Timely and Accurate Detection of Model Deviation in Self-adaptive Software-intensive Systems

Yanxiang Tong
State Key Laboratory for Novel Software Technology, Nanjing University
Nanjing, China
tongyanxiang@gmail.com

Yi Qin
State Key Laboratory for Novel Software Technology, Nanjing University
Nanjing, China
yiqincs@nju.edu.cn

Yanyan Jiang
State Key Laboratory for Novel Software Technology, Nanjing University
Nanjing, China
jyy@nju.edu.cn

Xiaoxing Ma
State Key Laboratory for Novel Software Technology, Nanjing University
Nanjing, China
xxm@nju.edu.cn

Chang Xu
State Key Laboratory for Novel Software Technology, Nanjing University
Nanjing, China
changxu@nju.edu.cn

Chun Cao
State Key Laboratory for Novel Software Technology, Nanjing University
Nanjing, China
caochun@nju.edu.cn

ABSTRACT

Control-based approaches to self-adaptive software-intensive systems (SAS) are hailed for their optimal performance and theoretical guarantees on the reliability of adaptation behavior. However, in practice the guarantees are often threatened by model deviations occurred at runtime. In this paper, we propose a moDel-guided Deviation Detector (D^3) for timely and accurate detection of model deviations. To ensure reliability a SAS can switch at runtime from a control-based optimal controller to a conservative mandatory controller once an unsafe model deviation is detected. D^3 achieves both high timeliness and high accuracy through a deliberate fusion of deviated parameter estimation, uncertainty compensation and safety region quantification. Empirical evaluation with three exemplar systems validated the efficacy of D^3 (71.1% shorter detection delay, 39.0% less FN rate, and 24.1% less FP rate), as well as the benefits of the adaptation-switching mechanism (abnormal rate dropped by 28.2%).

ACM Reference Format:
Yanxiang Tong, Yi Qin, Yanyan Jiang, Xiaoxing Ma, Chang Xu, and Chun Cao. 2021. Timely and Accurate Detection of Model Deviation in Self-adaptive Software-intensive Systems. In *Proceedings of The 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2021).* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

ESEC/FSE 2021, 23 - 27 August, 2021, Athens, Greece

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Control theory has been increasingly adopted in developing self-adaptive software-intensive systems [52, 55]. In designing a control-based self-adaptive system (control-SAS for short), developers first identify a nominal model (e.g., a linear time-invariant model) of the subject system (a.k.a. the managed system, or the *plant* in control jargon), and then design a managing system (a.k.a. the *controller* in control jargon) based on the identified model with some (feedback) control mechanism such as Proportional-Integral-Derivative control or Model Predictive Control. The advantage of this approach is that the control mechanisms have well-established mathematical foundations that theoretically guarantee the optimality and reliability of the resulting SAS [25, 45, 60].

However, software-intensive systems behave differently from physical devices to which control theory is usually applied. The behavioral patterns of the former are more dynamic and uncertain than the latter. During its execution, a managed software-intensive system's behavior often deviates from the identified nominal model [10, 25, 46]. This *model deviation*, if exceeds a certain region, will invalidate the theoretical guarantees and threaten the safety of the whole system [9, 16, 44, 60].

The problem of model deviation has been acknowledged in the literature, but to our knowledge no satisfactory solution was given [10, 11, 23, 41, 46, 56]. Using robust controllers [11, 41] can mitigate the problem by tolerating light deviations at the cost of less optimality and greater complexity in controller design, but the problem itself is still there. Some authors proposed to monitor system outputs and rebuild the controller through system re-identification once the outputs violate some specified criteria [10, 23, 46]. However, this approach is problematic in that model deviations could have happened much earlier than their manifestation as abnormal outputs. It is crucial to report dangerous deviations as early as possible, before they cause disastrous consequences. Sliding window-based model deviation detection approaches [12, 23, 32] require extensive domain knowledge and human efforts to tune their parameters. The emerging learning-based approaches [16, 53]

reduce the dependence on domain knowledge and human efforts, but are not timely and accurate enough, as to be shown in Section 5.

In this paper, we propose the moDel-guided Deviation Detector (D^3), to support timely and accurate model deviation detection for control-SASs. The key intuition of D^3 is to *measure the nominal model's parameter values*. Comparing with existing approaches that monitor the managed system's output, measuring the parameter values save us the time that a model deviation propagates to the managed system's abnormal output. However, since the nominal model's parameters are unobservable, we have to detect model deviation through parameter estimation. D^3 integrates two lightweight techniques, namely environmental uncertainty compensation, and safety regions quantification, in order to improve detection accuracy without sacrificing too much timeliness.

Based on D^3, an adaptation-supervision mechanism is implemented to alleviate the impact of model deviation with a dual-track adaption strategy. The mechanism using a supervision loop to guard the adaptation loop of a control-SAS. Once D^3 detects model deviation, our mechanism will switch the SAS from the control theory-based optimal controller to a conservative controller that ensures the mandatory requirements but may scarify some system utilization. When model deviation disappears, the SAS can switch back to the optimal controller for higher system utilization.

We evaluate our approach on three representative exemplar SAS systems, namely, SWaT [8], RUBiS [18], and video encoder [47]. We compare D^3's performance with two baseline approaches (i.e., a sliding window-based detector [23], and an SVM-based detector [16]). The results show the effectiveness of D^3 which achieves 71.1% shorter detection delay, 39.0% lower FN rate, and 24.1% lower FP rate, as well as the usefulness of D^3-based adaptation-supervision mechanism by reporting a 28.2% lower abnormal rate.

In the remainder of this paper, section 2 gives the necessary background, and discusses the motivation of our work. Then, section 3 introduces the adaptation-supervision mechanism, and section 4 details our D^3 with three main techniques. Next, section 5 presents the experimental evaluation of our approach. Finally, section 6 discusses the related work and section 7 presents conclusions and future research directions.

2 BACKGROUND AND MOTIVATION

In this section, we first introduce the background of control-SASs, with a motivating example of Secure Water Treatment testbed ("SWaT" for short) [8]. Then we discuss the impact of model deviation on self-adaptive systems as the limitation of existing approaches in handling model deviation. Finally, we discuss our scope and assumptions in addressing the problem of model deviation.

2.1 Control-based self-adaptive systems

In recent years, control-based self-adaptive systems [25, 26, 52, 55] became a research hotspot for its fewer requirements on the developers' mathematical and software knowledge to design ad-hoc self-adaptation solutions. The methodology of implementing control-SASs includes a systematic data collection and model fitting procedure (a.k.a., *system identification*) and a control theory-guided controller designing procedure (a.k.a., *controller synthesis*). The former enables the automatic construction of an approximate model for the managed system, and the latter provides theoretical guarantees to the derived managing system's behavior in controlling adaptation.

Specifically, in system identification, control-SASs assume that the managed system's behavior can be captured by a quantitative *nominal model* to improve productivity and cope with infinite kinds of environmental dynamics. A *nominal model* describes the relationship between the managed system's output, the controller's output, and the environmental input. Many types of nominal models have been proposed to support self-adaptation in various scenarios, including linear time-invariant system [4, 32, 57], linear time-varying system[34], and nonlinear time-invariant system [11].

In controller synthesis, control-SASs leverage various control-theoretical techniques to design the optimal controllers for various scenarios. Two of the most prevailing techniques are proportional-integral-derivative (PID) control [24, 41, 56], and model predictive control [4, 5, 46, 48]. A controller's workflow resembles the conventional self-adaptation mechanism that consists of continuous *adaptation loops* of monitoring, analyzing, planning and executing. A controller's behavior should be subject to the control properties, for example, the SASO properties (i.e., stability, accuracy, settling time, and overshoot) [33].

We use the SWaT system to explain the concepts of control-SASs. SWaT is a fully operational scaled-down water treatment plant that produces doubly-filtered drinking water. SWaT consists of five water-processing tanks, as well as the pipes that connect those tanks. The in-coming valve and out-going valve of each tank can be controlled remotely via network. The objective of a self-adaptive SWaT system is to enable safe (e.g., no overflow or underflow in any of the tanks) and efficient (e.g., maximum clean water production) water filtering under different environmental situations (e.g., the initial water level of the five tanks, and the in-coming water flow of the first tank).

