Reset-Based Recovery for Real-Time Cyber-Physical Systems with Temporal Safety Constraints

Fardin Abdi Taghi Abad*, Renato Mancuso*, Stanley Bak†, Or Dantsker‡ and Marco Caccamo*
*Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, USA
†United States Air Force Research Lab, Rome, USA
‡Department of Aerospace Engineering, University of Illinois at Urbana-Champaign, Urbana, USA

***Abstract*—In traditional computing systems, software problems are often resolved by platform restarts. This approach, however, cannot be naïvely used in cyber-physical systems (CPS). In fact, in this class of systems, ensuring safety strictly depends on the ability to respect hard real-time constraints. Several adaptations of the Simplex architecture have been proposed to guarantee safety in spite of misbehaving software components. However, the problem of performing recovery into a fully operational state has not been extensively addressed.**

In this work, we discuss how resets can be used in CPS as an effective strategy to recover from a variety of software faults. Our work extends the Simplex architecture in a number of directions. First, we provide sufficient conditions under which safety is guaranteed in spite of fault-induced resets. Second, we introduce a novel technique to express not only state-dependent safety constraints, as typically done in Simplex, but also time-dependent safety properties. Finally, through a proof-of-concept minimal implementation on a small R/C helicopter and simulation-based system modeling, we show the effectiveness of the proposed recovery strategy under the assumed fault model.

1. Introduction

There are an increasing number of CPS applications in almost all the vital infrastructures of our modern society. Such systems often have a set of safety requirements that need to remain satisfied at all times because a violation could have catastrophic consequences. However, software components can exhibit unexpected deviations from the intended behavior due to bugs, potentially violating the safety requirements. Unfortunately, formally assessing the correctness of software components is a hard problem since the existing approaches currently require a large amount of effort (cost) as well as specialized knowledge which is not yet widespread.

The difficulty of producing 100% correct software is a strong incentive to develop techniques to enforce safety requirements in CPS in spite of unexpected misbehavior. However, safety is not the only goal of a CPS, also the capability to remain in a fully operational state is of paramount importance. Techniques designed to maintain safety have received substantial attention in the literature [1]–[7]. In comparison, the problem of restoring nominal operation for a CPS has received little attention. In this work, we improve the state of the art of safety enforcement for CPS and discuss the use of resets as a strategy to fully recover from transient faults.

The Application-level Simplex architecture, proposed in [1]–[4], represents a well-known method to provide safety guarantees for CPS. In the Simplex architecture, a verified, *simple* safety controller ensures the stability of the plant. This conservative safety controller is complemented by a high-performance *complex* controller. A decision module continuously evaluates the safety properties and forwards actuation commands from the complex controller whenever the system operates within the safety margins. If a misbehavior is detected from the state of the plant, control is transferred to the safety controller. This prevents the occurrence of faults within the complex controller from compromising the safety of the plant. The main issue with the Application-level Simplex is that safety and complex controller are implemented as two applications on the same platform. Hence, in presence of platform-level faults, there is no guarantee of correct behavior from the safety controller. The issue is addressed in the System-level Simplex architecture [7] by moving the safety controller and the decision module into a dedicated processing unit. The safety controller in both the Application- and System-level Simplex has safety boundaries that are typically pessimistic and statically computed at design time. The work in [5] demonstrated that real-time reachability analysis can be employed to relax such static constraints. In fact, a plant can be allowed to abandon its safety boundaries as long as (i) no constraints are violated, and (ii) the state can be guaranteed to re-enter the safety region.

In this paper, we build upon the work in [5] and improve over System-level Simplex [7] in two main directions. First, to the best of our knowledge, we are the first to extensively discuss how platform-wise resets can be employed in CPS as a way to (i) perform fault recovery and to (ii) restore a full operational status. Second, we extend real-time reachability to check safety properties that depend not only on the current state of the system as originally proposed in [5], but also on its history.

