_4 and massFlowSensor_6) and the nominal inputs to the valves and the pumps, the diagnosis engine is unable to provide a precise diagnosis when the leaks are significant. We have an ambiguous diagnosis solution. We define the ambiguity as two or more faults having similar probabilities. This result is not surprising since there is no sensor between the two leaks. We can address this weakness in two ways: (1) add more sensors, or (2) elicit richer information from sensors through targeted system excitation. Since the first option entails changing the physical system, we use the second approach: we design valve and pump inputs so that when applied to the physical system, they generate sufficient information in the measurements. The new information is used for diagnosis disambiguation. The diagnosis result generates fault probabilities $q_i$ and fault parameter estimations $\hat{p}_i$. Assume $\mathcal{A} \subset \mathcal{F}$ is the set of ambiguous diagnoses, with corresponding fault parameter estimates $\{\hat{p}_i\}_{i \in \mathcal{A}}$. Our objective is to design a sequence of inputs $\{u_k\}_{k=0}^{r}$ that maximizes the Euclidean distance between the system outputs that correspond to the ambiguous fault modes. We achieve this objective by solving the following optimization problem:

$$\min_{u_0,\ldots,u_r} \sum_{i=1}^{r} \sum_{k=0}^{r} \sum_{i=1}^{r} \left\| \hat{y}_k^i - \hat{y}_k^j \right\|^2,$$

subject to:

$$\hat{y}_k^i = h(u_k; \hat{p}_i), i \in \mathcal{A}, u_k \in \mathcal{U}, \forall k \in \{0, \ldots, r\},$$

In perfect diagnosis the diagonal values should correspond to (1,0.5).
where $\mathcal{U}$ is a set constraining the inputs that can be applied to the system. For example, in the case of the fuel system model, the valve inputs are constrained to take values in the interval $[0, 1]$. At a first glance, it may appear that the choice for the disambiguation cost function is arbitrary; it is not. We recall that the key term for computing $f(p_i|y_{0:T}, u_{0:T})$ is the product of conditional p.d.f.s $\prod_{k=0}^T f(y_k|u_k, p_i)$. To evaluate each conditional p.d.f. $f(y_k|u_k, p_i)$, we need to evaluate the quadratic term $(y_k - \hat{y}_k)^T \Sigma^{-1}_u (y_k - \hat{y}_k)$. Let $i$ be the true fault mode, with $y_k = \hat{y}_k + u_k$ the output measurements expressed in term of the simulated output $\hat{y}_k$ and a realization of the measurement noise $u_k$. For each ambiguous fault mode $j$, the quadratic expression $(y_k - \hat{y}_k)^T \Sigma^{-1}_u (y_k - \hat{y}_k)$ becomes $(\hat{y}_k + u_k - \hat{y}_k)^T \Sigma^{-1}_u (\hat{y}_k + u_k - \hat{y}_k)$. For the ground truth case $i = j$, the previous expression becomes $u_k^T \Sigma^{-1}_u u_k$, which is the smallest quantity that we can get under the additive noise assumption. The smaller the quantity $(y_k - \hat{y}_k)^T \Sigma^{-1}_u (y_k - \hat{y}_k)$ is for all $k$, the closer to 1 the conditional p.d.f. $f(p_i|y_{0:T}, u_{0:T})$ becomes. Inversely, the larger $(y_k - \hat{y}_k)^T \Sigma^{-1}_u (y_k - \hat{y}_k)$ for all $k$ becomes, the closer to zero $f(p_j|y_{0:T}, u_{0:T})$ gets. Hence by solving (6), we in fact maximize the conditional p.d.f. $f(p_i|y_{0:T}, u_{0:T})$ for the ground truth fault mode $i$, while in all the other fault modes $j$, $f(p_j|y_{0:T}, u_{0:T})$ becomes smaller. The disambiguation problem minimizes the diagnosis uncertainty. The information theory entropy, defined as $-\sum_{i \in A} q_i \log q_i$, where $q_i$ are the probabilities of the ambiguous fault modes, is a metric that measures the diagnosis uncertainty. Indeed, solving (6) brings the probabilities $q_i$ closer to zero or one, hence minimizing the entropy.

3.4 Improving the Scalability of the Diagnosis Disambiguation Algorithm

Solving the diagnosis disambiguation problem involves solving an optimization problem, where the optimization variables are control inputs over a time horizon. When solving optimization problems, gradient-free algorithms scale poorly with the number of optimization variables. The reason we are forced to use gradient-free methods is that the FMU representation of the physics-based model is a black box from the optimization perspective. Numerical approximation of gradients does not solve the problem: we would have to simulate the FMU for at least the number of the optimization variables. To use gradient-based optimization methods, we have to open the model. We alleviate the gradient computation problem by constructing approximation models, known as surrogate models. Such models are emulators that mimic the behavior of the original model while enabling the evaluation of gradients at a cheaper computational cost. Surrogate models are constructed using a data-driven approach and focus on the input-output behavior, disregarding the inner working of the simulation code. To generate AD endow surrogate models, we use Pytorch [52]. Thus, we can use gradient-based algorithms that, unlike gradient free algorithm, scale linearly with the number of optimization variables.

We trained a surrogate model using a NN with two hidden layers of size 300, with tanh as activation function. The surrogate model has as inputs the exogenous inputs and parameters of the physics-based model, and as outputs the sensor measurements of the physics-based model. The training data is generated by simulating the physics-based model. For a large number of inputs and outputs, we have a combinatorial explosion in the size of the training data. However, the training data generation is done off-line, hence time constraints are less relevant. We generated more than 500k training data samples, where the size of the input and the output is 21 and 8, respectively. The model simulations were done using the FMU [13] representation of the physics-based model, integrated into Python code. The input consists of valve and pump control signals, and the outputs are represented by mass flow rates. The inputs were randomly drawn from their domain of definition, using the uniform distribution. Since the surrogate model accepts parameters of the physics-based model as inputs, we can simulate faults. We trained the model using Adam algorithm, with a step
size 0.001. We used the MSE loss function and all other hyper-parameters were left at their default values. The loss function was minimized at MSE = 0.0001.

The goal is to solve (6) and compute actions that yield measurements that distinguish amongst ambiguous diagnoses. We recall that using the 8 mass flow rate sensors, the diagnosis algorithm cannot distinguish between faults 3 and 5, under the high leak scenario. We solved the disambiguation optimization problem (6), where \( \mathcal{A} = \{3, 5\} \), \( \hat{p}_i = 0.75 \), for \( i \in \{3, 5\} \), \( \hat{h} \) is the NN based model, and \( \mathcal{U} = [0, 1]^{11} \times [-5, 5]^2 \) defining the constraint set for the valve and pump inputs. Such bound constraints were implemented in Pytorch through clipping. For comparison, we solved (6) using both the Pytorch-based Adam algorithm and Scipy-based Powell algorithm, for various values. The Pytorch-based implementation was 30 times faster than the Powell-based implementation. The difference in efficiency comes from two reasons: (1) Pytorch implementation used batch execution to evaluate the model at multiple inputs as determined by \( \tau \), and (2) the gradient of the loss function is generated through AD. Using the learned actions, we applied again the diagnosis algorithm to the faulty fuel system resulting in the isolation of the correct faults with probability one. In addition, the leak parameters are correctly estimated, as being around 0.75.

