Design and Validation of Fault-tolerant Embedded Controllers

Saurav Kumar Ghosh[1], Soumyajjit Dey[1], Dip Goswami[2], Daniel Mueller-Gritschneder[3], Samarjit Chakraborty[3]
[1]Indian Institute of Technology Kharagpur, [2]Eindhoven University of Technology, [3]TU Munich
{saurav.kumar.ghosh,soumya}@cse.iitkgp.ernet.in, d.goswami@tue.nl, {daniel.mueller,samarjit}@tum.de

***Abstract*—Embedded control systems are an important and often safety-critical class of applications that need to operate reliably even in the presence of faults. We show that intermittent fault scenarios caused by wear-out effects due to a higher density and a smaller geometry of the embedded electronic components may become a reliability concern for real-time embedded control applications. To mitigate the effects of such intermittent faults, we propose a novel fault-tolerant controller design method such that the resulting controllers ensure closed loop stability (i.e., guarantee safety) with only possibly degraded performance under such fault scenarios.**

In order to measure the amortized performance offered by the software implementations of such fault-tolerant controllers, we provide a program analysis methodology that statically estimates the quality of control guaranteed by the C code implementation of the fault-tolerant control law. This combination of fault-tolerant controller design followed by performance feedback computed using a formal analysis is illustrated with a case study from the automotive domain.

I. Introduction

As feature sizes of semiconductor technologies shrink, faults in the hardware of the embedded computing platform become a rising threat to the correct execution of embedded applications [1]. For example, it is foreseeable that electronic control units (ECUs) in cars will use processors in the near future that are susceptible to intermittent faults. This situation will be aggravated by the fact that such ECUs might be subjected to extreme temperatures and electromagnetic radiations and cannot be equipped with active cooling mechanisms. Feedback control applications running on such ECUs often implement safety critical functionality (e.g., brake, autopilot), which should remain operational under all circumstances. Thus, the design of the control law and the computation platform must assure correct functionality even in the presence of faults.

Intermittent faults are caused by wear out effects such as semiconductor aging degradation due to Negative Bias Temperature Instability (NBTI), Hot Carrier Injection (HCI) or Time-dependent Dielectric Breakdown (TDDB) [1]. In domains where cooling features cannot be provided (e.g., automotive), the thermally induced faults are dramatically rising. They indicate that a certain part of the hardware is about to fail permanently. Intermittent faults can become extremely critical for the execution of safety-critical control applications. They are detected by integrating built-in checker hardware or by regularly executing checker software on the embedded platform. In this work, we consider a computation platform (illustrated in Fig. 1) with dual modular redundancy (DMR). The input is replicated, the control signal is computed on both compute units and the outputs are compared by a checker module. If an intermittent fault affects the computation of the control signal, the output of the compute units will differ. In this case, the computed control signal is not forwarded to the actuators and the old control signal is held.

The work of S. K. Ghosh is supported by a TCS Research Fellowship. This work is also partially supported by the H2020 project I-MECH (Intelligent Motion Control Platform for Smart Mechatronic Systems)

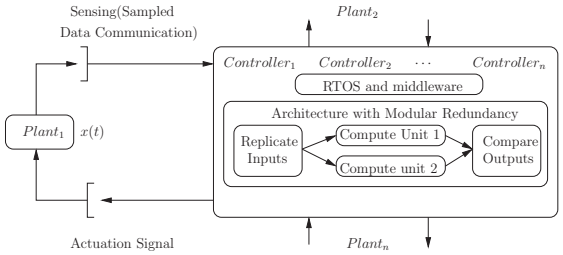

Fig. 1. Embedded control systems with DMR.

Contributions: In this work, we design fault-tolerant control laws [2] that guarantee stability and control performance on the type of computation platform shown in Fig. 1 in the face of intermittent faults. Our contributions are as follows.

We evaluate the implication of intermittent faults on platforms like Fig. 1 and estimate the probability that hardware faults impact the execution of a control algorithm. Given such quantitative knowledge about the impact of faulty behavior of the hardware on the execution of control laws, we propose a fault-tolerant controller design method such that resultant controllers are robust against such quantified intermittent faults. Such fault-tolerant controllers take into account the derived intermittent fault model to ensure closed loop stability (i.e., *safety*), albeit with degraded performance compared to non-faulty scenarios. Our method imposes minimal timing/cost overhead by suitably designing appropriate control algorithm, which only requires to adapt the software implementation without adding further hardware redundancy.

We formally validate the actual performance guarantee offered by the fault-tolerant controller under faulty executions. While the control theoretic performance guarantees are mathematically verifiable for the underlying control law at the time of design, such guarantees do not really hold true for the actual implementation [3]. Also, there does not exist any standard methodology which may formally verify performance guarantees offered by a fault-tolerant controller over the actual program implementation of the controller software executing on an unreliable hardware. Our methodology performs a source level formal analysis of the fault-tolerant software controller implementation and estimates the risk involved in deploying the implementation on an unreliable hardware. Based on these estimates, we can synthesize alternate fault-tolerant controllers when the analysis reports unacceptable levels of performance degradation.