In order to evaluate the validity and feasibility of the proposed strategy, we conduct a case study using a radio-controlled helicopter testbed. For our study, we use sensor traces acquired in flight while manually injected faults trigger platform-level recovery through resets. The acquired data is used to tune and validate our helicopter model. Next, we perform simulation-based analysis of the complete system based on the validated model. Our results show that: (i) restarting represents a feasible fault recovery approach; (ii) it is possible to formulate system constraints so that static and time-dependent safety constraints are respected despite the occurrence of resets; and (iii) if the frequency of faults is not too high, the proposed recovery methodology has a negligible impact on system's performance.

This paper is organized as follows. A brief review of the related works is presented in Section 2. In Section 3, we formalize the two categories of safety guarantees that our design can provide. In Section 4, a background on the

Simplex architecture is presented. Section 5, provides the overall design. The evaluation on the helicopter system is provided in Section 6. Section 7 concludes the paper.

2. Related Work

Restart based strategies are generally divided into two categories, *revival*, which is to reactively restart a failed component, and *rejuvination*, which is to prophylactically restart functioning components to prevent state degradation. [8] introduces recursively restartable systems as a design paradigm for highly available systems and uses a combination of revival and rejuvenation techniques. Authors in [9]–[11] propose the concept of microreboot which consists of having fine-grain rebootable components and trying to restart them from the smallest component to the biggest one in the presence of faults. Some works have focused on failure and fault modelling [12]–[14] and try to find the optimal rejuvenation strategy. Authors in [15], [16] propose an auditable restoration for distributed systems. These techniques are proposed for traditional computing systems and are not applicable to CPS. In [17], authors propose improving reliability of real-time control systems by executing simultaneous task replica of varying complexity. When a task fails, one of the shadow tasks can provide the output while the failed task is restarted to a clean state. However, this system has no mechanism to guarantee any system constraints. To our knowledge, this is the first work to extensively consider enabling systematic restart-based recovery with safety guarantees for CPS.

3. System Constraints

The *safety requirements* of a system are conditions that need to remain satisfied at all times during system operation. In this work we consider two categories of constraints: *Hard* Constraints and *Overrun* Constraints.

Hard Constraints are expressed in the form of hard, physical constraints over the system's state space. When considered together, they determine the feasible regions of the state space where the system can operate. Each hard constraint is presented as a linear inequality of the form $a_m^T \cdot x \leq 1$, where $x \in \mathbb{R}^n$ is the vector of state variables of the system and $a_m \in \mathbb{R}^n$ is a vector of constants. For instance, for a helicopter, a hard constraint is imposed on the altitude to prevent a crash.

On the other hand, **Overrun Constraints** are defined on the trajectory of the system over time. An overrun constraint has the following form:

$$\forall t; \int_t^{t+T^{win}} \text{Stress}(x(\tau)) \cdot d\tau \leq C \tag{1}$$

Here "Stress" is a *non-negative* function that defines the amount of instantaneous stress on the system for a given state, x. C is the maximum amount of accumulated stress that is allowed over any time window of length T^{win}. Hence, each overrun constraint with index k is specified by the tuple $\langle \text{Stress}_k(x), C_k, T_k^{win} \rangle$.

For example, the propulsion system is limited by its ability to dissipate heat. Consequently, motor datasheets specify the maximum time the motor can be operated at full power. For instance, the maximum duration allowed for "Hacker" brushless motors to operate at full power is 15 seconds [18], [19]. this can be implied as

$$\text{Stress}(p) = \begin{cases} 1 & p > p_h \\ 0 & p \leq p_h \end{cases}, \forall t; \int_t^{t+16} \text{Stress}(p(\tau)) \cdot d\tau \leq 15$$

where p represents the instantaneous propulsion power and p_h the threshold for full power level.

The combination of all the constraints, is referred to as **System Constraints**. The goal of design verification techniques for CPS is to ensure that all the system constraints are met throughout operation.