To build such a self-adaptive SWaT system following the control-SASs methodology, we can use the following linear time-invariant system to describe the nominal model of SWaT in system identification.

$$\begin{cases} \vec{x}(k) = A \cdot \vec{x}(k-1) + B \cdot \vec{u}(k-1) \\ y(k) = C \cdot \vec{x}(k) \end{cases} \tag{1}$$

Where (k) denotes a serial number of the adaption loop. For the variables, $\vec{x}(k)$ describes SWaT's current running state (i.e., the in-coming and out-going water of the five tanks), $\vec{x}(k-1)$ describes the historical state, $y(k)$ describes the system's output value (i.e., the measured water levels of five tanks), and $\vec{u}(k-1)$ describes its received control signal (i.e., the valves' opening time in an adaptation loop).

The parameters in Equation 1, including A, B, and C, are identified in system identification since their values cannot be measured directly. Each of these parameters represents a specific dynamical feature of SWaT. Specifically, A represents the system's delay property, which describes the interval between the time when a controller requests to turn on/off a valve and the time when the valve is turned on/off. B represents the system's controllability, which describes the influence of turning on/off a valve upon the system's running state. C represents the system's observability,

which describes the mapping between each tank's in-coming and out-going water flow and the measured water level.

Based on the identified parameter values, SWaT's developers provide a suite of well-designed controllers to enable the system's adaptation to different environmental situations. These controllers leverage both control-theory and rule-based domain knowledge to guide the control strategies of various valves. Specifically, there are a total of six controllers. Four controllers control the in-coming and out-going valves of one or two tanks, respectively. Two controllers control the general water treatment procedure and coordinate with the other four controllers.

Control theory guarantees the self-adaptive SWaT system subjecting to the control properties. For example, on the one hand, the water in all tanks should neither overflow nor underflow (i.e., subject to no overshoot property). On the other hand, the water level in all tanks should reach the desired level quickly (i.e., subjecting to low settling time property), in order to maximize SWaT's clean water production.

2.2 Model deviation in control-based self-adaptive systems

Model deviation damages the effectiveness of control-SASs. Model deviation is the mismatch between the nominal model's identified parameter values and its actual parameter values at runtime. The occurrence of model deviation may invalidate the theoretical guarantees provided by control theory and could potentially cause the abnormal behavior of a control-SAS.

Let us consider two real-world situations of model deviation in SWaT. The first situation is reported by Adepu et al. that network attacks could cause SWaT's abnormal behavior [1]. For example, if the controller's signal of turning on a valve is blocked or tampered, the valves' opening time will be changed accordingly. Then, when the controlled tank's water level exceeds an alarming level, the corresponding controller will still control the valves according to the unchanged B value and cause tank overflow.

In the second situation, the physical condition of SWaT's pipes and valves will also lead to model deviation. For example, physical abrasion [21] may wear the valves and result in the change of water flow rate (i.e., system's controllability feature B). Then, when a controller still controls the valves based on the unchanged B value, the water flow into the tank will be smaller than what the controller expected and may cause the tank to underflow.

Lots of self-adaptive system researchers acknowledge the existence of model deviation, and propose different solutions. In their pioneering push-button-method, Filieri et.al first identify model deviation, and propose to monitor system outputs and rebuild the controller through system re-identification. Some other works [10, 24, 46] also rely on re-identification to handle model deviation. Some researchers focus on improving the controller's robustness to tolerate light deviations [11, 41].

These two solutions have their own limitations. For the robust-strengthening approaches, model deviation does cause the controller designed by field experts to behave abnormally, in our previous motivating example of SWaT. We cannot simply assume that developer-designed controllers could overcome model deviation as in [11, 41]. For the re-identification approaches, their detection of model deviation is problematic since model deviations could have happened much earlier than their manifestation as abnormal outputs. (Kang et.al report that it takes around 5 minutes for an detected attack to fail SWaT's execution [39]). The latency between model deviation's occurrence and its observable consequences makes it crucial to report dangerous deviations as early as possible.

As we discussed in Section 1, the challenge of timely and accurate detection of model deviation is to balance the detection's timeliness and accuracy. On the one hand, if a detector requires a long detection time to improve its accuracy, it may leave the managed system executing without guarantees for a long time and increase its chance to behave abnormally. On the other hand, if the detector sacrifices its accuracy for shorter detection delay, it may result in lots of missing detections and false alarms and could also leave the system behaving abnormally.

Existing works fail to balance these three factors in detecting model deviation. For the sliding window-based approaches that have been exploited long by self-adaptive researchers [12, 23, 32], their performance largely depends on manual or empirical settings (e.g., size of the sliding window-based and the threshold for monitored system's output values). It often requires extensive domain and mathematical knowledge to determine a good setting, which is impossible for non-expert developers. For those newly-proposed learning-based approaches [16, 53], they usually combine multiple test results for improving their detection accuracy, which significantly increases their detection delay. What's more, these approaches often require binary training sets, the positive instance of which can hardly be prepared since the managed system's behavior under model deviation is unpredictable.

Our D^3 addresses the challenge above following the guidance of the nominal model. First, D^3 fundamentally decreases its detection delay by directly estimating the nominal model's parameter values. Second, D^3 improves its estimation accuracy and detection accuracy by two supporting techniques, namely, environmental uncertainty compensation and theoretical safety region quantification. Third, D^3 further reduces the time overhead of these two techniques by using Kalman filter estimation and probability-based quantification, respectively.

2.3 Our scope and assumptions

Due to the various application scenarios of self-adaptive systems, the proposed D^3 has its scope and assumptions for ease of discussion and presentation. We delineate our scope based on existing work and use pre-existing assumptions that have been widely used by other self-adaptation or control theory researchers.

Nominal model. In this work, we focus on linear time-invariant system ("LTI system" for short), one of the most widely-used models among self-adaptation researchers [4, 32, 46, 56, 57]. Nevertheless, our D^3 can be extended to other types of nominal models since our supporting techniques of uncertainty compensation and safety region quantification is independent to the types of the nominal model.

Model deviation. We focus on two types of model deviation, discrete behavior and inaccurate environmental interaction, following the two reported cases in our motivating example of SWaT. The former occurs when a controller produces its control signals (e.g.,

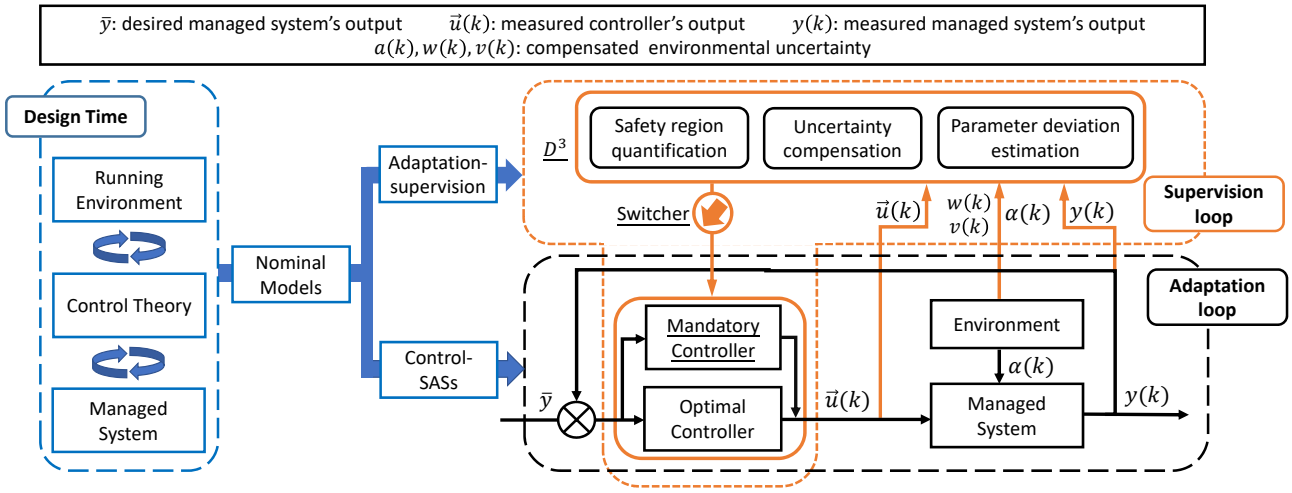

Figure 1: Overview of D^3-based adaptation-supervision mechanism

block or tamper with the instruction to open the valves) and the latter occurs when the managed system receives the control signals (e.g., water flow changing due to physical abrasion).