4 PROGNOSTICS

The prognostics module is responsible for (1) constructing a progression/degradation model of diagnosed faults, and (2) estimating the RUL of the faulted component. Our approach to prognostics is hybrid: we combine a physics-based model of the system with a data-driven model of degradation. The data-driven degradation model uses diagnosis results to update its parameters after each diagnosis. Extrapolations of the degradation model combined with simulations of the system model enable the assessment of future health of the components and the evaluation of system reliability metrics (e.g., whether or not the mission objectives are achieved).

We model the degradation of a component using a dynamical model for parameter evolution that is trained online. In particular, for each fault parameter \( p_i \) we represent its changes over time using Non-linear Autoregressive Models with Moving Average and Exogenous Input (NARMAX [12])

\[
p_k = g(p_{k-1}, \ldots, p_{k-n_p}, e_{k-1}, \ldots, e_{k-n_e}) + e_k,
\]

where \( p_k \) and \( e_k \) are the fault parameter and the noise. The noise represents the uncertainties in the model. The positive integer \( n_p \) and \( n_e \) are the maximum delays for the fault parameter and noise, respectively. There are several nonlinear function representations for \( g \): NNs, fuzzy logic-based models, radial basis functions, wavelet basis, polynomial basis or generalized additive model. In this work we represent the map \( g \) using polynomial basis, i.e.,

\[
p_k = \Theta_0 + \sum_i \Theta_i^p p_{k-i} + \sum_j \Theta_j^e e_{k-j} + \sum_{i,j} \Theta_{ij}^{pe} p_{k-i} e_{k-j} + \sum_{i,j} \Theta_{ij}^{pe} p_{k-i} p_{k-j} + \ldots,
\]

where \( \Theta_0, \Theta_i^p, \Theta_j^e, \Theta_{ij}^{pe} \) and \( \Theta_{ij}^{pe} \) are constant parameters that have to be learned. Depending how many basis terms we choose in the representation, NARMAX models can become fairly complex. Yet, NARMAX models are popular because they can represent complex systems with sparse nonlinear representations. They select the basis terms that best explain the observations using robust algorithms for model structure selection. Figure 3 depicts an example prognosis scenario using a simulated fault progression. In blue, we show example diagnoses generated every second over a 100 second period. Such diagnoses are ingested by the prognostics module every time the diagnosis module finishes its computation, prompting it to update its fault progression model. The plots to the right of Figure 3 show the computed fault progression prediction for the leak 5 fault parameter when the prognostics module has diagnostic information of the first 20 seconds and 70 seconds, respectively. With more data from diagnostics, our NARMAX model is able to more accurately track the actual fault progression.

Alongside diagnoses, the prognostics module also takes as input a description of end of life for every possible fault. This description can either be some fixed threshold for the faulted parameter,
or a boolean function encoding the relationship between simulation outputs and mission objectives. In the example depicted in Figure 3, we have set an end-of-life threshold for the leak 5 faulted parameter of 0.8. Coupled with our degradation model, we can then determine the RUL of the faulted component by computing when the threshold will be reached (or, given a function encoding mission objectives, when the system can no longer achieve such objectives). We show the RUL estimates in red. As the fault magnitude (depicted in blue) increases, our RUL estimates more closely track the ground truth, depicted in the plot as a dashed line.

5 RECONFIGURATION

The reconfiguration module is tasked with adjusting the settings of a system to mitigate the detrimental effects of present faults. Defining a realistic dynamical system requires an expressive language that encompasses a wide range of features, functions, and behaviors. We cast reconfiguration as an automated planning problem and define it in PDDL+ [26], a highly expressive extension of the standardized planning modeling language [46]. PDDL+ enables accurate modeling of complex hybrid systems as planning domains, including various constraints, goals, and features. Our system merges the data from its diagnosis and prognosis modules, and the information on the structure and function of the target system, into a holistic planning model that accurately reflects the real-world scenario.

A high-level overview of the reconfiguration approach is depicted in Figure 4. The PDDL+ model contains the information passed on from other modules. Diagnosis and prognosis modules provide information on the state and temporal evolution of the active fault. This information is encoded into the PDDL+ domain as temporal activity (i.e., processes and events) that update fault information according to the prognosis predictions.

The mission specification and objectives provide the planning goal conditions and global problem constraints that any valid solution needs to satisfy. A solution to a reconfiguration task is a plan that ends in a state where all goal conditions are satisfied. In the fuel system scenario, the goal condition is reached after the system has been continuously operational for a given duration. Constraints are normally enforced by PDDL+ events which are automatically triggered once a constraint is violated ("must-happen" behavior). Effects of such events can irreversibly falsify some goal conditions, forcing the planner into exploring alternative solutions. In the fuel system,
examples of constraints are maintaining a sufficient engine mass flow rates to (Figure 5), and not exceeding fuel reserves limited by tank capacity.

Finally, the PDDL+ domain’s system dynamics and action model is defined based on the structure of the target system model (e.g., defined in Modelica). Actions are defined for each reconfigurable component of the system, along with accompanying exogenous activity constructs (i.e., events and processes that are triggered by executed actions). The resulting composable PDDL+ model is then passed into our automated planner. Based on the composition and dynamics of the PDDL+ model, the planner decides if and when to adjust the reconfigurable components of the system to complete the mission objectives while avoiding violating the system’s constraints.

Expressive planning models ensure accurate representation of complex systems, though the resulting planning problems belong to the undecidable class and are notoriously challenging to solve. To tackle such problems, we used Nyx, a novel Python-based lightweight PDDL+ planner. Nyx uses a forward-chaining heuristic search to traverse the state space in pursuit of a valid plan (sequence of temporal actions achieving reconfiguration objectives). Nyx mitigates some of the challenges of hybrid planning by exploiting a planning-via-discretization approach where the continuous dynamics of a planning model are approximated using a uniform time-step and step-functions. This requires a fine balance between finding a discretization quantum fine enough to enable finding meaningful solutions, and coarse enough to maintain reasonable efficiency of the search. Even in the discretized setting, the complexity of PDDL+ problems can be overwhelming for planners, increasing exponentially with time discretization and the number of grounded actions, and often resulting in state-space explosion. We use a resilience-focused model-specific heuristic to efficiently traverse the state space. The Nyx planner exploits an anytime search algorithm. Instead of terminating after finding the first-encountered goal state, Nyx returns all solutions found within a given run-time budget. The generated set of solutions are ranked according to the relevant metric (e.g., minimize leaked fuel).

To circumvent limitations on representing complex dynamics in PDDL+, the Nyx planner is equipped with semantic attachments [24], i.e., external functions to which some or all of the model’s continuous dynamics are delegated. In the case of the fuel system, we delegate the fluid dynamics calculations to an external function in the form of an attached FMU. We pass the planning
Table 3. Planning Problem Setup Common Across Both Scenarios

| Type          | Description                                                                                                                                 |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| **Goal**      | - deliver 30 gallons of fuel per hour to each engine for a continuous period of 10 hours.                                                 |
| **Actions**   | - adjust the opening of valves (11 count). Opening interval 1 - open/close only.  
- increase/decrease speed of pumps (2 count). Adjustment interval +/- 1.                |
| **Time**      | - discretized time step = 1 hour  
- time horizon = 10 hours                                                                                                                        |
| **Constraints** | - massflow at engine ±30% of nominal reference point                                                                                                                                 |
| **Initial State** | - Cross-valves closed (valves 8, 15, 16)  
- All other valves open (valves 6, 7, and 9-14)  
- Nominal pump speed = 3.035  
- Allowed pump speed range [0, 5] |

state (containing the valve and pump settings) as inputs to the FMU and then simulate it over the duration of the planner’s discretized time-step $\Delta t$. After the simulation, the extracted values are passed back to the planner and used to update the values of the state variables. During the search phase, whenever a continuous change takes place in the system, a simulation is performed using the external semantic attachment, all other changes are assumed to be discrete (i.e., mode switches). The search cycle as well as the external function calls are outlined in the PDDL+ Planning box in the diagram depiction of the Reconfiguration Module (Figure 4).