4. Background on Simplex Architecture

The goal of using Simplex architecture, originally proposed in [1]–[4], is to enable a system designer to use an unverified controller on the system while ensuring the same safety guarantees that a verified safety controller would offer. The safety controller is designed by approximating the system with linear dynamics in the form: $\dot{x} = Ax + Bu$, for state vector x and input vector u. In this approach, *safety constraints* are expressed as linear constraints in an LMI form. These constraints, along with the linear dynamics for the system, are the inputs to a convex optimization problem that produces both linear proportional controller gains K, as well as a positive-definite matrix P. The resulting linear-state feedback controller, $u = Kx$, yields closed-loop dynamics in the form of $\dot{x} = (A + BK)x$. Given a state x, when the input Kx is used, the P matrix defines a Lyapunov potential function ($x^T Px$) with a negative-definite derivative. As a result, the stability of the linear system is guaranteed using Lyapunov's direct or indirect methods. Furthermore, the matrix P defines an ellipsoid in the state space where all constraints are satisfied when $x^T Px < 1$. If sensors' and actuators' saturation points were provided as constraints, the states inside the ellipsoid can be reached using control commands within the sensor/actuator limits.

It follows that the ellipsoid of states, $\mathcal{R} = \{x | x^T Px < 1\}$, is a subset of the recoverable states. As long as the system's state remains inside of the ellipsoid, the system will be driven toward the equilibrium point, i.e. where $x^T Px = 0$, when control is handed over to the safety controller. Since the potential function is strictly decreasing over time, any trajectory starting inside $\mathcal{R}$ will remain there for an unbounded time window. Therefore no unsafe states will ever be reached as there are no such states in $\mathcal{R}$.

5. Methodology

In this section we describe our design methodology. The core of this design is the Simplex Architecture which was reviewed in section 4. Our first goal is to extend Simplex to provide runtime guarantees for *hard* constraints and the additional category of *overrun* constraints in spite of faults in complex controller. The second goal is to allow recovery from faults through platform-level restarts.

Our design is comprised of a verified, simple Safety Controller (SC), an unverified Complex Controller (CC), and a real-time reachability module (RTR). The architecture is depicted in Figure 1. Thanks to the properties discussed in Section 4 and the design methodology in section 5.1, SC

should always be able to stabilize the system. SC, therefore, is set as the default controller of the physical plant. The real-time reachability module (RTR), on the other side, periodically checks whether the safety requirements of the system remain satisfied under all possible control commands of the CC. If all these conditions hold, CC would safely be left in charge of control for the next cycle. This approach prevents logical bugs in the CC from violating any of the system safety constraints.

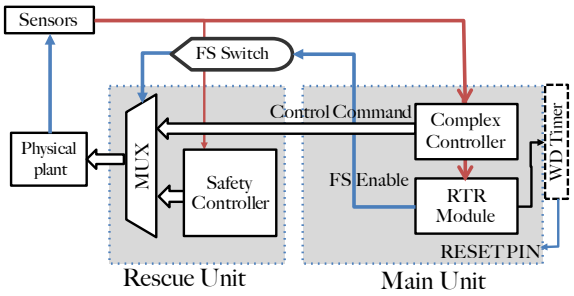


Figure 1: Restartable Simplex architecture with dedicated safety controller hardware unit.

Furthermore, in this design, SC is implemented on a dedicated hardware unit and control is automatically transferred to the SC while the main computation unit undergoes a restart. This prevents the SC from being affected by faults on the elements of the main unit. The mechanism to reliably switch between controllers is enabled by a *fail-safe switch* (FS). The input to the FS is the `FSEnable` signal, which can assume three values: *active low*, *active high*, or *invalid*. The FS selects control commands from the CC if it receives an active high `FSEnable` signal. Conversely, if the input is active low or invalid, which is the case during the restart of the main unit, the FS selects the SC as the control unit. This approach constitutes a reliable way to immediately put the safety controller in charge when the main unit is undergoing a restart. Under this design, the system remains stable no matter what the frequency or the length of resets is, because the SC is able to maintain stability for arbitrarily long time intervals.

In this paper, the assumed fault model for the RTR modules is **fail-stop** and for the CC is **Byzantine**. We rely on hardware watchdog timers to ensure that the main unit will restart after any fail-stop failure. Periodic, controlled resets or any strategy based on misbehavior detection performed by the RTR module can be employed to recover main unit from Byzantine failures.

This architecture is similar to what was proposed in our previous work, System-level Simplex [7]. System-level Simplex, however, exploits this hardware redundancy to protect the system against the faults that may occur in the underlying layers of system such as the OS.

In the rest of this section, we explain in detail how each component should be implemented and how the proposed architecture meets our design goals.