Overall scheme: The proposed scheme is composed of three components: (i) a parameterized intermittent fault model that

captures the failure behavior of the underlying hardware with a high probality – Section III. (ii) a controller design which is tailored to guarantee stability for the derived fault model (i) – Section IV. (iii) a source code level formal analysis of the fault-tolerant control software implementation of (ii) to guarantee a given performance requirement – Section V. The guarantees on stability and performance from (ii) and (iii) hold as long as fault model (i) is satsified by the underlying hardware (which has a sufficiently high probability). Essentially, it is similar to design risk involved with failure probability (usually, very low) of system components (e.g., sensors, actuators) which is common in real-life.

II. Control laws as Software

Let $\mathcal{P} = (A, B)$ be a discrete, linear time invariant (LTI) plant derived from a continuous time dynamics with sampling period h. The state-space equation of $\mathcal{P}$ can be given as $x(t+1) = Ax(t)+Bu(t)$. Following usual notations, the plant state at the t-th time instant is given by the vector $x(t)$ and $u(t)$ defines the control input at the t-th time instant. A stabilizing feedback controller K_i for $\mathcal{P}$ senses the state vector of the plant x and decides the control action by computing the value of control variables in u following some state feedback control law, $u(t) = K_i x(t)$. The dynamics of the resulting closed loop system is obtained as $x(t+1) = (A + BK_i)x(t) = A_i x(t)$. In that way, for a given set $\{K_i \,|\, i \in I \triangleq \{1, 2, \cdots, n\}\}$ of independent stabilizing controllers for $\mathcal{P}$, we may obtain a set $\{A_i \,|\, i \in I \triangleq \{1, 2, \cdots, n\}\}$ of closed loop dynamic matrices. We consider a *control software implementation* as an imperative program $\mathcal{T}$ defined over the state variables $\in x$ executing over a computation platform with dual modular redundancy illustrated in Fig. 1. $\mathcal{T}$ is considered as a loop free program where switching conditions among K_i-s are expressed using `if-then-else` constructs. The program computes updates of the control action u using a set $\{K_i \,|\, i \in I\}$ of control matrices as defined earlier. We now formalize the intermittent fault model affecting the control law computation.

III. Modeling faults and control errors

Studies on wear out failures [1] reveal that the duration w between activations of intermittent faults can be modeled with an Exponential distribution given by $CDF_w^{EXP}(w, \beta) = 1 - e^{\frac{w}{\beta}}$ where β is the Mean Time Between Activations (MTBA). However, not every activation of a fault corrupts the computation being implemented by the software [4]. In the scope of this paper, a fault causes a control error, whenever it corrupts the output of the control software. In this case, the voter in the DMR platform as depicted in Fig. 1 will block the command to be written to the actuator. Given an intermittent fault model, we can calculate the probability of computation error P_{error} in any control loop iteration (length is the sampling period h) as $P_{error} = CDF_{EXP,w}(h, \beta)(1 - P_{mc})$ where P_{mc} is the probability that the fault gets masked. The masking probability P_{mc} for a computation platform can be derived through extensive fault injection campaigns carried out on actual hardware or representative RTL simulation setups. However, the details of such fault injection campaigns are out of the scope of the current work.

There are two corner cases for the MTBA β of intermittent fault scenarios [4]. For very large values of the MTBA β, the intermittent faults are rare, i.e., similar to transient faults. Thus, the time between faults affecting the control loop is very long. Because of the inherent ability of feedback loops to reject disturbances, such intermittent faults will not cause instability. In contrast, for very low values of β, i.e., with intermittent faults happening very frequently similar to permanent errors, the faulty compute unit can be quickly detected by any error detection module before endangering stability and is switched off. Thus, the redundancy is removed, and the remainder correct compute unit is used to move the system to a safe state as soon as possible for repair. In the scope of this work we assume that β ranges between [1, 100] ms. Above 100 ms, the error can be treated as transient, below 1 ms the error is detected and handled similar to a permanent error. In this light, we modify our fault probability P_{error} computation as

$$P_{error} = \frac{1}{99}(1 - P_{mc}) \int_1^{100} CDF_{EXP,w}(h, \beta) \mathrm{d}\beta. \quad (1)$$

It should be noted that the computation of the control action can take (actually does take) much less time than the sampling period h. But, intermittent faults can affect the stored value of the control action in memory registers in a similar manner as it affects the actual computation. Hence, we calculate the probability of intermittent faults causing erroneous control law computations over the total sampling period h.