Based on the two types of model deviation, in this work, we focus on the deviation of the managed system's controllability features (i.e., the parameter B in the nominal model) and assume that the managed system's delay and observability features (i.e., parameter A and C in the nominal model) are comparatively stable. On the one hand, the deviation of the controllability features is reported and validated in recent literature [1, 2, 38]. On the other hand, our proposed approach can be extended to handle the deviation of the other two features.

We also assume that $B(i)$ is a unary parameter for ease of presentation. Nevertheless, our approach can be applied to a multinary parameter. In fact, in our later experiments, we already run D^3 to detect model deviation on binary parameters.

Environmental uncertainty. Environmental uncertainty is "the known unknown" information at runtime [4, 20]. As a result, one has to model the target uncertainty first before addressing it. In this paper, we assume that the environmental uncertainty can be described by a linear time-invariant model or normal distribution. The former is a widely used approach in describing uncertain environmental input by the control theory researchers [13, 58] and the latter is a typical setting for describing uncertainty's influence among self-adaptation researchers [19, 59].

3 AN ADAPTATION-SUPERVISION MECHANISM

To address the problem of model deviation, this paper proposes an adaptation-supervision mechanism to supervise the execution of control-SASs as shown in Figure 1. Intuitively, we added a new parallel *supervision loop* to a control-SAS. The supervisor continuously monitors the adaptation loop for model deviation. When model deviation is detected, the supervisor immediately "falls back" to a safer (but less efficient) mandatory controller and recovers to the main control loop successful model re-identification.

The supervisor consists of three major components: a model-guided deviation detector (D^3, which is this paper's primary focus), a mandatory controller, and a switcher. Underlined blocks in Figure 1 denote these newly added components.

In the adaption-supervision mechanism, we first require the system designer to provide a conservative *mandatory controller* that guarantees minimal system functionality (i.e., mandatory requirements) even on nominal-model deviation. Mandatory controllers have been extensively studied by the community of control-SASs, e.g., by adopting architectural-guided [27] or goal-driven rules [54].

With both control theory-based optimal controller and mandatory controller, the *switcher* coordinates them on the model-based deviation detector (D^3). First, the switcher immediately changes the adaptation loop from using the original controller to using the mandatory controller on model deviation to reduce the unpredictable influences to a minimum. Second, the switcher is also responsible for consistently conducting model re-identification and turning back to the optimal controller when the abnormal situation finishes (e.g., the network attack of SWaT was blocked by a firewall). The switcher can be implemented following the push button method [23].

Therefore, the design of an efficient and effective model-deviation detector (our D^3) should be the major research challenge. Such a design is presented as follows.

4 D^3: MODEL-GUIDED DEVIATION DETECTOR

The three major technical designs (and contributions) of the model-guided deviation detector D^3 are:

(1) D^3 *directly estimates the nominal model's parameter values* by modeling the nominal model's parameters as *stochastic variables*. In contrast with existing work, which is mostly based on the observable outputs $y(k)$, this white-box approach fundamentally accelerates the model-deviation detection.
(2) D^3 *compensates environmental uncertainty* in the nominal model by introducing compensation terms to reduce the

effects of measurement errors, yielding improved model-deviation estimation accuracy. D^3 adopts the Kalman filter, a fast Gaussian process to accelerate its estimation on the parameter values.

(3) D^3 *directly quantifies the nominal model's theoretical safety region* without the need for managed system's execution traces. Based on the necessary conditions under which model deviation can make the controller behave abnormally, D^3 directly (and thus timely) reports model deviation with sufficient probabilistic confidence.

These technical details are elaborated on as follows.

4.1 Parameter deviation estimation

We first enhance the nominal model (Equation 1) explicitly modeled time-variant model parameters B:

$$\begin{cases} B(k) = B(k-1) + \omega \cdot (\tilde{B} - B(k-1)) + p(k) \\ \vec{x}(k) = A \cdot \vec{x}(k-1) + B \cdot \vec{u}(k-1) \\ y(k) = C \cdot \vec{x}(k), \end{cases} \tag{2}$$

where: (1) Model deviation caused by discrete behavior is described as the sudden changes of centripetal force ω that drive $B(k)$ away from $\tilde{B}$. (2) Model deviation caused by inaccurate environmental interaction is described as a random walk procedure, the step of which subjects to a normal distribution $p(k)$.

Based on Equation 2, $B(k)$'s value in each adaption loop can be estimated by a Gaussian process [42]. Considering that both $B(k)$ and $\vec{x}(k)$ values cannot be directly measured, we perform Gaussian process in an iterative manner. Specifically, in the i-th adaptation loop, we first estimate the value of $\vec{x}(i-1)$ based on the historical variable values $y(i-2)$ and $\vec{u}(i-2)$ and the previously identified parameter value $\tilde{B}$. Then, based on the estimated $\vec{x}(i-1)$ value, we further derive the value of $B(i)$ from the current variable values $y(i)$ and $\vec{u}(i-1)$. If we do not detect model deviation with the newly-derived $B(i)$ value, we will update the value of $\tilde{B}$ by considering the estimated $B(i)$ value, to make the future estimation more accurate.

4.2 Uncertainty compensation

Due to various environmental uncertainties, a direct estimation of $B(i)$'s value is usually inaccurate. To reduce uncertainty-based errors in the estimations of $B(i)$, we refine the nominal model by compensating the effects of environmental uncertainty and accelerate our estimation with the Kalman filter.

Using compensation terms to refine a nominal model is a widely-used technique in the control theory community [29]. However, one has to carefully design the effective compensation terms. Here, we introduce three different compensation terms. One is the measured environmental input $a(k)$, which can be used to posterior calibrate our estimated $B(i)$'s value. We use a linear time-invariant model to describe the influence brought by environmental input.

The other two compensation terms, $w(k)$ and $v(k)$, captured the influences of measurement errors. Specifically, $w(k)$ describes the measurement error's impact on the managed system's internal state, and $v(k)$ describes the error's impact on the system's external behavior. According to our assumptions, we depict $w(k)$ and $v(k)$ as normal distribution of variance W and V respectively.

Then, the deviation-annotated nominal model is refined with three compensation terms, as follows.

$$\begin{cases} B(k) = B(k-1) + \omega \cdot (\tilde{B} - B(k-1)) + p(k) \\ \vec{x}(k) = A \cdot \vec{x}(k-1) + B \cdot \vec{u}(k-1) + \gamma \cdot a(k) + w(k) \\ y(k) = C \cdot \vec{x}(k) + v(k) \end{cases} \tag{3}$$

Where γ is the coefficient of the linear model that describes $a(k)$. The value of γ which is determined during system identification along with other model parameters.

Based on Equation 3, D^3 further alleviates the impact of measurement error by transforming the nominal model to difference equations. In the original nominal model, the measurement error on parameter values (e.g., A) is multiplied by other variables (e.g., $\vec{x}(k-1)$). As a result, a large variable value would amplify the measurement error and cause a bigger error in the following estimation. However, in the context of control-SASs, which is free from severe changes, a variable's values in two consecutive adaptation loops should not differ a lot [14, 51]. As a result, by changing to a difference equation, D^3 avoids the multiplier effect on the inevitable measurement error. In this paper, we use $\Delta var(k)$ to denote the difference between variable var's values in the k-th and $(k-1)$-th adaptation loop (i.e., $\Delta var(k) = var(k) - var(k-1)$). Then the difference equations of the nominal model are listed as follows.

$$\begin{cases} B(k) = B(k-1) + \omega \cdot (\tilde{B} - B(k-1)) + p(k) \\ \Delta\vec{x}(k) = A \cdot \Delta\vec{x}(k-1) + B \cdot \Delta\vec{u}(k-1) + \gamma \cdot \Delta a(k) + w(k) \\ \Delta y(k) = C \cdot \Delta\vec{x}(k) + v(k) \end{cases} \tag{4}$$

Notice that if the controller's control signal keeps unchanged, the difference equations are ineffective since the estimated B will be eliminated by multiplying zero ($\Delta\vec{u}(k-1) = \vec{u}(k) - \vec{u}(k-1) = 0$). In this situation, we use the original nominal model to estimate $B(k)$'s value instead of the difference equations.