A solution to the fuel system reconfiguration is a sequence of timed actions which adjust the positions of valves and pump speeds, such that the system is operational for required time window. The plan specifies when to change the positions and settings of given reconfigurable components such that mission objectives are achieved.

### 5.1 Experimental Results

We present results for the reconfiguration of the fuel system in two double fault scenarios. In the first scenario, we consider a valve stuck open and a leaky pipe (that is static in intensity), where the mission requires immediate isolation of the fault and limiting the fuel loss. The second scenario is more complex: a valve stuck closed and a leaky pipe which worsens over time. They are set up in such a fashion that not using the faulty components would prevent achieving mission objectives. Thus the reconfiguration module must reason with and use the faulted components. Table 3 shows the PDDL+ problem setup used for addressing both scenarios.

**Scenario 1: Static Leak** includes two faults: valve 11 stuck open, and a leaky pipe 6, with constant intensity of 0.7 (see Figure 2 for valve and leak locations). This scenario requires immediately isolating the leak, forcing the planner to reconfigure the system at the beginning of the mission to minimize fuel loss. The fuel tanks hold a total of 630 gallons, which is just enough to run both engines at 30 Gal/hr for 10 hours (with an additional 30 Gal reserve).

Figure 6 shows a visual representation of the reconfigured system in scenario 1, according to the solution found by the PDDL+ planner (shown in Figure 8). The reconfiguration module isolates the leaky pipe 6 by closing the surrounding valves 10, 12 and 15, followed by rerouting the fuel around the fault by opening cross valve 8 and increasing the speed of the pumps to deliver the fuel at a sufficient rate to both engines. All reconfiguration actions are executed simultaneously as soon as possible, i.e., at time $t = 0.0$, to preempt any further fuel loss.

A comparison of the results between the reconfigured system and the unchanged system shows a stark difference, and is described in Table 4. Without reconfiguration, the faulted pipeline fails to deliver any fuel to the engine, leaking 300 Gal of fuel into the vessel in approximately 3.5 hours. On the other hand, the reconfigured system delivers 95.51% of fuel to the engines compared to an

---

2For clarity, the information on the fault and its progression can be explicitly encoded in the PDDL+ domain or obscured inside the semantic attachment.
Fig. 6. Scenario 1: reconfigured system (green circles represent valves that were opened due to reconfiguration, while the red circles represent valves that were closed. Orange rectangles denote faulted components).

Fig. 7. Reconfigured and unchanged system fuel usage for Scenario 1.

(a) Cumulative fuel drawn from the tanks. (b) Cumulative fuel fed to the engines.

Fig. 8. Reconfiguration plan for the fuel system in Scenario 1 (duration = 10 hours). The leading numbers specify timing of the actions (i.e., 0.000 means action executed at time $t = 0.0$, the very beginning of the mission).

Table 4. Scenario 1: Reconfigured System Fuel Loss Comparison against an Unchanged System

| System            | Fuel Delivered (% of nominal) | Fuel Lost |
|-------------------|-------------------------------|-----------|
| Reconfigured System | 95.51%                        | 0.13%     |
| Unchanged System   | 49.93%                        | 49.96%    |

unfaulted system in nominal conditions, while losing only 0.13% of the spent fuel. The advantages of reconfiguration with respect to fuel usage can be further analyzed by tracking the cumulative fuel siphoned from the tanks and delivered to the engines in Figure 7.

Scenario 2: Worsening Leak contains two faults: valve 8 stuck shut, and a leaky pipe 6. The intensity of the leak in the initial state is 0.1 but increases by 0.1 per hour. This configuration of the system and the placement of the faults prevents the planner from isolating the leak and rerouting the fuel around it (as was done in scenario 1). In order to mitigate the leak, the reconfiguration
module can shut off one of the fuel tanks. However, doing that too early would result in a failure as one tank does not hold enough fuel to supply both engines simultaneously. The reconfiguration module has no alternative but to allow some fuel to leak into the vessel. Here, fuel tanks hold a total of 750 gallons, which is sufficient to supply fuel to both engines at 30 Gal/hr for 10 hours (with a reserve for leakage).

The reconfigured system for Scenario 2 is depicted in Figure 10, followed by the temporal plan outlined in Figure 9, which shows the adjustments made to valve positions and pump speeds after 5 hours, once the leak in pipe 6 becomes uncontrollable. The reconfiguration is achieved by opening the cross valve 16 at t = 5hr and increasing the speed of pump 1 at t = 6 hr to distribute the fuel from tank 1 between both engines. Simultaneously, at time t = 6hr, valve 10 is closed, shutting off tank 2, and valve 12 is closed to prevent a backward leak, isolating the leak in the process (since cross valve 15 is closed by default). By examining the results from Table 5, it is apparent that the reconfigured system performs well, losing only 3.66% of the fuel, while delivering 91.39% of fuel to the engines compared to an unfaulted system in nominal state. On the other hand, while the unchanged system delivered 82.2% of cumulative fuel to the engines, compared to the nominal system, it had to spend a substantial overhead of the tank reserves, losing 34.93% of spent fuel. In fact, in the unchanged system, the leak would cause tank 2 to run dry after 7.55 hours. The differences in fuel usage (spent and received) between the reconfigured and unchanged system are further highlighted in Figure 11.

6 END-TO-END RESULTS

We evaluate the resilience metric defined in Section 2.5 under three scenarios. These scenarios include leak faults only, and are chosen to demonstrate the impact of the diagnosis, prognostics and
reconfiguration modules on satisfying the system requirements. In Scenario 1, we include no health monitoring feature, while in Scenario 2, we include only the diagnosis and the reconfiguration modules, but no prognostics. The goal of Scenario 2 is to demonstrate the resilience cost we pay when no prognostics is used. In Scenario 3, we consider the impact of all three modules. For each scenario, we report the average system resilience over all considered faults. We consider two types of fault degradation profiles: linear and exponential. The fault parameter takes values in Refs. [0, 1], with 0 corresponding to no leak, and 1 corresponding to a catastrophic loss of fuel. The RUL of a pipe is defined as the average time until the fault parameter reaches 0.8.

Diagnosis setup: We consider a time horizon of 1,800 seconds, and a diagnosis window of 10 seconds, each window having 10 samples. For each leak, fault we consider the following linear and exponential degradation models:

\[
p_{\text{linear}}(t) = \begin{cases} 
0 & t < 0, \\
\frac{1}{1200} t & t \in [0, 960], \\
0.8 & t > 960,
\end{cases}
\]

\[
p_{\text{exponential}}(t) = \begin{cases} 
0 & t < 500, \\
2^{\frac{t-500}{400}} - 1 & t \in [500, 839.2], \\
0.8 & t > 839.2.
\end{cases}
\]

The linear and exponential degradations reach failure after 960s and 839.2s, respectively. For each scenario we evaluate the resilience metric from Section 2.5 using the following parameters: \( y_1^d = y_2^d = 1\text{kg/s}, T = 1800\text{s}, \beta_1 = \beta_2 = 3600\text{kg}, \) and \( \alpha_1 = \alpha_2 = 0.5. \)

Reconfiguration setup: For Scenarios 2 and 3, we consider an initial temporal horizon \( T = 1800\text{s}, \) and a discretized time step \( \Delta t = 20\text{s}. \) The temporal discretization dictates the accuracy of the dynamics and the state space size, thus this value needs to be fine enough to produce meaningful results while simultaneously ensuring solvability in a reasonable time. To demonstrate the impact of time discretization on performance and resilience scores, we also present results obtained using a time step \( \Delta t = 50\text{s} \) and \( \Delta t = 100\text{s}. \) This is a crucial parameter in reconfiguration planning as it dictates the fault degradation update frequency and the frequency of decision points for reconfiguration actions\(^3\). Overall, the more frequent the updates, the more accurate the discretized model of the system and its evolution over time, thus the resilience metric results are more reliable and trustworthy for finer time discretization.