5.1. Safety Controller (SC) Design

SC is a simple verified controller that is responsible for ensuring that hard and overrun constraints are satisfied. In our approach, (i) we first find a region of states such that, for any trajectory taken by the system inside this region, all the constraints remain satisfied. Next, (ii) we design the SC such that it can keep the state inside this region for an unbounded amount of time as long as the starting state is inside the region. In order to do this, we first express all the constraints as linear inequalities. Then we use a LMI solver to derive a feedback controller with the discussed properties.

Hard constraints, as described in Section 3, are already in the form of linear inequalities. Hence, for a system with q hard constraints, we can easily define a region $\mathcal{S}$ such that all the hard constraints are satisfied:

$$\mathcal{S} = \{x | a_m^T \cdot x \leq 1, m = 1, \ldots, q\}. \quad (2)$$

For an overrun constraint, the integral constraint is converted to an instantaneous constraint by defining region O such that:

$$O = \{x | \text{Stress}(x) \leq (1-\alpha)C/T^{win}\}. \quad (3)$$

The integration of Stress(x) over any trajectory of length T^{win} inside O would remain less than the maximum permitted accumulated stress, C. Here, α is the **Manoeuvrability coefficient** and is $0 \leq \alpha \leq 1$. The choice and impact of α will be further discussed in the context of Lemma 1. It can be easily shown that for any trajectory of length T^{win} inside the region O, the overrun constraint holds:

$$\forall t; \int_t^{t+T^{win}} \text{Stress}(x(\tau)) \cdot d\tau \leq \frac{(1-\alpha)C}{T^{win}} \times T^{win} \leq C \quad (4)$$

However, in order to use the region O in a LMI-solver, we choose p linear inequalities to determine a convex subset, $Conv(O) \subseteq O$ such that:

$$Conv(O) = \{x | c_i^T \cdot x \leq 1, i = 1, ..., p\} \subseteq O$$

Since linear matrix inequality can only handle linear systems, we also need to restrict the system with actuator saturation limits (so that actuation values do not saturate the actuators). Therefore, assuming that $u \in \mathbb{R}^m$ is the control signal to the actuators, saturation limits can be expressed as:

$$b_j^T \cdot u \leq 1, j = 1, \ldots, r$$

For a system whose dynamics are described by $\dot{x} = Ax + Bu$, the SC is a linear state feedback control given by $u = Kx$. The feasible region Γ of the system with q hard constraints, p overrun constraints (each overrun constraint itself has p_i linear inequalities) and r actuator constraints can be described by:

$$\begin{aligned} \Gamma = \{x | a_m^T \cdot x \leq 1, m = 1, \ldots, q, \\ c_{i,k}^T \cdot x \leq 1, k = 1, \ldots, p_i, i = 1, \ldots, p, \\ b_j^T \cdot u \leq 1, j = 1, \ldots, r\} \end{aligned} \quad (5)$$

Note that $\Gamma \subseteq \mathcal{S}$, $\Gamma \subseteq O$, and Γ embeds saturation limits. Now, we can use a LMI solver[1] to find Γ, the gain matrix K of SC (SC is a state feedback controller), and the matrix Q. The latter matrix Q is found such that the Lyapunov potential function $V(x) = x^T Q^{-1} x$ constructed

1. This is a standard minimization problem and can be solved using the approach proposed in the second appendix of the technical report in [20].

for the system under the feedback control of K (i.e. $\dot{x} = (A + BK)x$) has negative-definite derivatives in the region $\mathcal{R} = \{x | x^T Q^{-1} x < 1\} \subseteq \Gamma$.

We refer to $\mathcal{R}$ as the *Stability region*. Due to the negative-definite derivatives of $V(x)$ inside $\mathcal{R}$, any trajectory starting in $\mathcal{R}$ will remain in $\mathcal{R}$ indefinitely.

5.2. Real-Time Reachability Module (RTR)

We know that the SC can stabilize the system and guarantee all the constraints as long as the state of the system is inside $\mathcal{R}$. The goal of RTR module is to allow the system to operate beyond the boundaries of region $\mathcal{R}$.