After introducing those compensation terms, it would be extremely time-consuming to perform the original Gaussian process parameter estimation. D^3 accelerates its estimation by using Kalman filter [40, 43], an more efficient linear quadratic estimation approach. Conceptually, Kalman filter accelerates Gaussian process by updating the system's state with the latest information only, instead of updating the state with all historical information. D^3's Kalman filter parameter estimation also follows an iterative manner as the Gaussian process parameter estimation. However, different from the Gaussian process, Kalman filter can give a more accurate normal distribution of the estimated parameters by filtering the effects of compensation terms (i.e., $a(i)$, $w(i)$ and $v(i)$).

4.3 Safety region quantification

Once the distribution of $B(i)$ is estimated, D^3 has to determine whether it suffers model deviation or not. We combine the nominal model listed in Equation 4 with a theoretical safety region and use a cumulative distribution function to calculate the probability that $B(i)$'s actual value exceeds the safety region. Unlike existing works that use a manual or empirical threshold, D^3's safety region represents the necessary condition under which the controller's behavior is guaranteed by control theory. As a result, D^3 would distinguish the uncollected normal execution from the abnormal executions and makes its detection more accurate.

It is the control theory, which helps developers design the original controllers, that determines the safety regions for the derived controllers. A controller's safety region can be derived by formal analysis [35, 37] or experimental study [30]. For example, Filieri et al. have carried on a formal assessment of the control properties of the designed SASs and give a safety region of the model parameter to avoid overshooting or long settling time. Formally, parameter B's safety region Θ_B can be defined as the interval between a lower bound Θ_B^L and an upper bound Θ_B^U.

Given a controller's safety region Θ_B, $B(i)$'s cross-border detection can still be inaccurate due to the estimation error of $B(i)$'s value. Most existing works combine the detection results in different adaptation loops to improve their accuracy, which inevitably increases their detection delay. D^3's estimated distributions of the parameter value synthesize the estimated values from different adaptation loops and enable us to avoid using multi-time detections.

D^3 uses a probability-based approach to detect B's deviation with a confidence interval CI. Specifically, suppose that the estimated $B(i)$'s distribution is subject to a normal distribution (i.e., $B(i) \sim N(\mu(i), P(i))$). The probability that $B(i)$'s value falls within the safety region can be calculated by a cumulative distribution function Φ for standard normal distribution. If the derived probability $\hat{p}(i)$ exceeds a confidence interval CI, D^3 will report model deviation and trigger the switching of adaptation.

By introducing probability-based detection with our previous nominal model, we further evolve our nominal model to its final form as follows.

$$\begin{cases} B(k) = B(k-1) + \omega \cdot (\tilde{B} - B(k-1)) + p(k) \\ \Delta\vec{x}(k) = A \cdot \Delta\vec{x}(k-1) + B \cdot \Delta\vec{u}(k-1) + \gamma \cdot \Delta a(k) + w(k) \\ \Delta y(k) = C \cdot \Delta\vec{x}(k) + v(k) \\ \hat{p}(k) = \Phi\left(\dfrac{\theta_B^U - \mu(k)}{\sqrt{P(k)}}\right) - \Phi\left(\dfrac{\theta_B^L - \mu(k)}{\sqrt{P(k)}}\right), B(k) \sim N(\mu(k), P(k)) \end{cases} \tag{5}$$

D^3 also uses an active detector to handle the situation when the original controller produces no control signal. Specifically, the active detector uses hypothesis testing to reveal possible model deviation. We use normal distribution to depict the variation of the managed system's output values when no control signal comes and detect model deviation by checking whether the measured variation is consistent with the pre-identified distribution.

5 EXPERIMENTS

In this section, we experimentally evaluate the effectiveness and usefulness of D^3 and D^3-based adaptation-supervision mechanism (and a Python implementation [1]) on three representative subject systems.

Our comparison baselines are SWDetecter [12] (the model deviation detection approach in the push-bottom method [23]) and LFM [16], which is one of the latest abnormal detection approaches for self-adaptive systems.

Particularly, we study the following two research questions:

RQ1 *Can D^3 timely and effectively detect model deviation for self-adaptive systems?*

RQ2 *How useful is D^3-based adaptation-supervision mechanism in terms of avoiding abnormal self-adaptation behavior?*

5.1 Experimental setup

Experimental subjects. We selected three prevalent subject systems from the community of self-adaptive systems. Each subject is provided with a control theory-derived controller and a mandatory controller, both of which are either given by the subject's developers or implemented following existing works.

SWaT, a water treatment testbed we described in Section 2. The original control theory-derived controller is inherited from the programmable logic controllers in [8]. The mandatory controller keeps the tank water level between upper and lower alarm levels locally.

RUBiS, a web auction system (studied in [11, 41, 50]) that can adapt to workload/network changes by adjusting the number of servers to satisfy quality-of-service levels. The original control theory-derived controller (from the push-button method [23]) is tuned for high system utilization and low response time. The mandatory controller guarantees in-time response and maintains affordable system utilization.

Video encoder, a video compression and streaming system (studied in [46–48]) that can adaptively change compression parameters to balance the throughout and video quality. The original control theory-derived controller (also from [23]) adjusts the compression parameters to achieve smooth and clear video stream. The mandatory controller keeps a no-lagging video stream by reducing the compression parameters accordingly.

Test configurations. We conducted the experiments on pre-defined test configurations and compare D^3 to SWDetecter and LFM over a series of *test configurations* ("configuration" for short). Given a subject system, each configuration is a simulated system run consisting of at most one model deviation. A configuration is *negative* if it contains no model deviation. Otherwise, a configuration that contains exactly one model deviation is categorized as *positive*.

To simulated model deviation in a configuration, we either change the subject's parameter values or change its environmental inputs, since we cannot directly manipulate the subject's controllability features (i.e., the value of parameter B). Changing the parameter values simulates the model deviation caused by discrete behavior, and changing the environmental inputs simulates the model deviation caused by inaccurate environmental interaction. How to change those settings is either according to existing works [16, 17] or based on our experience of the subject systems.

Specifically, a configuration is denoted by a quaternion of $(env_i, env_d, para_i, para_d, t)$, in which env_i and env_d denote the initial and deviated environmental input, $para_i$ and $para_d$ denote the initial and deviated model parameters, and t denotes the time point of model deviation. Initially, the subject and its running environment are reset with the initial parameter $para_i$ and env_i. At time point t, the model parameter is changed to $para_d$ and the environmental input is changed to env_d. For a negative configuration, $para_i = para_d$ and $env_i = env_d$ for the entire execution. For a positive configuration, $para_i/env_i$ is changed to a significantly different $para_d/env_d$ at time t.

[1] https://github.com/tongyanxiang/MoD2

For SWaT, we consider the model deviation caused by discrete behavior and inaccurate environmental interaction, respectively. We derive a total of 200 negative configurations and 400 positive configurations. The negative configuration uses the original settings in SWaT as the values of $para_i$ and $para_d$, and uses environment input extracted from SWaT's real-world testbed [8] as the values of env_i and env_d. The first 200 positive configurations share the same values of $para_i$, $para_d$ and env_i with the negative configurations. Their env_d values are determined by introducing manipulation disturbance to the system's valves. The last 200 positive configurations share the same values of env_i, env_d and $para_i$ with the negative configurations. Their $para_d$ values are determined by injecting attacks to the system's internal network [16]. For all positive configurations, we randomly generate their t values in an interval of $[0.2T, 0.6T]$, where T is the length of the system runs. Such interval enables us to study self-adaptation's performance in both normal and model-deviated situations. We denote the set of negative configurations as SWaT$^-$, the set of disturbance-based positive configurations as SWaT^+_D, and the set of attack-based positive configurations as SWaT^+_A.

For RUBiS, we consider the model deviation caused by discrete behavior only. We derive 200 negative configurations and 200 positive configurations. These configurations share the same environmental input, which is generated based on two real-world workloads WorldCup [6] and ClarkNet [7]. We use the original settings in RUBiS as the $para_i$ values for both negative and positive configurations. The values of $para_d$ of positive configurations are derived by introducing network jamming and delaying to RUBiS's service time. For positive configurations, we randomly generate their t values in an interval of $[0.2T, 0.6T]$, where T is the length of the input user workload. We denote the set of negative configurations as RUBiS$^-$, and the set of positive configurations as RUBiS$^+$.

For video encoder, we only consider the model deviation caused by inaccurate environmental interaction and also derive 200 negative configurations and 200 positive configurations. These configurations share the original settings in video encoder. We use the real-world surveillance video streams as env_i values for both negative and positive configurations. The values of env_d of positive configurations are derived by different kinds of real-world video streams (e.g., advertisement video or animation clip video). For positive configurations, we randomly generate their t values in an interval of $[0.2T, 0.6T]$, where T is the length of input mp4 steams. We denote the set of negative configurations as Encoder$^-$, and the set of positive configurations as Encoder$^+$.