We now adapt this intermittent fault model in closed control loop iterations. One or more continuous erroneous closed loop executions are referred to as an *error period*. Let m denote the number of such consecutive control errors, i.e., the length of an error period. We refer to the number of error-free sampling periods following an error period as an *error recovery period*. Let n be the minimum number of error-free executions after any error period, i.e. the length of error recovery period ($>$ sampling period by definition).

Based on the probability of error in each iteration as computed in (1), it is possible to select a maximum value M for the control design such that only error periods shorter or equal to $m \leq M$ are regularly encountered in any error scenario. For this, M must be selected such that error periods with $m > M$ appear only with very low probability. During run time, the exact length of a specific error period is known because it can be monitored. In contrast, the length of the recovery period is unknown because the occurrence of the next control error happens randomly in the future. For n, we must assure stability in the case of recovery period with minimal length $n \geq N$. Hence, the intermittent fault model can be summarized as follows:

- The maximum length of an *error period* is M. i.e. there can be maximum M consecutive control errors.
- The minimum length of *error recovery period* is N. i.e. any error period is followed by a minimum N error-free control loop executions.

Our proposed fault-tolerant controller design methodology

uses these as design parameters.

IV. Fault-tolerant Controller Design

Having formalized our fault model, we now present the design of a fault-tolerant controller under the assumption that, the number consecutive erroneous control loops will be less than M and any such error period will be followed by a minimum of N error-free control loop executions with the following design objectives:

O1: The closed-loop system meets the *stability* and *performance* requirements in the non-faulty scenarios.

O2: The closed-loop system should be *stable* under *all* circumstances (even in faulty scenarios).

O3: In the faulty scenarios, the performance can potentially degrade without violating some minimum performance requirement.

A. Control Application at Error Periods

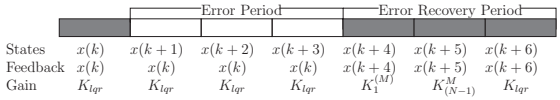

Fig. 2. Fault Scenario with M=3 and N=3.

At the occurrence of errors, the control system does not forward the computed control law to the actuators. Therefore, the old control input based on the older feedback signal is applied to the plant. Thus, in an erroneous execution, the control law becomes

$$u(k) = Kx(k-1). \tag{2}$$

For subsequent occurrence of erroneous executions in a single error period, the control law (2) gets further delayed,

$$u(k) = Kx(k-i), \tag{3}$$

where $1 < i \leq M$. Fig. 2 shows an example with three consecutive erroneous executions, i.e., $i = 3$. Note that for a given erroneous execution sequence (as shown in Fig. 2), the control law (3) can also be described as,

$$u(k+i) = Kx(k). \tag{4}$$

With a single erroneous execution and control law (2), the closed-loop system becomes,

$$x(k+2) = Ax(k+1) + BKx(k). \tag{5}$$

Hence, we have a closed-loop system,

$$x(k+2) = [A^2 + (AB+B)K]x(k). \tag{6}$$

A generalized form of system closed-loop dynamics with an error period of i consecutive errors is given by,

$$x(k+i+1) = [A^{i+1} + (A^iB + A^{i-1}B \cdots + B)K]x(k), \tag{7}$$

where $1 < i \leq M$. Thus, we have $(M+1)$ (sub)systems: one system for error-free execution and M systems for various categories of error periods. All these switching subsystems are modeled by (7) with $0 < i \leq M$. Depending on the occurrence of errors these $(M+1)$ systems *switch* among themselves with time. Such *arbitrary* switching can potentially result in instability [5]. Essentially, the major challenge in fault-tolerant design is to make sure that such switching does not lead to an unstable system, i.e., to a safety failure.

One of the strongest possible ways to ensure stability of a switched systems is to design the switching systems such that a common quadratic Lyapunov function (CQLF) exists among all the $(M+1)$ systems [6]. In this work, we propose a novel design method for the feedback gains that ensure the existence of a CQLF among the switching systems resulting from various error periods.

B. Background: CQLF

Consider the discrete-time switching LTI (sub)systems,

$$x(k+1) = A_i x(k) \tag{8}$$

where $i \in \{1, 2, 3...\}$ and the A_i are *stable*. That is, $x(k) \to 0$ as $k \to \infty$. Discrete-time LTI systems (8) are stable iff all eigenvalues of matrix A_i lie inside the *unit* circle (or magnitude less than unity).