In this section, we derive a set of conditions that if satisfied at the beginning of a cycle, hard and overrun constraints are guaranteed to remain satisfied throughout that cycle under any behavior of the CC. At every cycle T_c, RTR checks if those switching conditions hold (Theorem 1 and 2). If they do, the CC is allowed to control the system. Otherwise, SC will be in charge.

Next, we describe the modifications of the implementation of the RTR module, with respect to [5], that are required to check the switching conditions for the additional category of overrun constraints.

In the rest of this section, we use $\text{Reach}_{=T}(x, C)$ to denote the set of states reached by system from an initial set of states x after exactly T seconds have elapsed under the control of controller C. $\text{Reach}_{\leq T}(x, C)$ can be defined as $\bigcup_{t=0}^{T} \text{Reach}_{=t}(x, C)$. In order to compute the reach set, we use a modified version of the the face-lifting technique in [5]. We described our technique in Section 5.2.3.

5.2.1. Switching Conditions for Hard Constraints.

Consider T_c the control interval and T_s an arbitrary settling time for the system after a restart.

Theorem 1. *The hard constraints of the system will always remain satisfied under the control of CC, if at every control interval, T_c, the following conditions hold:*

1) $Reach_{\leq T_c}(x, CC) \subseteq \mathcal{S}$;
2) $Reach_{\leq T_s}(Reach_{\leq T_c}(x, CC), SC) \subseteq \mathcal{S}$;
3) $Reach_{=T_s}(Reach_{\leq T_c}(x, CC), SC) \subseteq \mathcal{R}$.

Proof. Due to space limitations, we hereby provide an intuition of the proof. For the full proof, refer to the technical report [21]. Condition 1 implies that all the states that can be reached under the CC within the next control cycle satisfy the hard constraints. If a switch to SC is triggered at any moment within the next control cycle, Condition 2 ensures that from the time of switching, for an interval of length T_s, the system will not violate any of the hard constraints. Finally, Condition 3, implies that by the end of T_s, the system is inside the stability region where the hard constraints will remain satisfied indefinitely. □

5.2.2. Switching Conditions for Overrun Constraints.

We assume that switching conditions for overrun constraints are checked only if the switching conditions for hard constraints are already satisfied. Therefore, Condition 3 in Theorem 1, implies that if a switch to SC occurs within the upcoming T_c time units, the SC will be able to safely bring the system back into the stability region within at most T_s time units. Hence, all the trajectories that satisfy the hard constraints have a form such that there is a time point t_s at which the trajectory enters the stability region $\mathcal{R}$ while the SC controller is in charge. For such trajectories, Lemma 1 implies a general condition under which a given overrun constraint remains satisfied throughout the execution.

Lemma 1. *Assume an arbitrary trajectory that at some point in time, t_s, enters the stability region $\mathcal{R}$ while the SC is the active controller from that point forward. An overrun constraint is satisfied throughout such a trajectory if the following condition holds:*

$$\forall t \in [0, t_s - T^{win}] : \int_{t}^{t+T^{win}} \text{Stress}(x(\tau)) \cdot d\tau \leq \alpha C \quad (6)$$

Proof. Here, α is the same maneuverability constant[2] used in design time of SC. Due to the lack of space, the proof is provided in the technical report [21]. □

Lemma 1, implies a general condition for a trajectory to satisfy overrun conditions. However, switching conditions need to be time-discrete in order to be checked by the RTR module in every cycle. Theorem 2 provides a way to derive discretized safe switching conditions based on Lemma 1.

Theorem 2's key idea is to track the accumulated stress during the past $T^{win} - (T_c + T_s)$ time window. Next, we compute the maximum of the sum for the stress that could be accumulated over the future interval of length $T_c + T_s$. This represents the worst-case accumulated stress from the current time until SC can bring back the system inside the stability region. Intuitively, if in total the worst-case future stress and the accumulated (past) stress are below the limit of the overrun constraint, the condition is satisfied. Otherwise, RTR will need to trigger a switch to SC.