For a fair comparison, we consider identical planning parameters and constraints for both scenarios. We also use a resilience metric-based heuristic estimate \( h(s) \) adjusted by the temporal information to bias the search toward states further into the plan (i.e., states for which more time has elapsed): \( h(s_t) = (\alpha_1 \mathcal{J}_1^1 - \alpha_2 \mathcal{J}_2^2) \omega(T - t)/T, \) where \( T \) is the temporal horizon, \( s_t \) is a planning

\(^3\)After time discretization, the reconfiguration actions can only be applied at multiples of the discretized time interval \( \Delta t. \) The system is assumed to be static between the decision points, and any updates due to events and processes are discrete.
state at time $t$, $J^1_t$ and $J^2_t$ measure the deviation from the required massflow rates and fuel loss, respectively, over the horizon $T$. Weights $\alpha_1 = 0.5$ and $\alpha_2 = 0.5$ as in the original resilience metric equation, whereas $\omega = 1.5$. We also impose a global constraint which must hold throughout the duration of the plan such that the fuel massflow rate at the engines cannot deviate by more than ±30% from the nominal massflow rate. To improve the runtime efficiency, we only return the first encountered solution, regardless of its quality (feasibility planning).

In **Scenario 2**, without prognosis the reconfiguration module assumes a static leak intensity and no degradation over time. However, to maintain accuracy a new diagnosis is recomputed after each time step $\Delta t$ has elapsed. The reconfiguration planning process is repeated after each diagnosis with updated fault information (leak intensity) and a shorter time horizon $T' = T - \Delta t$. The initial action from each reconfiguration iteration is appended to the global plan. **Scenario 3** does not require periodic re-planning as it incorporates the degradation model directly in the PDDL+ domain.

Leak faults 1, 2, 7, and 8 are unrepairable by the reconfiguration module. This is due to the location of the leaks which, by design of the system, makes it impossible to isolate the fault and mitigate its effects. If the reconfiguration module fails to find any solutions, the vessel might request assistance from human operators. To avoid confusion, in Tables 7 and 8, we denote such unrepairable cases with "×", whereas "***" signifies that there exists a plausible repair but the reconfiguration module did not find it within the allotted time. The reduced resilience captures the loss of fuel at the leak points. As expected, in the case of the exponential degradation model, the resilience loss is more significant.

**Scenario 1 results**: No faults are tracked and no actions are taken in response to faults. Table 6 presents the effects of tracking no faults on the resilience metric.

**Scenario 2 results**: We assume we detect and isolate the leaks but use no prognosis. The reconfiguration algorithms assume that the fault magnitudes remain constant, not making use of the predicted fault degradation models. In both fault degradation modes, the fault probabilities of the true fault are dominant, once the fault magnitudes overcame the measurements noise. For each 180 windows, the diagnosis algorithm takes under 2 seconds to obtain a solution. Hence, we are well under the 10 seconds windows size, indicating the feasibility of real-time implementation of the diagnosis module. For small leaks, due to the measurement noise, the estimates oscillate close to zero in the beginning. As the magnitude of the leaks increase, the diagnosis algorithm is able to accurately track the true fault parameter, for both degradation models. Good fault estimates are imperative for training accurate prognosis models.

Reconfiguration in **Scenario 2** using only diagnosis fails to improve the resilience of the system. The inability to solve the tasks using more accurate models can stem from the gradient of fault degradation, and the way the heuristic directs the search algorithm. In essence, the heuristic prioritizes states in which more time has elapsed. This, in turn, causes the reconfiguration actions to adjust the valve and pump settings to be applied later in the plan just before the system hits a critical point. However, without an accurate model of fault degradation the planner underestimates its evolution and postpones taking action until it is too late. After a re-diagnosis, it becomes clear that the fault has become uncontrollable and cannot be mitigated via reconfiguration. This is particularly visible when the fault degradation is updated more frequently at a smaller rate, with sparse steep increases in leak intensity, it is more likely that the planner will attempt to reconfigure earlier and make it before the point of no return.

| Degradation model | leak_fault_1 | leak_fault_2 | leak_fault_3 | leak_fault_4 | leak_fault_5 | leak_fault_6 | leak_fault_7 | leak_fault_8 | Average |
|-------------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|---------|
| Linear            | 0.65        | 0.65        | 0.65        | 0.65        | 0.64        | 0.64        | 0.66        | 0.66        | 0.649   |
| Exponential       | 0.66        | 0.66        | 0.66        | 0.66        | 0.66        | 0.66        | 0.68        | 0.68        | 0.676   |
There configuration module has found a valid plan to all solvable tasks proving that planning with coarser time discretization reduces accuracy of the solution since it underestimates the fault degradation rate (i.e., assumes steady state conditions for longer time intervals between updates). This is seen by the resilience metric scores increasing in the opposite direction of the planning model granularity.

Table 7. Scenario 2: Fuel System Resilience Results with Reconfiguration based on Diagnosis Only

| Degradation Model | Δt (s) | leak_fault_1 | leak_fault_2 | leak_fault_3 | leak_fault_4 | leak_fault_5 | leak_fault_6 | leak_fault_7 | leak_fault_8 | Average |
|-------------------|--------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|---------|
| Linear            | 20s    | 0.65        | 0.65        | 0.65        | 0.65        | 0.64        | 0.66        | 0.66        | 0.66        | 0.65    |
| Exponential       | 20s    | 0.66        | 0.66        | 0.66        | 0.66        | 0.66        | 0.66        | 0.66        | 0.66        | 0.66    |
| Linear            | 50s    | 0.65        | 0.65        | 0.65        | 0.65        | 0.64        | 0.64        | 0.64        | 0.64        | 0.65    |
| Exponential       | 50s    | 0.66        | 0.66        | 0.66        | 0.66        | 0.66        | 0.66        | 0.66        | 0.66        | 0.66    |
| Linear            | 100s   | 0.65        | 0.65        | 0.758       | 0.758       | 0.64        | 0.64        | 0.64        | 0.64        | 0.66    |
| Exponential       | 100s   | 0.68        | 0.68        | 0.758       | 0.758       | 0.68        | 0.68        | 0.68        | 0.68        | 0.67    |

Green cell background indicates that the resilience score improved with reconfiguration, red cell indicates decrease in resilience with reconfiguration and white cell indicates no change ('×' denotes no reconfiguration possible, '*' denotes that the planner was unable to find a solution).