Evaluation criteria. We measure the timeliness by the mean time delay (MTD, i.e., the average interval between the deviation point and the detection point of a positive configuration). Notice that we only consider the correctly-detected positive configurations in MTD.

We measure the accuracy by the false-negative rate (FN rate, i.e., the percentage of positive configurations that are falsely detected to be negative) and the false-positive rate (FP rate, i.e., the percentage of negative configurations that are falsely detected to be positive). Noticing that each of our positive configurations can be divided into a negative part and positive part, FP also accounts for the positive configurations that are falsely detected in their negative parts.

We measure the usefulness by the positive configuration's abnormal rate (i.e., the ratio of a subject system's abnormal operation time to its suffered model deviation time). The abnormal operation time is measured as the time during which the subject system violates any control properties (e.g., overshoot of the water level in SWaT) or fails to fulfill its mandatory requirements (e.g., overflow or underflow in SWaT). The model deviation time is measured as the subject system's execution time after we inject model deviation. Notice that a positive configuration may never violate any control property, as we discussed in Section 2 that model deviation is a necessary condition, not a sufficient condition, to the system's abnormal behavior.

Experimental procedure. We conducted all the experiments on an ECS server of Alibaba Cloud with 8 CPUs and 16GB of memory.

To answer **RQ1**, we compare three approaches' achieved MTD, FN rate, and FP rate on different configuration sets. We first compare the performance of D^3 and SWDetector on SWaT$^-$, SWaT^+_D, RUBiS$^-$, RUBiS$^+$, Encoder$^-$, Encoder$^+$. We then compare the performance of D^3, SWDetector, and LFM on SWaT^+_A, since LFM is proposed to address the challenges of attack-induced abnormal behavior only. The compared SWDetector is set with two different thresholds ($\theta = 3\sigma$ or $\theta = 6\sigma$), which is two of the suggested settings in [49].

We also study the impact of different settings in D^3, since we perform system identification for determining its parameters (i.e., γ, W, V). We compare the performance of D^3 with different amount of data for identification (20–100%, step of 20%).

To answer **RQ2**, we compare three subjects' abnormal rate on different configuration sets with and without D^3-based adaptation-supervision. We also implemented and evaluated two SWdetector-based adaptation-supervision mechanisms to study the role of the proposed D^3 in our mechanism. We measure the abnormal rate on SWaT^+_A, RUBiS$^+$, and Encoder$^+$ respectively.

5.2 Experiment Results

RQ1 (effectiveness) Table 1 compares the performance of D^3 and SWDetector for the three subjects. Considering detection timeliness, D^3's detecting time for model deviation varies in different subjects. The reported MTD is 39.25 seconds for SWaT, 131.10 seconds for RUBiS, and 0.76 seconds for video encoder. This variance is caused by the different lengths of these subjects' adaptation loops. For example, the length of an adaptation loop in RUBiS is 60 seconds, which results in a comparatively larger detection delay. When comparing with SWDetectors, D^3's achieved MTD is 140.30s smaller for the subjects and averagely 0.76s larger for the subject RUBiS (0.75s–0.76s). We notice that the two SWDetectors report smaller detection delay for subject video encoder. This is because our injected model deviation would produce a severe change of the managed system's output, which favors the SWDetector's monitoring on the output values.

When considering detection accuracy, D^3 achieves good FN and FP rate in detecting model deviation. Generally, the average FN rate of D^3 is 0.7% (0.0%–2.0%) and the average FP rate of D^3 is 1.0% (0.0%–3.0%). We notice that for SWaT and RUBiS, D^3's reported FN and FP rate are zero. In other words, D^3 neither missed detecting the model deviation in a positive configuration nor falsely reported

	SWaT$^-$ & SWaT$^+$			RUBiS$^-$ & RUBiS$^+$			Encoder$^-$ & Encoder$^+$		
	MTD(s)	FN(%)	FP(%)	MTD(s)	FN(%)	FP(%)	MTD(s)	FN(%)	FP(%)
D^3	39.25	0.0	0.0	131.10	0.0	0.0	0.76	2.0	3.0
SWDetector (θ=3σ)	274.92	94.0	93.3	361.24	3.0	8.5	0.00	40.0	46.5
SWDetector (θ=6σ)	OT	100.0	0.0	350.40	0.0	0.0	0.01	1.0	2.5

Table 1: Comparison of D^3 and SWDetector

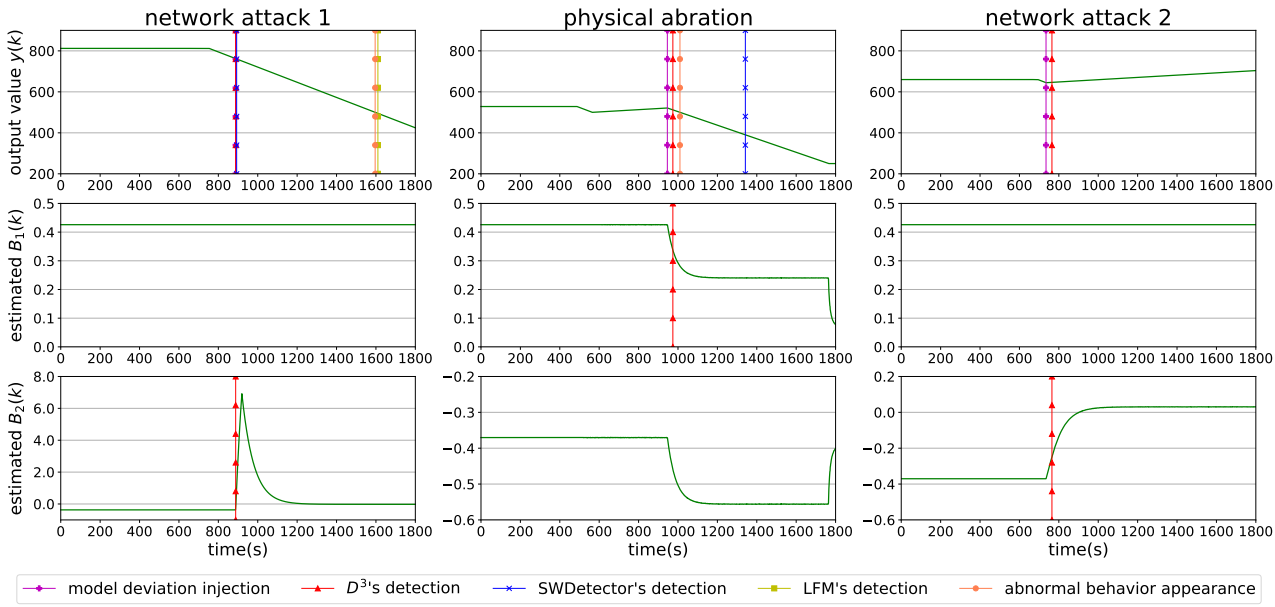

Figure 2: A case study on three positive configurations

an alarm in a negative configuration for these two subjects. For video encoder, D^3 reports a 3.0% FN rate and a 2.0% FP rate. This is caused by the similarity between the subject's input video stream and its output compressed stream, which makes our compensation terms less effective in describing their suffered measurement errors.

Comparing with SWDetectors of different settings, D^3's FN rate is averagely 39.0% lower (1.5%–97.0%), and its FP rate is averagely 24.1% lower (4.3%–46.7%). When considering the SWDetector of the best setting only (i.e., the window size ws is 28, and the threshold θ is 6σ), D^3's FN rate is averagely 33% lower (0.0%–100.0%), and its FP rate is averagely 0.1% higher (0.0%–0.5%). This result shows that SWDetector must sacrifice its FN rate in order to improve its FP rate, while D^3 balances these two metrics well.