Theorem 2. *Assuming that $T^{win} > T_s + T_c$, an overrun constraint will always remain satisfied under the control of CC, if at every control interval j the following condition holds:*

$$\sum_{p=j-(R+(M-1)\cdot L)}^{j} (\text{MaxSumStress}_x([p \cdot T_c, (p+1) \cdot T_c])) + \text{MaxSumStress}_x([(j+1) \cdot T_c, T_s]) \leq \alpha C \quad (7)$$

Here, $\text{MaxSumStress}_x([t_1, t_2])$ is a function defined to over-approximate the sum of stress over the trajectory in the interval of $[t_1, t_2]$. It is defined as:

$$\int_{\tau=t_1}^{t_2} \text{Stress}(x(\tau)) \cdot d\tau \leq \text{MaxSumStress}_x([t_1, t_2])$$

Additionally, we define L, Q, and R as follows:

$$L = \left\lceil \frac{T^{win} - (T_s + T_c)}{(M-1)T_c} \right\rceil, Q = \left\lfloor \frac{j}{L} \right\rfloor, R = j - QL$$

Here M is the length of the array we use to keep track of past stress, in which an element stores the cumulative stress over L consecutive control cycles.

2. The *Maneuverability Coefficient* is a design parameters such that $0 \leq \alpha \leq 1$. The choice of a larger α can increase the stability region of the SC. At the same time, it makes the conditions of Theorem 2 harder to satisfy, resulting in more frequent switches from CC to SC. Thus, α needs to be chosen carefully to balance this trade-off.

Proof. Due to the lack of space the proof is provided in the technical report [21]. □

Figure 2 illustrates how Theorem 2 can be applied to the system. In this example, the goal is to guarantee that the accumulated stress over any window of size $14T_c$ will not exceed a fixed threshold. The future time required for the system to settle in the $\mathcal{R}$ is $T_c + T_s = 3T_c$. Hence, we need to compute the stress over the future $3T_c$ plus the stress over the past $11T_c$ time units. The sum of the values from PS[1] to PS[4] in memory provides the accumulated stress over the past 14 cycles (i.e. from $8T_c$ to $22T_c$), which is an over-approximation of what required (11 cycles).

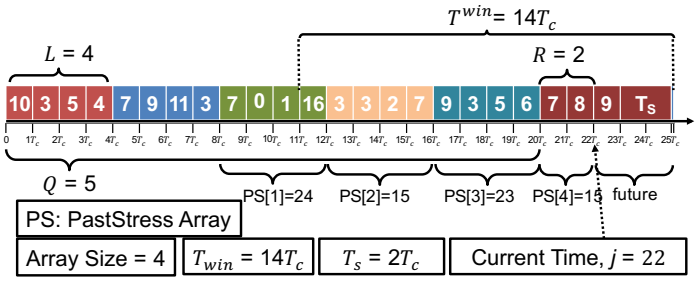


Figure 2: An example to clarify Theorem 2. Each element shows the maximum cumulative stress within that control cycle.

5.2.3. RTR Module Implementation. The structure of the main loop of the RTR module is presented in Algorithm 1. Only a single overrun constraint is considered. Each iteration of the while loop in Algorithm 1 performs the required computation for a single control cycle.

ALGORITHM 1: RTR module execution flow

```
1  Algorithm RTRModule()
2      set R = 0 and initialize all elements of array PastStress to
       MAXDOUBLE
3      while true do
4          state^{j-1} = readCurrentStateFromSensors /*At the
           beginning of j-1th control cycle*/
5          CCcommand = Control Command applied at j-1th cycle
6          state^j, MaxSumStress^{J-1} =
           Reach_{=T_c}(state^{j-1}, CCcommand)
7          PastStress[M] += MaxSumStress^{J-1}
8          SumOfPastStress = Sum all elements of PastStress
9          reachCC, MaxSumStress^J =
           Reach_{≤T_c}(state^j, CC, SumOfPastStress)
10         reachSCAtTs, - = Reach_{=T_s}(reachCC, SC)
11         reachSCBeforeTs, sumStressUntilSettling =
           Reach_{≤T_s}(reachCC, SC)
12         If R == L Then Shift PastStress to left; set R = 0 ; End
13         /*At the end of j-1th control cycle*/
14         If SumOfPastStress + MaxSumStress^J +
           sumStressUntilSettling < αC and reachCC ⊆ S and
           reachSCAtTs ⊆ R and reachSCBeforeTs ⊆ S then Put CC
           in charge; else switch to SC; End
15         R++ and Update WD timer.
16     end
```