Table 8. Scenario 3: Fuel System Resilience Results with Reconfiguration based on Both Diagnosis and Prognosis

| Degradation Model | Δt (s) | leak_fault_1 | leak_fault_2 | leak_fault_3 | leak_fault_4 | leak_fault_5 | leak_fault_6 | leak_fault_7 | leak_fault_8 | Average |
|-------------------|--------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|---------|
| Linear            | 20s    | 0.65        | 0.65        | 0.688       | 0.662       | 0.911       | 0.911       | 0.66        | 0.66        | 0.724   |
| Exponential       | 20s    | 0.66        | 0.66        | 0.626       | 0.626       | 0.927       | 0.927       | 0.68        | 0.68        | 0.723   |
| Linear            | 50s    | 0.65        | 0.65        | 0.721       | 0.721       | 0.915       | 0.915       | 0.66        | 0.66        | 0.735   |
| Exponential       | 50s    | 0.66        | 0.66        | 0.652       | 0.652       | 0.930       | 0.930       | 0.68        | 0.68        | 0.738   |
| Linear            | 100s   | 0.65        | 0.65        | 0.758       | 0.758       | 0.917       | 0.917       | 0.66        | 0.66        | 0.746   |
| Exponential       | 100s   | 0.68        | 0.68        | 0.717       | 0.717       | 0.937       | 0.937       | 0.68        | 0.68        | 0.748   |

Green cell background indicates that the resilience score improved with reconfiguration, red cell indicates decrease in resilience with reconfiguration and white cell indicates no change ('×' denotes no reconfiguration possible, '*' denotes that the planner was unable to find a solution).

The reconfiguration module only managed to solve two problems in each degradation fault model and only using the coarsest discretization. Thus the confidence in these results is low, since they lack accuracy.

Table 9. Scenario 3: Average Reconfiguration Runtimes (in Seconds)

| Fault Model | Time Interval (Δt) | Linear | Exponential |
|-------------|-------------------|--------|-------------|
| Leak 3/4    | 20s               | 1080.3 | 114.6       |
|             | 50s               | 103.8  | 31.9        |
|             | 100s              | 214.0  | 73.5        |
| Leak 5/6    | 20s               | 97.1   | 56.75       |
|             | 50s               | 221.9  | 73.5        |
|             | 100s              | 214.0  | 167.2       |

Using both diagnosis and prognosis (Scenario 3) significantly maintains accuracy of the fault evolution via a continuous process in the PDDL+ model, thus the reconfiguration task only needs to be solved once. In contrast, Scenario 2 (static diagnosis only) requires continual iterative re-diagnosis and re-planning with each elapsed time interval. Table 9 shows the average planner runtimes per discretization quantum.
7 COMPARISON WITH THE STATE-OF-THE-ART

There are currently no widely recognized benchmarks for measuring resilience, making it difficult to draw direct comparisons between various approaches. Resilience studies tend to be application and metric-specific, ranging from qualitative discussions to the assessment of resilience indices. In the context of CPSs, the discussion of resilience typically focuses on either hardware or software components. In our work, we specifically emphasize hardware-related resilience. Note that there is no universally applicable definition of resilience that can be used for all applications.

Software resilience aims to ensure the uninterrupted operation of software programs, minimizing the costs associated with restarting failed simulations in terms of both time and power consumption [32]. Various approaches have been proposed for implementing software resilience, such as Coordinated Checkpoint and Restart (C/R) [20, 58], Uncoordinated C/R [31], and Local Failure Local Recovery (LFLR) [61]. Our approach to hardware resilience shares similarities with the self-adaptive software systems paradigm [16], which utilizes architectural models for high-level decision-making regarding system adaptation. This approach shows promise in constructing resilient software systems in a cost-effective manner. It incorporates two key ideas: (1) rescheduling to mitigate performance degradation caused by unresponsive devices and (2) scaling up to optimize performance by efficiently utilizing CPU and memory resources in processor nodes.

In the hardware domain, the study of resilience primarily focuses on energy systems. A comprehensive review of resilience in energy systems can be found in Ref. [30], where the resilience discussion is, in part, organized around four functions: resist, restabilize, rebuild, and reconfigure. The term resist pertains to the system’s ability to endure disturbances without significantly affecting its performance. This function is typically achieved through feedback control, as exemplified by our fuel rate controllers. Restabilize concerns the system’s capacity to swiftly restore critical functionalities to better absorb disruptions, particularly addressing cascading effects. In our case, the presence of dual fuel lines and connecting pipes and valves ensures fuel reserve diversity and flexibility, enabling this function. When a leak is detected, valves are closed to prevent fuel losses, or opened to access available fuel resources. Recovery reflects the system’s capability to return to its normal state. We implement this function through automated planning. However, in our scenario, we cannot guarantee a return to the original state but rather aim to mitigate the effects of disruptions. This function relates to the existence of redundancies, which must strike a balance between cost and benefit. Increasing the number of valves and tanks can enhance the recovery function but comes at a cost, such as increased ship mass and reduced maneuverability. Lastly, the reconfigure function pertains to a system topology or structure that makes it more fault-tolerant. This concept involves learning optimal reconfigurations and applying them to future disruptions. We implement this function in real-time, searching for the best topology (i.e., pipe interconnections) to minimize the impact of disruptions. Furthermore, we can encode various topologies associated with different types of disruptions and transfer this knowledge for use in the future or similar systems.

In the energy domain, power systems are prime candidates for resilience analysis. A comprehensive examination of current practices regarding resilience in power systems can be found in [11]. The authors define resilience using attributes-based metrics, such as robustness, adaptability, resourcefulness, and recoverability, as well as performance metrics. Performance metrics aid in the interpretation of quantitative data describing infrastructure outputs, specifying disturbances, and formulating metrics to gauge infrastructure resilience. Nevertheless, even within the power system domain, the proposed metrics have yet to achieve widespread acceptance and fail to comprehensively capture the essence of power system resilience. In our research, we employ a resilience definition based on a performance metric that pertains to our use case: fuel loss reduction. Unlike the power system domain, where resilience enhancement methods primarily rely on redundancies, we concentrate on the inherent resilience that may exist in the system, even if it is not fully known.
This type of resilience is leveraged during operation when we develop optimal action plans based on the current and predicted system health. Our goal is to ensure that operational constraints that define resilience are always satisfied.

There are three key features of our approach to resilience: first, its applicability extends to autonomous systems; second, it operates in real-time, ensuring continuous functionality; and third, it is supported by a comprehensive stack of health monitoring modules. This stack includes software modules implementing diagnosis, prognostics, and reconfiguration functionalities. Each module is designed to be scalable for real-time implementation, does not rely on failure progression models, and applies to both continuous and discrete system models. We have shown that this methodology is applicable to a wide range of systems, provided we possess a model and a basic understanding of the potential faults that may impact the system. In our previous work, we showed how fault-augmented Modelica models can be used for diagnosis [48] and maintenance [59]. We demonstrated that our optimization-based approach to diagnosis is effective when applied to both the time domain [44] and frequency domain [42]. In Ref. [45], we showed early evidence that our optimal control approach to disambiguation can be successful when applied to a simplified version of a fuel system. Moreover, our AI-planning approach helped agents operating in open worlds to detect novel domain shifts and adapt their environment models accordingly [56].

The contributions of our diagnosis approach can be viewed from two perspectives: (1) minimize diagnosis uncertainty by devising control inputs that effectively reduce it, and (2) improve the numerical efficiency of the algorithms. Diagnosis uncertainty reduction bears a similarity to the concept of active testing in the context of reducing the number of diagnostic hypotheses in Boolean circuits [25]. The reduction in uncertainty is achieved by generating test vectors for components, which is analogous to our process of generating inputs for through optimal control. The primary distinctions lie in two aspects: (1) the representations employed—[25] utilizes Boolean expressions to represent the model, while we employ hybrid Differential-Algebraic Equations DAEs, and (2) the approach to compute the test inputs - we employ optimization algorithms whereas [25] relies on combinatorial search. The optimization-based methodology for computing the input test can be seen as a finite horizon optimal control problem, which forms the foundation of model predictive control [29]. We enhance the numerical efficiency of the disambiguation optimization algorithm by utilizing surrogate models implemented in PyTorch. Surrogate models [21] play a crucial role in accelerating numerical simulations in computer-aided engineering applications. In our particular case, we have employed a deep learning platform to represent the surrogate model, enabling us to leverage gradient-based optimization algorithms that are much faster than gradient-free methods. The utilization of the differential programming paradigm for control applications is presently a thriving area of research that seeks to exploit the advancements in deep learning for improving the numerical efficiency of control algorithms [1, 43].