	SWaT^+_A		
	MTD(s)	FN(%)	FP(%)
D^3	9.43	0.0	0.0
LFM	1600.03	0.5	0.0
SWDetector (θ=3σ)	84.60	1.0	84.0
SWDetector (θ=6σ)	1.06	37.5	0.0

Table 2: Comparison results of D^3, SWDetector and LFM

Table 2 gives the experimental results of the compared approaches on dataset SWaT^+_A (i.e., the positive configurations by injecting attacks to SWaT system). Basically, D^3 performs better than the other three approaches (including the SWDetectors of different settings) in terms of detection timeliness and detection accuracy. D^3 successfully detect all model deviations in a quite short time (averagely 9.43 seconds) with no false alarm. The learning-based LFM approach also achieves a good detection accuracy. However, it require much longer time (1600.03 seconds, 1590.60 seconds longer than D^3's) to detect model deviation. The reason for LFM's large detection delay is that the accuracy of the trained classifier should be above a threshold of 85% to avoid false alarm. For SWDetector with a better setting (i.e., $\theta = 6\sigma$), although it reports the shortest detection time (1.06 seconds) and no false alarm, it fails to detect 37.5% positive configurations, which is much larger than the other two compared approaches.

To better present D^3's effectiveness on detecting model deviation, we also perform a detailed study on the positive configurations that the baseline approaches fail to give a timely and accurate detection. We list three of those positive configurations in Figure 2. The first and the third configurations are derived by injecting network attacks to SWaT, and the last one is derived by injecting inaccurate environmental interaction. For each of the configurations, we list the measured output value $y(k)$ (in the first line chart) and D^3's estimated parameter values $B(k)$ (in the last two line charts, corresponding to the two arguments in $B(k)$). We use different marked lines to denote the time of model deviation injection, the time of D^3's detection, the time of SWDetector's detection, the time of LFM's detection, and the time of abnormal behavior appeared, respectively. For the time of D^3's detection, we also mark it on

	SWaT$^-$ & SWaT$^+$			RUBiS$^-$ & RUBiS$^+$			Encoder$^-$ & Encoder$^+$		
	MTD(s)	FN(%)	FP(%)	MTD(s)	FN(%)	FP(%)	MTD(s)	FN(%)	FP(%)
20%	39.26	0.0	0.0	131.40	0.0	0.0	0.77	2.0	3.0
40%	39.24	0.0	0.0	131.70	0.0	0.0	0.76	2.0	3.0
60%	39.23	0.0	0.0	132.60	0.0	0.0	0.77	2.0	3.0
80%	39.14	0.0	0.0	131.70	0.0	0.0	0.77	2.0	3.0
100%	39.25	0.0	0.0	131.10	0.0	0.0	0.77	2.0	3.0

Table 3: Comparison of D^3's performance with different amount of identification traces

	original	D^3-based	SWDetector-based (θ=3σ)	SWDetector-based (θ=6σ)
SWaT	14.0% (162.45s)	0.0% (0.00s)	10.6% (114.35s)	7.1% (71.32s)
RUBiS	61.1% (1953.00s)	6.3% (191.10s)	8.4% (300.90s)	7.5% (278.10s)
video encoder	16.2% (2.92s)	0.4% (0.09s)	6.5% (1.18s)	0.1% (0.02s)

Table 4: Comparison of the abnormal rates w/wo adaptation-supervision mechanisms

the corresponding $B_i(k)$'s line chart, indicating which argument exceeds its safety region.

For the first configuration, all of the three approaches successfully detect model deviation. However, LFM requires longer detection time since it has to combine multiple testing results to give an accurate detection. Such high detection delay makes LFM's detection less effective since SWaT's execution has already broken the control property (i.e., stability property) by the time LFM reports model deviation. For the second configuration, SWDetector's detection is too slow to be effective since the monitored output value shows no difference from the negative configuration in the early stages. For the third configuration, both SWDetector and LFM fail to detect model deviation. Their missing detections are reasonable since this model deviation has not caused severe consequence in SWaT's execution till the end of the execution trace. However, according to [16], this model deviation will eventually cause SWaT's abnormal behavior if the execution time extends from 30 minutes to 60 minutes.

Table 3 compares D^3's performance with different amount of traces for identification. Generally, changing the amount of identification traces has a very limited impact on D^3's effectiveness in detecting model deviation. The mean detection delay fluctuates in a very small range (39.14s–39.26s for SWaT, 131.10s–132.60s for RUBiS, and 0.76s–0.77s for video encoder) as the used amount of identification traces changes. D^3 with different amount of identification traces reports exactly the same FN and FP rate. D^3's stable performance shown in Table 3 reveals that its achieved performance is independent of the system identification procedure. In other words, it is our proposed three techniques, namely parameter deviation estimation, uncertainty compensation, and safety region quantification, that support D^3's timely and accurate model deviation detection.

In summary, the answer to our research question RQ1 is as follows. *D^3 can detect model deviation with low detection delay (average* 57.04 *seconds) and great accuracy (*0.7% *FN rate and* 1.0% *FP rate). Comparing with the baseline approaches, D^3 achieves smaller detection delay (average* 140.30 *seconds or* 71.1% *smaller), as well as a better accuracy (*39.0% *lower FN rate and* 24.1% *lower FP rate).*

RQ2 (usefulness) Table 4 compares the three subjects' abnormal rates on different positive configuration sets with different adaptation-supervision mechanisms. For the subjects with our D^3-based mechanism, the average abnormal rate is 2.2% (0.0%–6.3%). Comparing with the subjects without our mechanisms (denoted as "original"), the abnormal rates drop 28.2% (14.0%–54.8%) in average. This result validates the usefulness of our approach in alleviating the impact of model deviation. Particularly, our mechanism achieves a 0.0% abnormal rate for SWaT system (i.e., prevent all model-deviation-caused severe consequences), while the abnormal rate without our mechanism is 14.0%. Considering that SWaT's original controllers are carefully designed and implemented by the field experts, one cannot assume that the original controllers' robustness could handle all cases of model deviation.

We also list each subjects' abnormal operation time along with their suffered abnormal rates. The result shows that without our mechanism, each subject averagely suffers 706.12 seconds abnormal operation time in each configuration. In other words, the corresponding control-SASs fail to provide any guarantees on the subject systems' behavior in nearly 11.77 minutes. Notice that the execution we collected only lasts for 30 minutes, the aforementioned abnormal operation time could be longer if model deviation has not been addressed properly in these configurations.

Comparing with the two SWDetector-based mechanisms, the results reflect the importance of our D^3 in guarding the subjects' adaptation. Specifically, D^3-based mechanism achieves 4.5% lower abnormal rate (1.7%–8.9%), as well as 63.92 seconds shorter abnormal operation time. We believe it is D^3's timely and accurate model deviation detection that makes our mechanism more effective in protecting control-SASs.

In summary, the answer to our research question RQ2 is as follows. *Our D^3-based adaptation-supervision mechanism can alleviate the impact of model deviation on the managed system. With the support of our mechanism, the abnormal rate averagely drops by* 28.2% *comparing the original control-SASs and averagely drops by* 4.5% *comparing the SWDetector-based mechanism.*

5.3 Threats to Validity

One major concern on the validity of our empirical conclusions is the selection of evaluation subjects. We only use three subjects as the managed systems. This might harness the generalization of our conclusions. A comprehensive evaluation requires a full understanding of the managed systems, as well as their suitable

control theory-derived controllers. This requirement restricts our choice of possible experimental subjects. Nevertheless, we believe that our selected subjects are representative for their different platforms (including network system and cyber-physical systems) and architectures (including single controller and multiple controllers). Moreover, all of the selected subjects are widely-used by other self-adaptation researcher as their experimental subjects or motivating systems.

Another concern is about injecting model deviations in the positive configurations. Since we cannot directly manifest the subject system's controllability features, we can only modify its parameters or inputs to simulate model deviation for reproduction considerations. This might make our experimental settings less realistic. To address this problem, we carefully design the injected model deviations. For SWaT, the model deviation is based on the reported physical abrasion [8] and network attacks [16]. For RUBiS, the model deviation is designed according to reported failures in web service systems [3]. And for video encoder, we use real-world video streams to simulate the model deviation.

6 RELATED WORK

Control theory has been widely exploited to implement self-adaptive systems for their theoretical guarantees. Some pieces of related work use control-theoretical techniques to refine their architecture-based system model for the managed system. For example, Checiu et al. use a server request model to describe the behavior of an adaptive web service system [15]. Filieri et al. use discrete-time Markov chain model to depict service-oriented applications [22]. Some others use control-theoretical techniques to guide the design of the managing system. Konstantinos et al. [4] propose the CobRA framework for designing self-adaptive web-services, which uses model predictive control to guide the adaptation strategy of the managing system. Gabriel et al. [50] combine control-SASs and traditional architecture-based SASs by introducing discrete-time Markov chain in their PLA adaptation framework. Different from these pieces of work, the focus of D^3 is not designing the managing system but providing self-adaptation assurances in the presence of model deviation. In other words, our work can be regarded as an complementary for them by alarming their managing system the occurrence of model deviation.