Here j is the cycle for which the RTR module needs to determine whether the switching conditions hold or not. The decision of the RTR module needs to be ready at the beginning of the cycle j. Hence, those conditions need to be assessed during the previous cycle, $j-1$. Algorithm 1 uses the state at the beginning of $(j-1)^{th}$ cycle (line 4), as well as the exact control command generated at the beginning of $(j-1)^{th}$ cycle (line 5) to find the $\texttt{state}^j$ which is the reachable set of states at the beginning of j (line 6). `PastStress` is an array of size M to keep track of the stress during the past cycles. In each iteration, before evaluating the conditions for cycle j, the `MaxSumStress` is computed over the $(j-1)^{th}$ cycle and added to the last element of `PastStress` (line 7).

Next, the reachable sets required to check Conditions 1-3 in Theorem 1 for cycle j and also the accumulated stress during the j^{th} cycle and until the settling time are computed (lines 9,10, and 11). Finally, in line 14 the algorithm checks weather all the constraints are met. In case they are all met, control commands from CC are forwarded to the actuators, otherwise SC is selected to control the system.

The function `Reach`, used in Algorithm 1, implements a modification of the face-lifting real-time reachability algorithm originally proposed in [5] to compute the maximum cumulative stress (`MaxSumStress`) over a given time, in addition to the reachable set of states. Here, an intuition of how function `Reach` is implemented is provided. For the full details, however, refer to our technical report [21]. The representation used to track the set of states for real-time reachability is a single n-dimensional hyper-rectangle (box), where n is the number of state space variables. The way the set of states changes over time is only based on the derivatives *near the boundaries of the tracked set of states*. In this technique, a set of states is tracked at specific snapshots in time. Time is iteratively advanced (by some time step) from the initial states, which changes the set of states being tracked, until the desired final time is reached. In order to compute the accumulated stress, we require the system designer to provide a function, `StressMax`, for each overrun constraint that returns the maximum value of Stress function over a hyper-rectangular region (box) of states. In every intermediate step of `Reach` function, maximum value of Stress function over the intermediate reachable box is derived using `StressMax`. The upper-bound on accumulated stress is found by multiplying the maximum of Stress by the length of the intermediate time interval.

6. Evaluation

In this section, we present the proposed architecture's evaluation architecture. We demonstrate robustness against faults and timely recovery of a RC helicopter system through a combined testbed evaluation and simulation-based system modelling.

6.1. Model Development and SC Design

For the helicopter system, the goal is to design a SC that can maintain the following constraints. The safety constraint is to maintain a minimum altitude of 10 meters from the ground (i.e. not crash). The overrun constraint for this system is on the amount of time that the vertical velocity (velocity over z-axis) is higher than 3 m/s seconds. This velocity shall not be maintained for more than 15 seconds within any time window of length 60 seconds:

$$\text{Stress}(\dot{z}) = \begin{cases} 1 & \dot{z} > 3 \\ 0 & \dot{z} \leq 3 \end{cases}, \forall t; \int_t^{t+60} \text{Stress}(\dot{z}(\tau))d\tau \leq 15$$

6.1.1. Model Dynamics and Linearization. The model and the linearization method in this section is obtained from the aerospace literature and has been proposed

and utilized for helicopter controller design in [22]–[25]. Due to the space limitations, we omit the details of the model and linearization approach. For the full details refer to our technical report [21].

In this model, control authority for the helicopter is obtained via lift, generated by the main and tail rotors. The main rotor lift is decomposed into three components $(-\omega_2, \omega_1, u)$ and the tail rotor force into one component $(0, -\omega_3, 0)$, all in the body-fixed frame. In our notation, we use m, g, and c to indicate mass, gravitational constant and equation constants, respectively. The terms μ and $\hat{\tau}$ represent the target vertical rotor force and heading angle, while the commands produced for the main and tail rotor are denoted as u and τ respectively and are defined in Equation 10. The full helicopter dynamics in the generalized coordinates are derived from the Euler-Lagrange equations.

$$m\ddot{\xi} = R \begin{pmatrix} -\omega_2 \\ \omega_1 - \omega_3 \\ -u \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ mg \end{pmatrix} \tag{8}$$

$$\mathbb{I}\ddot{\eta} = -C(\eta, \dot{\eta})\dot{\eta} + \tau \tag{9}$$

here, $\xi = (x, y, z)$ are the coordinates in the 3D space, and $\eta = (\phi, \theta, \psi)$ represent the classical Euler angles, 'yaw', 'pitch', and 'roll'. $C(\eta, \dot{\eta})$ and R refer to the Coriolis matrix and the orientation matrix of the helicopter, respectively.