We employ a data-driven approach to prognostics, utilizing AR models that are learned online based on estimates of fault parameters. These models are then used to extrapolate and predict the evolution of fault parameters over time. In Refs. [8, 38], a comparison of various prognostics methods at both the component and system levels is provided. Our approach falls under the statistical data-driven approach, where the progression of damage is determined by condition monitoring data [65]. Linear AR models are not reliable for long-term estimations due to their sensitivity to initial conditions and systematic errors in prediction. To address this limitation, we employ non-linear AR models. Another alternative to AR models is the use of NNs, which have the potential to understand system behavior without requiring exact correlations between input and output parameters [7]. In our approach, we treat the degradation model as autonomous, meaning it is not directly dependent on the system’s state. However, there is an implicit dependence since fault parameters are estimated using state-dependent trajectories. We chose the degradation model based
on considerations of computational efficiency and data availability. Depending on the size of the architecture, NN models require significant computational resources and large training datasets, which may not be readily available in an autonomous ship application.

AI Planning-based approaches to system reconfiguration have been explored in the past to a limited extent [3], especially when using expressive models. Temporal and hybrid planning have been successfully used to control and reconfigure realistic systems such as Software Architectures [4], Urban Traffic Management [66], Dual-Arm Robot [10], and Power Generators [54]. Some planning-based reconfiguration approaches mitigate the impact of faults in complex systems such as Power Distribution Systems [63] or spacecraft operations [69], the latter of which also incorporated MBD of the fault. To the best of our knowledge, this is the first application of PDDL+ planning to fault mitigation via system reconfiguration. Unfortunately, standard AI Planning models, even if encoded in PDDL+, often lack expressive power to accurately represent all relevant aspects of the target system. Semantic attachments are used to bridge that gap in expressiveness, injecting external knowledge and computation into the planning phase. There have been few attempts at merging expressive planning with semantic attachments. Interestingly, semantic attachments have been used in diverse ways: non-linear dynamics calculations [6], heuristic estimators for improved search guidance [9], or belief propagation and posterior computation [64]. The approach of Ref. [64] is most similar to ours, using an external function to predict the evolution of the system. However, none of the existing PDDL+ planning approaches incorporated semantic attachments as complex as the FMU simulator of the fuel system presented in this article.

**Algorithm complexity discussion:** The complexity of the resilience framework depends on the complexity of the algorithms utilized by the three modules. For testing various fault hypotheses, we employ the Powell algorithm [57]. In our case, since we optimize for a single variable, the convergence rate primarily relies on the efficient bi-directional line search performed using Brent’s method [15]. In the worst-case scenario, Brent’s method exhibits a linear convergence rate. The overall convergence time is primarily affected by the FMU simulation time, which can be partially controlled by adjusting the DAE solver parameters (e.g., tolerance, number of collocation points). For diagnosis disambiguation, we employ Adam [39], a variant of the gradient descent algorithm. In the best-case scenario, the squared norm of the loss function gradient converges at a rate of $O(\log(T)/\sqrt{T})$ [72], where $T$ represents the number of iterations. The prognostics module utilizes recursive least squares to update the degradation model parameters. The estimation error of these parameters is known to converge to zero at a rate of $O(1/\sqrt{T})$ [41], where $T$ represents the number of iterations. We control the time complexity of these algorithms by limiting the number of iterations allocated to each task, where we take into account the maximum allotted time and the time per iteration. The AI planner uses a discretized state and action space, sufficiently coarse to assure efficient search. In the discretized setting, the complexity of PDDL+ increases exponentially with the time discretization and the number of states/actions. We control the complexity of the search by bounding the temporal horizon and the depth limit of the search graph. These settings are complemented by a resilience-focused heuristic search that intelligently explores the state space.

**8 CONCLUSIONS**

This article has shown that designing a resilient system is not simply a matter of including a module for “resilience.” Rather, resilience must be built in at the ground level for the system to be able to be as successful as possible at achieving its mission in the face of unforeseen faults and damage. Each of the modules must be adapted from what one sees in conventional design, and all these adaptations must be carefully coordinated to achieve resilience. Our run-time architecture requires (1) a diagnosis module that detects and isolates any fault using sensor data, (2) a disambiguation module that further isolates the fault by manipulating control inputs, (3) a prognostics module...
that predicts fault progression, and (4) a reconfiguration module which activates internal redundancy to ensure mission success. To achieve these we used parameter estimation, regression and AI planning. We could have used alternative approaches to building these four modules, however, there is no avoiding needing the modules. We have shown that each of the modules is needed to achieve the best system resilience. We have shown that a model-based approach allows all designs to use exactly the same run-time and design-time software. We only need to be provided a system description represented in a systems modeling language (such as Modelica) and a PDDL description of system goals, constraints and possible actions. This description is relatively simple because system dynamics is encoded through use of the FMU (of the Modelica model). From this model and PDDL description provided at design time we fully automatically compile the data structures required for run-time operation. In particular, we build an FMU, decompose the model and construct surrogate models for efficiency, and build the PDDL structures needed for planning.

REFERENCES

[1] B. Amos, I. Jimenez Rodriguez, J. Sacks, B. Boots, and J. Zico Kolter. 2018. Differentiable MPC for end-to-end planning and control. In Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018 (NeurIPS’18). 8299–8310.

[2] J. F. Archard. 1953. Contact and Rubbing of flat surfaces. Journal of Applied Physics 24, 8 (1953), 981–988. DOI: https://doi.org/10.1063/1.1721448

[3] N. Arshad and D. Heimbigner. 2005. A Comparison of Planning Based Models for Component Reconfiguration. Technical Report CU-CS-995-05. Colorado University.

[4] N. Arshad, D. Heimbigner, and A. Wolf. 2003. Deployment and dynamic reconfiguration planning for distributed software systems. In IEEE ICTAI.

[5] M. S. Arulampalam, S. Maskell, and N. Gordon. 2002. A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking. IEEE Transactions on Signal Processing 50, 2 (2002), 174–188.

[6] J. Bajada, M. Fox, and D. Long. 2015. Temporal planning with semantic attachment of non-linear monotonic continuous behaviours. In Proceedings of the 24th International Conference on Artificial Intelligence (IJCAI’15). AAAI Press, 1523–1529.

[7] O. Bektas and J. A. Jones. 2016. NARX time series model for remaining useful life estimation of gas turbine engines. In PHM Society European Conference, Vol. 3. Issue 1. DOI: https://doi.org/10.36001/phme.2016.v3i1.1610

[8] O. Bektas, J. Marshall, and J. Jones. 2020. Comparison of Computational Prognostic Methods for Complex Systems Under Dynamic Regimes: A Review of Perspectives. Archives of Computational Methods in Engineering 27, 4 (2020), 999–1011.

[9] S. Bernardini, M. Fox, D. Long, and C. Piacentini. 2017. Boosting search guidance in problems with semantic attachments. In International Conference on Automated Planning and Scheduling. Vol. 27. 29–37.