Model deviation has received the attention of self-adaptation researchers since control-SASs emerged. According to control theory, the precision of the nominal model in control-SASs directly determines the effectiveness of the derived managing system. Baresi et al. propose using a grey-box discrete-time feedback controller to support robust self-adaptation that can overcome the light extent of model deviation [11]. Filieri et al. use continuous learning mechanisms to keep the nominal model updating at runtime [24]. Maggio et al. use Kalman filter to revise the identified nominal model by updating its state values [46]. Comparing with these pieces of works, our D^3-based mechanism concentrates on the model deviation that would cause the managing system to violate control properties, as well as the managed system behaving abnormally. Together with these works that address slight deviation of the model parameters, we can provide model-deviation-free self-adaptation for control-SASs.

The key component of our proposed mechanism is a timely and accurate detector for model deviation, which is very similar to the works on abnormal detection. Many research efforts have been made on abnormal detection from both control theory community and self-adaptation community. Window-based approach is the most-widely used abnormal detection approach among control theory researchers [31], which is very similar to the compared SWDetector approach in Section 5. For self-adaptation researchers, Jiang et al. derive invariants by observing messages exchanged between system components for robotic systems [36]. Chen et al. propose an SVM-based method to detect network attacks to SWaT system [16]. Qin et al. use context information to refine the derived invariants and combine multiple testing results to improve the accuracy of abnormal detection [53]. As we discussed in Section 2, the major difference between our D^3 and these approaches is that we directly estimate the values of nominal model's parameters instead of monitoring the system's output values. By doing so, we reduce the time delay for detecting model deviation.

Environmental uncertainty has always been a challenge for self-adaptation researchers. Most of these works focus on uncertainty's impact on the managed system's state. Esfahani et al. identify internal uncertainty and external uncertainty and propose a probability-based approach to assess both the positive and negative consequences of uncertainty [20]. Ghezzi et al. propose an adaptation framework to manifest non-functional uncertainty via model-based development [28]. Angelopoulos et al. also use Kalman filter to alleviate uncertainty's impact on the estimated states of the managed system's nominal model [4]. D^3's handling of uncertainty follows these pioneering works. For example, we introduce three compensate terms of measured environmental input, external uncertainty, and internal uncertainty in our nominal model and use Kalman filter to estimate an accurate $B(k)$ value. However, the focus of our D^3 is on the uncertainty's impact on the nominal model's parameters, which is addressed by our iterative estimation method.

7 CONCLUSION

In this paper, we present an adaptation-supervision mechanism for alleviating the impact of model deviation in control-SASs. The key to our mechanism is a novel detector, D^3, to detect model deviation timely and accurately. D^3 proposes and combines different techniques, including parameter deviation estimation, uncertainty compensation, and safety region quantification, to balance the detector's timeliness and accuracy. Once D^3 identifies the occurrence of model deviation, our mechanism switches from an control-based optimal controller to a conservative mandatory controller to alleviate the impact of model deviation. Our experimental evaluation shows the effectiveness of D^3 (71.1% shorter detection delay, 39.0% less FN and 24.1% FP) and the usefulness of D^3 adaptation-supervision mechanism (i.e., abnormal rate drop by 28.2%).

REFERENCES

[1] Sridhar Adepu and Aditya Mathur. 2016. An investigation into the response of a water treatment system to cyber attacks. In *Proceedings of 17th IEEE International Symposium on High Assurance Systems Engineering*. IEEE, 141–148.

[2] Cristina Alcaraz and Stephen Wolthusen. 2014. Recovery of structural controllability for control systems. In *International Conference on Critical Infrastructure Protection*. Springer, 47–63.

[3] Nadia Alshahwan and Mark Harman. 2011. Automated web application testing using search based software engineering. In *Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering*. IEEE, 3–12.

[4] Konstantinos Angelopoulos, Alessandro Papadopoulos, Vitor Silva Souza, and John Mylopoulos. 2016. Model predictive control for software systems with CobRA. In *Proceedings of the 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. IEEE, 35–46.

[5] Konstantinos Angelopoulos, Alessandro Papadopoulos, Vítor Silva Souza, and John Mylopoulos. 2018. Engineering self-adaptive software systems: From requirements to model predictive control. *ACM Transactions on Autonomous and Adaptive Systems* 13, 1 (2018), 1–27.

[6] Martin Arlitt and Tai Jin. 2000. A workload characterization study of the 1998 world cup web site. *IEEE Network* 14, 3 (2000), 30–37.

[7] Martin Arlitt and Carey Williamson. 1996. Web server workload characterization: The search for invariants. *ACM SIGMETRICS Performance Evaluation Review* 24, 1 (1996), 126–137.

[8] Kaung Myat Aung. 2015. Secure water treatment testbed (SWaT): an overview. *Singapore University of Technology and Design* (2015).

[9] Abhijit Badwe, Ravindra Gudi, Rohit Patwardhan, Sirish Shah, and Sachin Patwardhan. 2009. Detection of model-plant mismatch in MPC applications. *Journal of Process Control* 19, 8 (2009), 1305–1313.

[10] Saeid Barati, Ferenc Bartha, Swarnendu Biswas, Robert Cartwright, Adam Duracz, Donald Fussell, and et al. 2019. Proteus: Language and runtime support for self-adaptive software development. *IEEE Software* 36, 2 (2019), 73–82.

[11] Luciano Baresi, Sam Guinea, Alberto Leva, and Giovanni Quattrocchi. 2016. A discrete-time feedback controller for containerized cloud applications. In *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 217–228.

[12] Michele Basseville and Igor Nikiforov. 1993. *Detection of abrupt changes: theory and application*. Vol. 104.

[13] Mogens Blanke, Michel Kinnaert, Jan Lunze, Marcel Staroswiecki, and Jochen Schröder. 2006. *Diagnosis and fault-tolerant control*. Vol. 2. Springer.

[14] Yuriy Brun, Giovanna Di Marzo Serugendo, Cristina Gacek, Holger Giese, Holger Kienle, Marin Litoiu, and et al. 2009. Engineering self-adaptive systems through feedback loops. In *Software Engineering for Self-Adaptive Systems*. Springer, 48–70.

[15] Laurentiu Checiu, Bogdan Solomon, Dan Ionescu, Marin Litoiu, and Gabriel Iszlai. 2011. Observability and controllability of autonomic computing systems for composed web services. In *Proceedings of the 6th IEEE International Symposium on Applied Computational Intelligence and Informatics*. IEEE, 269–274.

[16] Yuqi Chen, Christopher Poskitt, and Jun Sun. 2018. Learning from mutants: Using code mutation to learn and monitor invariants of a cyber-physical system. In *Proceedings of the 39th IEEE Symposium on Security and Privacy*. IEEE, 648–660.

[17] Zhikai Chen, Lingxi Xie, Shanmin Pang, Yong He, and Qi Tian. 2021. Appending adversarial frames for universal video attack. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 3199–3208.

[18] OW2 Consortium et al. 2008. Rubis: Rice university bidding system. *URL http://rubis. ow2. org* (2008).

[19] Mário de Castro and Ignacio Vidal. 2019. Bayesian inference in measurement error models from objective priors for the bivariate normal distribution. *Statistical Papers* 60, 4 (2019), 1059–1078.

[20] Naeem Esfahani, Ehsan Kouroshfar, and Sam Malek. 2011. Taming uncertainty in self-adaptive software. In *Proceedings of the 19th ACM SIGSOFT Symposium on the Foundations of Software Engineering and 13th European Software Engineering Conference*. 234–244.

[21] Oreste Fecarotta, Riccardo Martino, and Cristina Morani. 2019. Wastewater pump control under mechanical wear. *Water* 11, 6 (2019), 1210.

[22] Antonio Filieri, Carlo Ghezzi, Alberto Leva, and Martina Maggio. 2011. Self-adaptive software meets control theory: A preliminary approach supporting reliability requirements. In *Proceedings of 26th IEEE/ACM International Conference on Automated Software Engineering*. IEEE, 283–292.

[23] Antonio Filieri, Henry Hoffmann, and Martina Maggio. 2014. Automated design of self-adaptive software with control-theoretical formal guarantees. In *Proceedings of the 36th International Conference on Software Engineering*. 299–310.