Theorem IV.1. (Discrete-time Lyapunov equation [7]) *Let $A_i \in \mathbb{R}^{n\times n}$. If there exist $P = P^T > 0$, $Q = Q^T > 0$ satisfying $A_i^T P A_i - P = -Q$, then A_i is stable.*

Theorem IV.2. ([5], [6]) *Consider A_i to be discrete-time LTI systems of the form (8). $V(x) = x^T Px$ is the CQLF of the systems A_i if there exist $P = P^T > 0$, $Q = Q^T > 0$ and P is the simultaneous solution of the discrete-time Lyapunov equation, $A_i^T P A_i - P = -Q < 0$. The existence of a CQLF is a necessary and sufficient condition for the stability of the system with switching subsystems.*

In the fault model described in Section III, the error period with $i\,(i \leq M)$ consecutive faulty executions will be followed by a minimum of N error-free executions (see Fig. 2). We exploit this fact to design several error recovery gains for N post-error control inputs (or executions) such that the overall closed-loop system recovers and ensures stability. The challenge is to ensure stability in presence of switching due to the error periods. The identification of switching *subsystems* is an important aspect in the design and analysis of such systems. The occurrence of these errors can be modeled by considering arbitrary switching between two subsystems (see Fig. 2): (i) S_i consisting of i erroneous executions and $(N-1)$ error-free executions in the error recovery period. (ii) S_{nf} with one error-free execution at the end of error recovery period. As long as switching between S_i and S_{nf} is stabilized, we can guarantee safe operation in the presence of such error periods.

In an error-free execution we use feedback gain K_{lqr} designed using LQR [8] method. As indicated in Fig. 2, the system continues to use the LQR gain K_{lqr} in the error period. From (7), we obtain,

$$x(k+i+1) = A_F^{(i)} \cdot x(k). \tag{9}$$

where $A_F^{(i)} = [A^{i+1} + (A^iB + A^{i-1}B \cdots + B)K_{lqr}]$.
Next, in the error recovery period, we first consider the following control law with the old feedback signal $x(k)$,

$$u(k+i+1) = K_r^{(i)} \cdot x(k). \tag{10}$$

With control law (10), we have,

$$\begin{aligned} x(k+i+2) &= Ax(k+i+1) + Bu(k+i+1), \\ &= AA_F^{(i)}x(k) + BK_r^{(i)}x(k) \\ &= [AA_F^{(i)} + BK_r^{(i)}]x(k). \end{aligned} \tag{11}$$

Similarly, in the subsequent error-free executions in the error recovery period, we have,

$$\begin{aligned} x(k+i+3) &= [A^2A_F^{(i)} + (AB+B)K_r^{(i)}]x(k), \\ x(k+i+4) &= [A^3A_F^{(i)} + (A^2B + AB + B)K_r^{(i)}]x(k), \cdots \\ x(k+i+N) &= [A^{N-1}A_F + (A^{N-2}B \cdots AB + B)K_r^{(i)}]x(k) \end{aligned} \tag{12}$$

Therefore, we have,

$$S_i : [A_N + B_N K_r^{(i)}] \text{ and } S_{nf} : A + BK_{lqr} = A_{cl} \quad (13)$$

where $A_N = A^{N-1}A_F^{(i)}$ and $B_N = A^{N-2}B \cdots AB + B$

The following theorem states the design of feedback gain $K_r^{(i)}$ such that S_i and S_{nf} have a CQLF.

Theorem IV.3. (Design of $K_r^{(i)}$) *Consider the subsystems S_i and S_{nf} in (13). If there exist $Y = Y^T > 0$ and Z such that the following LMIs hold,*

$$\begin{bmatrix} Y & YA_N^T + Z^T B_N^T \\ A_N Y + B_N Z & Y \end{bmatrix} > 0, \qquad A_{cl}^{-1} Y - Y A_{cl}^T > 0. \quad (14)$$

then S_i and S_{nf} have a CQLF with the feedback gain $K_r^{(i)} = ZY^{-1}$.

Proof. By pre- and post-multiplying (14) by $\begin{bmatrix} Y^{-1} & 0 \\ 0 & Y^{-1} \end{bmatrix}$, we obtain

$$\begin{bmatrix} Y^{-1} & A_N^T Y^{-1} + Y^{-1} Z^T B_N^T Y^{-1} \\ Y^{-1}(A_N + B_N Z Y^{-1}) & Y^{-1} \end{bmatrix} > 0.$$

Further, substituting $P = Y^{-1}, K_r^{(i)} = ZY^{-1}$, we obtain,

$$\begin{bmatrix} P & (A_N + B_N K_r^{(i)})^T P \\ P(A_N + B_N K_r^{(i)}) & P \end{bmatrix} > 0. \quad (15)$$

Equation (15) gives

$$P - (A_N + B_N K_r^{(i)})^T P (A_N + B_N K_r^{(i)}) > 0. \quad (16)$$

We replace Y in equation (14) and obtain,

$$A_{cl}^{-1} P^{-1} - P^{-1} A_{cl}^T > 0 \rightarrow A_{cl}^{-1} - P^{-1} A_{cl}^T P > 0$$
$$\rightarrow P A_{cl}^{-1} - A_{cl}^T P > 0 \rightarrow P - A_{cl}^T P A_{cl} > 0 \quad (17)$$