[10] R. Bertolucci, A. Capitanelli, M. Maratea, F. Mastrogiovanni, and M. Vallati. 2019. Automated planning encodings for the manipulation of articulated objects in 3d with gravity. In AI*IA - International Conference of the Italian Association for Artificial Intelligence. Springer, 135–150.

[11] N. Bhusal, M. Abdelmalak, M. Kamruzzaman, and M. Benidris. 2020. Power system resilience: Current practices, challenges, and future directions. IEEE Access 8 (2020), 18064–18086.

[12] S. A. Billings. 2013. Nonlinear System Identification: NARMAX Methods in the Time, Frequency, and Spatio-Temporal Domains.

[13] T. Blochwitz, M. Otter, J. Akesson, M. Arnold, C. Clauss, H. Elmqvist, M. Friedrich, A. Junghanns, J. Mauss, D. Neumerkel, H. Olsson, and A. Viel. 2011. The functional mockup interface for tool independent exchange of simulation models. In Proceedings of the 8th International Modelica Conference. 105–114.

[14] S. Bogomolov, D. Magazzeni, A. Podelski, and M. Wehrle. 2014. Planning as model checking in hybrid domains. In AAAI Conference on Artificial Intelligence.

[15] R. P. Brent. 1971. An algorithm with guaranteed convergence for finding a zero of a function. Comput. J. 14, 4 (01 1971), 422–425. DOI: https://doi.org/10.1093/comjnl/14.4.422

[16] J. Cámara, P. Correia, R. de Lemos, and M. Vieira. 2014. Empirical resilience evaluation of an architecture-based self-adaptive software system. In Proceedings of the 10th International ACM Sigsoft Conference on Quality of Software Architectures (QoSA’14). Association for Computing Machinery, 63–72.

[17] M. Cashmore, M. Fox, D. Long, and D. Magazzeni. 2016. A compilation of the full PDDL+ language into SMT. In International Conference on Automated Planning and Scheduling, Vol. 26. 79–87.

ACM Transactions on Cyber-Physical Systems, Vol. 8, No. 1, Article 7. Publication date: January 2024.
[18] A. Cimatti, E. Giunchiglia, F. Giunchiglia, and P. Traverso. 1997. Planning via model checking: A decision procedure for AR. In Recent Advances in AI Planning. Springer, 130–142.

[19] A. Coles, A. Coles, M. Fox, and D. Long. 2012. COLIN: Planning with continuous linear numeric change. J. Artif. Int. Res. 44, 1 (2012), 1–96.

[20] J. T. Daly. 2006. A higher order estimate of the optimum checkpoint interval for restart dumps. Future Generation Computer Systems 22, 3 (2006), 303–312. DOI: https://doi.org/10.1016/j.future.2004.11.016

[21] S. Davis, S. Cremaschi, and M. Eden. 2017. Efficient surrogate model development: Optimum model form based on input function characteristics. In 27th European Symposium on Computer Aided Process Engineering. Computer Aided Chemical Engineering, Vol. 40. Elsevier, 457–462.

[22] J. de Kleer, A. Mackworth, and R. Reiter. 1992. Characterizing diagnoses and systems. Journal of Artificial Intelligence Research 56, 2–3 (1992), 197–222.

[23] G. Della Penna, D. Magazzeni, and F. Mercorius. 2012. A universal planning system for hybrid domains. Appl. Intell. 36, 4 (2012), 932–959.

[24] C. Dornhege, P. Eyherich, T. Keller, S. Trüg, M. Brenner, and B. Nebel. 2009. Semantic attachments for domain-independent planning systems. In Nineteenth International Conference on Automated Planning and Scheduling.

[25] A. Feldman, G. Provan, and A. van Gemund. 2010. A model-based active testing approach to sequential diagnosis. JAIR 39, 1 (sep 2010), 301–334.

[26] M. Fox and D. Long. 2006. Modelling mixed discrete-continuous domains for planning. Journal of Artificial Intelligence Research 27, 1 (2006), 235–297.

[27] M. Fox and D. Long. 2011. PDDL2.1: An extension to PDDL for expressing temporal planning domains. (2011). https://doi.org/10.1613/jair.1129 arXiv: https://arxiv.org/abs/arXiv:1106.4561

[28] P. Fritzson. 2015. Principles of Object-Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical Approach (2 ed.). Wiley, Hoboken, NJ.

[29] C. Garcia, D. Prett, and M. Morari. 1989. Model predictive control: Theory and practice—A survey. Automatica 25, 3 (1989), 335–348.

[30] P. Gasser, P. Lusenberger, M. Cinelli, W. Kim, M. Spada, P. Burgherr, S. Hirschberg, B. Stojadinovic, and T. Y. Sun. 2021. A review on resilience assessment of energy systems. Sustainable and Resilient Infrastructure 6, 5 (2021), 273–299. DOI: https://doi.org/10.1080/23789689.2019.1610600

[31] A. Guermouche, T. Ropars, E. Brunet, M. Snir, and F. Cappello. 2011. Uncoordinated checkpointing without domino effect for send-deterministic MPI applications. In 2011 IEEE International Parallel & Distributed Processing Symposium. 989–1000. DOI: https://doi.org/10.1109/IPDPS.2011.95

[32] N. Gupta, J. R. Mayo, A. S. Lemoine, and H. Kaiser. 2020. Implementing software resiliency in HPX for extreme scale computing. (2020). https://doi.org/10.48550/arXiv.2004.07203

[33] A. Hertle, C. Dornhege, T. Keller, and B. Nebel. 2012. Planning with semantic attachments: An object-oriented view. Proc. of ECAI 242.

[34] R. Isermann. 2005. Model-based fault-detection and diagnosis—status and applications. Annual Reviews in Control 29, 1 (2005), 71–85.

[35] S. Julien and J. Uhrmann. 1997. New extension of the Kalman filter to nonlinear systems. In Signal Processing, Sensor Fusion, and Target Recognition VI (Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, Vol. 3068). 182–193. DOI: https://doi.org/10.1117/12.280797

[36] R. E. Kalman. 1960. A new approach to linear filtering and prediction problems. Transactions of the ASME—Journal of Basic Engineering 82, Series D (1960), 35–45.

[37] E. Kang, E. Jackson, and W. Schulte. 2010. An approach for effective design space exploration. In Monterey Conference on Foundations of Computer Software: Modeling, Development, and Verification of Adaptive Systems (FOCS’10), 33–54.

[38] S. Kim, J. Choi, and N. Kim. 2021. Challenges and opportunities of system-level prognostics. Sensors 21, 22 (2021), 1–25. DOI: https://doi.org/10.3390/s21227655

[39] Y. Li, T.R. Kurfess, and S. Y. Liang. 2000. Stochastic prognostics for rolling element bearings. J. Artif. Int. Res. 44, 1 (2012), 1–96.

[40] D. P. Kingma and J. Ba. 2015. Adam: A method for stochastic optimization. In International Conference on Learning Representations (ICLR’15). Retrieved from http://arxiv.org/abs/1412.6980

[41] Y. Li, T.R. Kurfess, and S. Y. Liang. 2000. Stochastic prognostics for rolling element bearings. Mechanical Systems and Signal Processing 14, 5 (2000), 747–762. DOI: https://doi.org/10.1006/mssp.2000.1301

[42] A. Marcet and T. Sargent. 1992. Speed of Convergence of Recursive Least Squares Learning with ARMA Perceptions. Economics Working Papers. Department of Economics and Business, Universitat Pompeu Fabra.