[24] Antonio Filieri, Henry Hoffmann, and Martina Maggio. 2015. Automated multi-objective control for self-adaptive software design. In *Proceedings of the 10th Joint Meeting on Foundations of Software Engineering*. 13–24.

[25] Antonio Filieri, Martina Maggio, Konstantinos Angelopoulos, Nicolás D'Ippolito, Ilias Gerostathopoulos, Andreas Berndt Hempel, and et al. 2015. Software engineering meets control theory. In *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. IEEE Press, 71–82.

[26] Antonio Filieri, Martina Maggio, Konstantinos Angelopoulos, Nicolás D'ippolito, Ilias Gerostathopoulos, Andreas Hempel, and et al. 2017. Control strategies for self-adaptive software systems. *ACM Transactions on Autonomous and Adaptive Systems* 11, 4 (2017), 1–31.

[27] David Garlan, S-W Cheng, A-C Huang, Bradley Schmerl, and Peter Steenkiste. 2004. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer* 37, 10 (2004), 46–54.

[28] Carlo Ghezzi, Leandro Sales Pinto, Paola Spoletini, and Giordano Tamburrelli. 2013. Managing non-functional uncertainty via model-driven adaptivity. In *Proceedings of the 35th International Conference on Software Engineering*. IEEE, 33–42.

[29] Graham Goodwin, Stefan Graebe, and Mario Salgado. 2001. *Control system design*.

[30] Jorn Gruber, Daniel Ramirez, Teodoro Alamo, and Eduardo Camacho. 2011. Min–Max MPC based on an upper bound of the worst case cost with guaranteed stability. *Journal of Process Control* 21, 1 (2011), 194–204.

[31] Fredrik Gustafsson and Fredrik Gustafsson. 2000. *Adaptive filtering and change detection*. Vol. 1. Citeseer.

[32] Zhijian He, Yao , Enyan Huang, Qixin Wang, Yu Pei, and Haidong Yuan. 2019. A system identification based oracle for control-cps software fault localization. In *Proceedings of the 41st International Conference on Software Engineering*. IEEE, 116–127.

[33] Joseph Hellerstein, Yixin Diao, Sujay Parekh, and Dawn Tilbury. 2004. *Feedback control of computing systems*. Wiley Online Library.

[34] Emilio Incerto, Mirco Tribastone, and Catia Trubiani. 2017. Software performance self-adaptation through efficient model predictive control. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*. IEEE, 485–496.

[35] Rolf Isermann. 2013. *Digital control systems*. Springer Science & Business Media.

[36] Hengle Jiang, Sebastian Elbaum, and Carrick Detweiler. 2013. Reducing failure rates of robotic systems though inferred invariants monitoring. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 1899–1906.

[37] Michael Johnson and Mohammad Moradi. 2005. *PID control*. Springer.

[38] Raphael Jungers, Atreyee Kundu, and Maurice Heemels. 2017. Observability and controllability analysis of linear systems subject to data losses. *IEEE Trans. Automat. Control* 63, 10 (2017), 3361–3376.

[39] Eunsuk Kang, Sridhar Adepu, Daniel Jackson, and Aditya Mathur. 2016. Model-based security analysis of a water treatment system. In *Proceedings of the 2nd International Workshop on Software Engineering for Smart Cyber-Physical Systems*. IEEE, 22–28.

[40] Youngjoo Kim and Hyochoong Bang. 2018. Introduction to Kalman filter and its applications. *Introduction and Implementations of the Kalman Filter* 1 (2018), 1–16.

[41] Cristian Klein, Martina Maggio, Karl-Erik Årzén, and Francisco Hernández-Rodriguez. 2014. Brownout: Building more robust cloud applications. In *Proceedings of the 36th International Conference on Software Engineering*. 700–711.

[42] Juš Kocijan, Agathe Girard, Blaž Banko, and Roderick Murray-Smith. 2005. Dynamic systems identification with Gaussian processes. *Mathematical and Computer Modelling of Dynamical Systems* 11, 4 (2005), 411–424.

[43] Qiang Li, Ranyang Li, Kaifan Ji, and Wei Dai. 2015. Kalman filter and its application. In *Proceedings of the 8th International Conference on Intelligent Networks and Intelligent Systems*. IEEE, 74–77.

[44] Dan Ling, Ying Zheng, Hong Zhang, Weidong Yang, and Bo Tao. 2017. Detection of model-plant mismatch in closed-loop control system. *Journal of Process Control* 57 (2017), 66–79.

[45] Marin Litoiu, Mary Shaw, Gabriel Tamura, Norha Villegas, Hausi Müller, Holger Giese, and et al. 2017. What can control theory teach us about assurances in self-adaptive software systems? In *Software Engineering for Self-Adaptive Systems III. Assurances*. Springer, 90–134.

[46] Martina Maggio, Alessandro Vittorio Papadopoulos, Antonio Filieri, and Henry Hoffmann. 2017. Automated control of multiple software goals using multiple actuators. In *Proceedings of the 11th Joint Meeting on Foundations of Software Engineering*. 373–384.

[47] Martina Maggio, Alessandro Vittorio Papadopoulos, Antonio Filieri, and Henry Hoffmann. 2017. Self-adaptive video encoder: Comparison of multiple adaptation strategies made simple. In *Proceedings of 12th IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. IEEE, 123–128.

[48] Claudio Mandrioli and Martina Maggio. 2020. Testing self-adaptive software with probabilistic guarantees on performance metrics. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1002–1014.

[49] Douglas Montgomery. 2020. *Introduction to statistical quality control*. John Wiley & Sons.

[50] Gabriel A Moreno, Javier Cámara, David Garlan, and Bradley Schmerl. 2015. Proactive self-adaptation under uncertainty: a probabilistic model checking approach. In *Proceedings of the 10th Joint Meeting on Foundations of Software Engineering*. 1–12.

[51] Alexander Palm, Andreas Metzger, and Klaus Pohl. 2020. Online reinforcement learning for self-adaptive information systems. In *Proceedings of the 32nd International Conference on Advanced Information Systems Engineering*. Springer, 169–184.

[52] Tharindu Patikirikorala, Alan Colman, Jun Han, and Liuping Wang. 2012. A systematic survey on the design of self-adaptive software systems using control engineering approaches. In *Proceedings of the 7th International Symposium on*

Software Engineering for Adaptive and Self-Managing Systems. IEEE, 33–42.

[53] Yi Qin, Tao Xie, Chang Xu, Angello Astorga, and Jian Lu. 2019. CoMID: Context-Based Multiinvariant Detection for Monitoring Cyber-Physical Software. *IEEE Transactions on Reliability* 69, 1 (2019), 106–123.

[54] Mazeiar Salehie and Ladan Tahvildari. 2012. Towards a goal-driven approach to action selection in self-adaptive software. *Software: Practice and Experience* 42, 2 (2012), 211–233.

[55] Stepan Shevtsov, Mihaly Berekmeri, Danny Weyns, and Martina Maggio. 2017. Control-theoretical software adaptation: A systematic literature review. *IEEE Transactions on Software Engineering* 44, 8 (2017), 784–810.

[56] Stepan Shevtsov and Danny Weyns. 2016. Keep it simplex: Satisfying multiple goals with guarantees in control-based self-adaptive systems. In *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 229–241.

[57] Stepan Shevtsov, Danny Weyns, and Martina Maggio. 2019. SimCA*: A Control-theoretic approach to handle uncertainty in self-adaptive systems with guarantees. *ACM Transactions on Autonomous and Adaptive Systems* 13, 4 (2019), 1–34.

[58] Silvio Simani, Cesare Fantuzzi, and Ronald Jon Patton. 2003. Model-based fault diagnosis techniques. In *Model-based Fault Diagnosis in Dynamic Systems Using Identification Techniques*. Springer, 19–60.

[59] Guojun Wang and Siva Sivaganesan. 2013. Objective priors for parameters in a normal linear regression with measurement error. *Communications in Statistics-Theory and Methods* 42, 15 (2013), 2694–2713.

[60] Xiaoyun Zhu, Mustafa Uysal, Zhikui Wang, Sharad Singhal, Arif Merchant, Pradeep Padala, and et al. 2009. What does control theory bring to systems research? *ACM SIGOPS Operating Systems Review* 43, 1 (2009), 62–69.