From (17) and (16), it is clear that S_i and S_{nf} have a CQLF and this completes the proof. □

The above theorem shows the design of controller $K_r^{(i)}$ with constant control input,

$$u(k + i + n) = K_r^{(i)} \cdot x(k) \quad (18)$$

where $1 \leq n \leq (N-1)$. That is, no updated feedback $x(k + i+n)$ is utilized in the error recovery period in the control law. In the following, we present a gain transformation to utilize the latest feedbacks. Using Equation (9), we have,

$$u(k + i + 1) = K_r^{(i)} \cdot x(k) = K_r^{(i)} (A_F^{(i)})^{-1} x(k + i + 1) \quad (19)$$

Equation (19) gives us the transformed gain,

$$K_1^{(i)} = K_r^i (A_F^{(i)})^{-1}.$$

Similarly, using (11) and (12), we obtain the transformed gains in the other error-free executions in the error recovery period,

$$K_2^{(i)} = K_r^{(i)} [A A_F^{(i)} + B K_r^{(i)}]^{-1},$$
$$K_3^{(i)} = K_r^{(i)} [A^2 A_F^{(i)} + (AB + B) K_r^{(i)}]^{-1}, \cdots \quad (20)$$
$$K_{N-1}^{(i)} = K_r^{(i)} [A^{N-2} A_F^{(i)} + (A^{N-3} B \cdots AB + B) K_r^{(i)}]^{-1}.$$

In summary, in an error period with i consecutive erroneous executions, we propose to use K_{lqr} in the erroneous executions and controllers $K_j^{(i)}$ with $1 \leq j \leq N-1$ are used in the first $(N-1)$ error-free executions in the error recovery period.

With M categories of error periods, we further need to consider arbitrary switching between systems: (i) S_i in (13) for $1 < i \leq M$ and, (ii) S_{nf} in (13). With the above design of fault-tolerant controllers for each category of error period, the switching between S_i and S_{nf} are taken care of. Further, it should be noted that the choice of subsystems (see Fig. 2) ensures that there is no switching between S_is – switching only happens between S_i and S_{nf}. Since the above controller guarantees switching stability between S_i and S_{nf}, the individual controller for each i and $1 < i \leq M$ guarantees stability of the overall system.

While we guarantee stability of the system under error scenarios by the design of the proposed fault-tolerant controller, the design does not ascertain the quality of control QoC of the fault-tolerant controller with respect to a typical performance requirement such as settling time, e.g., $x_1 \leq 4$ within $2sec$. A testing based approach for simulating the closed loop for all possible fault scenarios and all possible initial states is also infeasible. To this end, we provide a toolflow based on 'backward reachability analysis' [9] and 'volume computation' of convex polyhedra [10] which gives a safe probabilistic estimate of QoC for the actual software implementation.

We consider the *performance criteria* for a discrete time closed loop system to be given in the form of a *linear* relation defined over some set $\{x_1, \cdots, x_n\}$ of state variables with a timing threshold provided in terms of number of closed loop iterations. More specifically, we restrict our attention to performance constraints of the form $\wedge_i x_i(l) < c_i$ for $i \in [1, n]$ where c_i is a constant. Though limited in expressiveness, these template of constraints is capable of capturing a wide class of real world control performance constraints.

For specifications with real time thresholds, the corresponding number of closed loop iterations are extracted using the sampling rate. We consider an *input profile* $\mathcal{V}$ as some compact set $\mathcal{V} \subseteq \mathbb{R}^n$ such that for any $v \in \mathcal{V}$, $v[i]$ is a possible value for $x[i](0)$, i.e. $x[i]$ at $t = 0$, $1 \leq i \leq n$. An input profile is essentially a collection of possible initial states.

The quality of control $Q(\mathcal{T})$ (QoC) of a control software implementation $\mathcal{T}$ is considered as the probability with which the system satisfies a given control performance requirement for a given intermittent fault model $\mathcal{I}$ and an input profile $\mathcal{V}$.

V. Formal Estimation of QoC

We now present a formal approach towards estimating the QoC of software implementations of control laws in the lines of [11]. We first generate all possible L length *error scenarios* from the intermittent fault profile (L, M, N). Here M and N are the *error* and *error recovery* periods respectively and the performance guarantee is defined over L control loop iterations. To this end we generate all L length binary encodings where 0 denotes fault free control law computation and 1 denotes that an intermittent fault has occurred thus resulting in erroneous control law computation. The strings are generated with the following restrictions.

1) Every substring comprising only 1's will be of maximum M length, the maximum error period.
2) Every substring comprising only 1's will be followed by at least N 0's which is the minimum length of error recovery period.