[43] I. Matei, A. Feldman, and J. de Kleer. 2018. Model-based diagnosis: A frequency domain view. In 2018 IEEE International Conference on Prognostics and Health Management (PHM).

[44] I. Matei, C. Zheng, S. Chowdhury, and J. de Kleer. 2021. Controlling draft interactions between quadcopter unmanned aerial vehicles with physics-aware modeling. Journal of Intelligent and Robotics Systems 101, 21 (2021), 1–21.

[45] I. Matei, M. Zhenirovskyy, J. de Kleer, and A. Feldman. 2018. Analytic redundancy relations guided parameter estimation for model-based diagnosis. In International Workshop on Principles of Diagnosis (DX’18).
[45] I. Matei, M. Zhenirovskyy, J. de Kleer, and K. Goebel. 2022. A control approach to fault disambiguation. Annual Conference of the PHM Society 14, 1 (2022), 1–8.

[46] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins. 1998. PDDL - The Planning Domain Definition Language. Technical Report CVC TR-98-003/DCS TR-1165. Yale Center for Computational Vision and Control.

[47] B. A. McElhoe. 1966. An assessment of the navigation and course corrections for a manned flyby of mars or venus. IEEE Transactions on Aerospace Electronic Systems 2, 4 (July 1966), 613–623. DOI: https://doi.org/10.1109/TAES.1966.4501892

[48] R. Minhas, J. de Kleer, I. Matei, B. Saha, B. Janssen, D.G. Bobrow, and T. Kurtoglu. 2014. Using fault augmented modelica models for diagnostics. In International Modelica Conference. 437–445.

[49] M. Najarian and G. J. Lim. 2019. Design and assessment methodology for system resilience metrics. Risk Analysis 39, 9 (2019), 1885–1898. DOI: https://doi.org/10.1111/risa.13274

[50] M. Panteli, P. Mancarella, D. N. Trakas, E. Kyriakides, and N. D. Hatzigianniu. 2017. Metrics and quantification of operational and infrastructure resilience in power systems. IEEE Transactions on Power Systems 32, 6 (2017), 4732–4742.

[51] P. Paris and F. Erdogan. 1963. Closure to “Discussions of ‘A Critical Analysis of Crack Propagation Laws’” (1963, ASME J. Basic Eng., 85, pp. 533–534). Journal of Basic Engineering 85, 4 (12 1963), 534–534.DOI: https://doi.org/10.1115/1.3656903

[52] A. Paszke, S. Gross, S. Chintala, G. Chan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. 2017. Automatic differentiation in PyTorch. In NIPS 2017 Workshop on Autodiff. 1–4.

[53] R. J. Patton, P. M. Frank, and R. N. Clark. 2000. Issues of Fault Diagnosis for Dynamic Systems. Springer-Verlag London.

[54] C. Piacentini, D. Magazzini, D. Long, M. Fox, and C. Dent. 2016. Solving realistic unit commitment problems using temporal planning: Challenges and solutions. In International Conference on Automated Planning and Scheduling, Vol. 26. 421–430.

[55] W. Piotrowski, M. Fox, D. Long, M. Magazzini, and F. Mercorio. 2016. Heuristic planning for PDDL+ domains. In International Joint Conferences on Artificial Intelligence. 3213–3219.

[56] W. Piotrowski, R. Stern, Y. Sher, J. Le, M. Klenk, J. de Kleer, and S. Mohan. 2023. Learning to operate in open worlds by adapting planning models. In International Conference on Autonomous Agents and MultiAgent Systems. ACM, 2610–2612. DOI: https://doi.org/10.1145/3545946.3599018

[57] M. J. D. Powell. 2007. A View of Algorithms for Optimization without Derivatives. Technical Report. University of Cambridge, UK.

[58] E. Roman. 2002. Survey of Checkpoint/restart Implementations. Technical Report LBNL-54942. Lawrence Berkeley National Laboratory.

[59] B. Saha, T. Honda, I. Matei, E. Saund, J. de Kleer, T. Kurtoglu, and Z. Lattmann. 2014. Model-based approach for optimal maintenance strategy. In European Conference of the Prognostics and Health Management Society.

[60] M. Staroswiecki and G. Comtet-Varga. 2001. Analytical redundancy relations for fault detection and isolation in algebraic dynamic systems. Automatica 37, 5 (2001), 687–699. DOI: https://doi.org/10.1016/S0005-1098(01)00005-X

[61] K. Teranishi and M. Heroux. 2014. Toward local failure local recovery resilience model using MPI-ULFM. In Proceedings of the 21st European MPI Users’ Group Meeting. 51–56. DOI: https://doi.org/10.1111/1.3656903

[62] D. Theilliol, H. Noura, and J. C. Ponsart. 2002. Fault diagnosis and accommodation of a three-tank system based on analytical redundancy. ISA Transactions 41, 3 (2002), 365–382. DOI: https://doi.org/10.1016/S0019-0578(07)60094-9

[63] S. Thiébaut, C. Coffrin, H. Hijazi, and J. Slaney. 2013. Planning with MIP for supply restoration in power distribution systems. In Proceedings of the 23rd International Joint Conference on Artificial Intelligence (Beijing, China) (IJCAI'13). AAAI Press, 2900–2907.

[64] A. Thomas, S. Amatya, F. Mastrogiavanni, and M. Baglietto. 2018. Towards perception-aware task-motion planning. In AAAI Fall Symposium on Reasoning and Learning in Real-World Systems for Long-Term Autonomy. Arlington, VA.

[65] G. E. P. Box and G. Jenkins. 1990. Time Series Analysis, Forecasting and Control. Holden-Day.

[66] M. Vallati, D. Magazzini, B. De Schutter, L. Chrupa, and T. McCluskey. 2016. Efficient macroscopic urban traffic models for reducing congestion: A PDDL+ planning approach. In AAAI Conference on Artificial Intelligence, Vol. 30.

[67] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, et al. 2020. SciPy 1.0: Fundamental algorithms for scientific computing in Python. Nature Methods 17, 3 (2020), 261–272. DOI: https://doi.org/10.1038/s41592-019-0686-2

[68] Y. Wang, . Chen, J. Wang, and R. Baldick. 2016. Research on resilience of power systems under natural disasters—a review. IEEE Transactions on Power Systems 31, 2 (2016), 1604–1613. DOI: https://doi.org/10.1109/TPWRS.2015.2429656

[69] B. Williams, M. Ingham, S. Chung, P. Elliott, M. Hofbaur, and G. Sullivan. 2003. Model-based programming of fault-aware systems. AI Magazine 24, 4 (2003), 61–61.

[70] W. Kufi Yu and T. Harris. 2001. A new stress-based fatigue life model for ball bearings. Tribology Transactions 44, 1 (2001), 11–18.
[71] X. Zhang, Q. Zhang, S. Zhao, R. Ferrari, M. M. Polycarpou, and T. Parisini. 2011. Fault detection and isolation of the wind turbine benchmark: An estimation-based approach. *IFAC Proceedings Volumes* 44, 1 (2011), 8295–8300. DOI: https://doi.org/10.3182/20110828-6-IT-1002.02808 18th IFAC World Congress.

[72] F. Zou, L. Shen, Z. Jie, W. Zhang, and W. Liu. 2019. A sufficient condition for convergences of adam and RMSProp. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR’19)*. 11119–11127. DOI: https://doi.org/10.1109/CVPR.2019.01138

Received 8 August 2022; revised 6 September 2023; accepted 20 October 2023