Let E be the set of all possible intermittent fault scenarios of L length encoded as binary strings and generated using the rules as defined above. We use the function GenEnc to generate the set E of all possible error encodings for the intermittent

fault profile $\mathcal{I} = (L, M, N)$. Using P_{error} computed in (1), the probability Pr_e of a fault scenario $e \in E$ is computed as

$$Pr_e = \prod_{i=1}^{L} \{e[i] * P_{error} + (1 - e[i]) * (1 - P_{error})\}$$

For each intermittent fault scenario $e \in E$, we generate an error annotated program $\mathcal{S}_e$ (see Algorithm 1) that captures L consecutive closed control loop iterations which are effected by the specific fault scenario e via the following steps.

Algorithm 1 Error Annotated Program $\mathcal{S}_e$

Require: Error Scenario e,Plant Description $\mathcal{T}$, Control Law $\mathcal{P}$, Intermittent Fault Profile $\mathcal{I}$

1: $m := 0; n := 0;$ ▷ Initialization
2: **for** each $itr \in [1, L]$ **do** ▷ Compute Control Law
3: **if** $(e[itr] = 0)$ **then** ▷ No error
4: **if** $(m \neq 0)$ **then**
5: $n++;$
6: **switch** (m) **do**
7: **case** $m <= M$
8: **switch** (n) **do**
9: **case** $n < N$
10: $\mathcal{T}$:$u(itr) := K_n^{(m)} X(itr - 1);$
11: **default**
12: $\mathcal{T}$:$u(itr) := K_{lqr} X(itr - 1);$
13: $m := 0; n := 0;$
14: **default**
15: $\mathcal{T}$:$u(itr) := K_{lqr} X(itr - 1);$
16: **else** ▷ Error
17: T:$u(itr) := u(itr - 1);$
18: $m++;$
19: $\mathcal{P} : X(itr) := AX(itr - 1) + Bu(itr);$ ▷ Update Plant

Step 1 The variables m and n are used to represent the last encountered *error period* length and the number of recovery gains applied respectively. Both are initialized to 0 for every encoding e (Line 2).

Step 2 For any closed loop iteration whose index is denoted by itr, if $e[itr] == 1$ then we know that an intermittent fault has occurred and we use the previously computed control law for the last iteration (Line 18). We also increase m to keep track of the length of the error period encountered (line 19).

Step 3 If $e[itr] == 0$ then we know no intermittent fault has occurred in the closed loop iteration. At this point, if the value of m is non-zero, we need to start the error recovery period and we increase n by 1 (Lines 4-6).

Step 4 Now depending on the length of the last error period m encountered, the recovery gains are chosen. This is denoted by a switch statement over m and for the sake of brevity, we abuse the *case* notation and denote the M possible case statements over the value of m as "**case** $m \leq M$" (Line 8).

Step 5 We now compute the control law depending on n which keeps track of the recovery gain to apply. Here also, we use a switch statement over n, combine all the cases under "**case** $n < N$" and select the required gain as $K_n^{(m)}$ (Lines 9-11).

Step 6 After applying $(N - 1)$ recovery actions, the value of n becomes N and we apply the original gain K_{lqr} and reset m and n to 0 to mark the end of error recovery period (Lines 12-14). We now present the formal approach towards QoC estimation in Algorithm 2.

Algorithm 2 QoC Estimation

Require: Error Scenario e,Plant Description $\mathcal{T}$, Control Law $\mathcal{P}$, Intermittent Fault Profile $\mathcal{I}$, Input Profile $\mathcal{V}$, Performance Requirement $\mathcal{R}$

1: $E \leftarrow$ GENENC($\mathcal{I}$); ▷ Generate All Error Scenarios
2: MCA(S,L,$\mathcal{M}_f$); ▷ Generate Probability of All Error Scenarios
3: $\mathcal{Q} := 0;$ ▷ Initialization
4: **for** each encoding $e \in E$ **do**
5: Compute S_e; ▷ Error Annotated Program
6: $A \leftarrow$ **assert**($\mathcal{R}$); ▷ Performance Assertion
7: $wp \leftarrow WP(\mathcal{S}_e, A);$ ▷ Generating Weakest Precondition
8: $\mathcal{Q}(\mathcal{S}_e) := (Vol(wp \wedge \mathcal{V})/Vol(\mathcal{V})) * Pr(e);$ ▷ QoC for S_e
9: $\mathcal{Q} := \mathcal{Q} + \mathcal{Q}(\mathcal{S}_e);$ ▷ Overall QoC

In Algorithm 2 we first generate the set E of all possible error scenario encodings using the function GENENC (Line 1). For each error scenario $e \in E$, we compute its corresponding error annotated program S_e using Algorithm 1 (Line 5). The closed range of values of the input profile $\mathcal{V}$ induce a convex polytope. The polytope $\mathcal{V}$ captures *all* possible initial values of the plant state vector $x(0)$. Let A denote the assertion guaranteeing the performance requirement of the system (Line 6). We compute the weakest pre-condition (WP) [9] of the error annotated program $\mathcal{S}_e$ subject to the performance assertion A as the post condition given by $wp = WP(\mathcal{S}_e, A)$ (Line 7). This induces a convex polytope $wp \wedge \mathcal{V} \subseteq \mathcal{V}$ which captures the segment of initial values for the state variables for which the assertion A is guaranteed to hold under the intermittent fault scenario e. Let the volume of a convex polytope be computed by a unary operation Vol. We calculate the volume of the polytope $wp \wedge \mathcal{V}$ and divide it by the volume of the polytope induced by the input profile $\mathcal{V}$. This ratio multiplied by the probability of the error scenario Pr_e serves as the QoC of the system under the error scenario e (Line 8). The overall QoC $\mathcal{Q}$ of the system is the sum of the QoC $\mathcal{S}_e$ of all possible error scenario $e \in E$ (Line 9).

We employ the static source code analyzer Frama-C WP plug-in [12] in order to generate weakest preconditions. The discrete domain model counting tool 'LattE' [10] implements Vol operations on the generated convex polytopes considering uniform distribution of variables within a specified region. The analysis is restricted to linear control systems as LattE is restricted to linear constraints. We employ *Volume Computation* instead of *Model Checking* tools because we are not analyzing a safety property like stability but a performance property. The volume of the state space satisfying the performance assertions formally relates with the QoC metric.

VI. CASE STUDY

We now illustrate the design of our proposed fault-tolerant controller using an automotive cruise control system as a case study. We use a discrete-time version of the linearized continuous-time model of this cruise control system [13] with sampling period $h = 10ms$, given by

$$x(t+1) = \begin{bmatrix} 1.00 & 0.01 & 0.00 \\ -0.0003 & 0.9997 & 0.01 \\ -0.0604 & -0.0531 & 0.9974 \end{bmatrix} x(t) + \begin{bmatrix} 0.0001 \\ 0.0001 \\ 0.0247 \end{bmatrix} u.$$

The system receives the reference or the commanded vehicle's speed from the driver and regulates the speed following the driver's command. Based on the reference speed and the feedback signals, the cruise control system regulates the engine's speed by adjusting the engine throttle angle (i.e., fuel rate) to increase or decrease the engine drive force. The state $x_1(t)$ (output state) indicates the vehicle speed and $u(t)$ (control input) is the engine throttle angle. We consider the system in braking mode and the objective is to choose $u(t)$ such that $x_1(t) = 0$, i.e., to stop the vehicle. The input profile of the system is $x_1 \in [0, 100], x_2 = 50, x_3 = 50$. Moreover, we must satisfy a performance requirement that the speed $x_1(t) \leq 4$ units in $2sec$ after the brake is pressed.

When the system is running without error, we design an optimal controller gain K_{lqr} using LQR [8],

$$u(k) = K_{lqr} \cdot x(k), \tag{21}$$

which minimizes the cost function,

$$C_{lqr} = x^T(k)Qx(k) + u^T(k)Ru(k), \tag{22}$$

where $Q = Q^T > 0$ and $R = R^T > 0$. With (21) and,

$$Q = \begin{bmatrix} 10^6 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, R = 1, \tag{23}$$

the LQR gain is found to be

$$K_{lqr} = \begin{bmatrix} -872.5367 & -131.4940 & -10.0972 \end{bmatrix} \tag{24}$$

Now, consider an intermittent fault profile (20,3,2), i.e., a maximum of three consecutive erroneous executions can happen and every error period is followed by a minimum of two error-free executions over the control period of length 20.

With $M = 3$, there are three categories of error periods. For each error period, the following controller gains are designed according to Theorem IV.3 and (20)

$$\begin{aligned} K_1^{(1)} &= \begin{bmatrix} -2092.6 & -190.6 & -39.0 \end{bmatrix}, \\ K_1^{(2)} &= \begin{bmatrix} -1457.9 & -113.8 & -30.6 \end{bmatrix}, \\ K_1^{(3)} &= \begin{bmatrix} 1822.1 & 350.6 & 3.4 \end{bmatrix}. \end{aligned} \tag{25}$$

The error profile $(20, 3, 2)$ covers 76.98% of all the possible intermittent fault scenarios over length $L = 20$ for $P_{mc} = 0.7$. The stability of the system for the error profile $(20, 3, 2)$ is guaranteed by the design of the fault-tolerant controllers. Apart from the stability, let the required performance guarantee be that the speed $x_1(t) \leq 4$ within $2sec$. The sampling period is $10ms$. Hence the speed $x_1(t)$ should reduce from a maximum of 100 units to less than 4 units within 200 applications of control action i.e the plant vector $x_1(t)$ should reduce by a fraction of $4/100 = 0.04$ within 200 closed loop control executions. Hence, any attempt to estimate the quality of control would have to be defined over 200 closed loop control executions. As we have to generate all possible intermittent error scenarios over 200 closed loop iterations, we run into scalability issues as 2^{200} is not a *tangible* number to work with. In order to improve scalability, we derive a *stricter* performance guarantee over 20 closed loop control executions. In every 20 iterations, we desire that the speed $x_1(t)$ should become $\sqrt[200/20]{0.04} = \sqrt[10]{0.04} = 0.724$ times the original value in order to meet the performance requirement $x_1(t) \leq 4$ within $2sec$. Note that the derived requirement is a stricter one in the sense that satisfaction of the derived requirement over 20 cycles will imply the satisfaction of the original requirement over 200 cycles but the opposite need not necessarily hold. In general, a performance requirement over L control loop iterations specified as $\wedge_i x_i(L) < c_i$ can be replaced with a stricter version over $l < L$ iterations as

$$\bigwedge_i \frac{x_i((j+1)\cdot l)}{x_i(j \cdot l)} < \sqrt[L/l]{\frac{C_i}{X_i}}, \forall j \in [0, \frac{L}{l} - 1]$$

where $x_i(0) \in [0, X_i]$. Note that if the performance criteria is made stricter, the probability computed is exact for this criteria and a lower bound for the actual performance criteria.

We formally verify the QoC of the fault-tolerant controller in (25) for the performance criteria $x_1(t) \leq 4$ within $2sec$ under the input profile $([0, 100], 50, 50)$. Our QoC estimation framework takes 55 min on a Xeon E5-2687WO with 192 GB memory to report that the fault-tolerant controller in (25) offers a QoC of 91.34% for the intermittent fault profile $(20, 3, 2)$.

VII. Conclusion

Traditionally, hardware faults are mitigated either by re-execution or hardware redundancy – both being expensive in terms of time or cost. Control applications are often robust against small number of faulty executions. The relevant research questions addressed in this work are: (i) platform aware characterization of intermittent faults, (ii) a controller design method which takes into account a given fault characterization while introducing performance-awareness in the control algorithms and (iii) a formal analysis flow that validates the performance of a software control implementation in presence of faults. These steps actually form a design validation cycle which is the key contribution of the present work.

References

[1] L. Rashid, K. Pattabiraman, and S. Gopalakrishnan, "Intermittent hardware errors recovery: Modeling and evaluation," in *QEST*, 2012.

[2] D. Goswami, D. Muller-Gritschneder, T. Basten, U. Schlichtmann, and S. Chakraborty, "Fault-tolerant embedded control systems for unreliable hardware," in *ISIC*, 2014.

[3] R. Majumdar, I. Saha, and M. Zamani, "Synthesis of minimal-error control software," in *EMSOFT*. ACM, 2012, pp. 123–132.

[4] L. Rashid, K. Pattabiraman, and S. Gopalakrishnan, "Characterizing the impact of intermittent hardware faults on programs," *IEEE Transactions on Reliability*, vol. 64, no. 1, pp. 297–310, 2015.

[5] Z. Sun and S. S. Ge, *Stability Theory of Switched Dynamical Systems*. Springer, 2011.

[6] O. Mason and R. Shorten, "On common quadratic Lyapunov functions for stable discretetime LTI systems," *IMA Journal of Applied Mathamatics*, vol. 69, no. 3, pp. 271–283, 2004.

[7] W. Rugh, *Linear Systems Theory*. Prentice-Hall, N.J., 1996.

[8] R. C. Dorf and R. H. Bishop, *Modern Control Systems*. Addison Wesley, 1995.

[9] E. W. Dijkstra, "Guarded commands, nondeterminacy and formal derivation of programs," *Communications of the ACM*, vol. 18, no. 8, pp. 453–457, 1975.

[10] J. A. De Loera, R. Hemmecke, J. Tauzer, and R. Yoshida, "Effective lattice point counting in rational convex polytopes," *Journal of symbolic computation*, vol. 38, no. 4, pp. 1273–1302, 2004.

[11] D. Lohar, A. Dunaboyina, D. Das, and S. Dey, "Failure estimation of behavioral specifications," in *SETTA*. Springer, 2016, pp. 315–322.

[12] P. Cuoq, F. Kirchner, N. Kosmatov, V. Prevosto, J. Signoles, and B. Yakobowski, "Frama-c," in *SEFM*. Springer, 2012, pp. 233–247.

[13] D. Goswami, R. Schneider, and S. Chakraborty, "Relaxing signal delay constraints in distributed embedded controllers," *IEEE Transaction on Control Systems Technology*, vol. 22, no. 6, pp. 2337–2345, 2014.