To satisfy the safety constraints, we set the SC's goal to stabilize the altitude ($\dot{z} = 0$) and to level the helicopter ($\phi = 0$, $\theta = 0$, $\psi = 0$). We set the linear domain such that the absolute value of the components of η are smaller than $\pi/3$. We use the following control transformation:

$$u = m\frac{\mu + g}{c\psi c_\theta}, \quad \tau = C(\eta, \dot{\eta})\dot{\eta} + \mathbb{I}\hat{\tau}. \tag{10}$$

The explicit form for the linearized system where small body forces (e.g. friction, air drag) are neglected is:

$$\begin{cases} \dot{z} = v_z, & \dot{v_z} = -\mu & \dot{\phi} = \gamma, & \dot{\gamma} = \hat{\tau}_\phi \\ \dot{\theta} = \omega, & \dot{\omega} = \hat{\tau}_\theta & \dot{\psi} = \lambda, & \dot{\lambda} = \hat{\tau}_\psi \end{cases} \tag{11}$$

The above equations can be represented as $\dot{x} = Ax + Bu$ where $x = [z, v_z, \phi, \gamma, \theta, \omega, \psi, \lambda]^T$ and $u = [\mu, \hat{\tau}_\phi, \hat{\tau}_\theta, \hat{\tau}_\psi]^T$. Now, we design the state feedback controller of $u = Kx$.

6.1.2. Safety and Overrun Constraints. The hard constraint on the minimum height can be expressed as a linear inequality as: $a_1 = [1/10, 0, 0, 0, 0, 0, 0, 0]^T$. For the overrun constraint, we consider a maneuverability coefficient $\alpha = 1$. Hence, Equation 3 can be written as

$$O = \{x | \text{Stress}(x) \leq 0\} \Rightarrow \forall x \in O, v_z = \dot{z} \leq 3$$

It follows that linear inequality resulting from the above constraint is: $c_1 = [0, 1/3, 0, 0, 0, 0, 0, 0]^T$.

6.1.3. SC Design. The goal is to design a SC which is able to satisfy these constraints. First, the linear model used for the system is only valid in the region $|\phi|, |\theta|, |\psi| \leq \pi/3$. In order to ensure that the system remains in this region, we add the following constraints:

$$\begin{aligned} a_2 = -a_3 &= [0, 0, 3/\pi, 0, 0, 0, 0, 0]^T \\ a_4 = -a_5 &= [0, 0, 0, 0, 3/\pi, 0, 0, 0]^T \\ a_6 = -a_7 &= [0, 0, 0, 0, 0, 0, 3/\pi, 0]^T \end{aligned}$$

Denoting with u_{max} the saturation limit for the lift power of the main rotor, we have $\mu < c_\psi c_\theta u_{max} - 1$. For the region $|\psi|, |\theta| < \pi/3$ we can write $\mu \leq \cos(\pi/3)^2 u_{max} - 1 = u_{max}/4 - 1$. We define $\mu_{max} = u_{max}/4 - 1$. Thus we have

$$b_1 = -b_2 = [1/\mu_{max}, 0, 0, 0]^T$$

Finally, the stability region of the SC can be defined as:

$$\Gamma = \{x \mid a_i^T x < 1, i = 1, \ldots, 7, c_1^T x < 1, b_1^T u < 1, b_2^T u < 1\}$$

Given the linearized system of equations (see Equation 11) with the set of constraints Γ, we need to solve the minimization problem in Section 5.1 to obtain K and Q. Finally, the SC has dynamics $u = Kx$ and the stability region can be derived as $\mathcal{R} = \{S \mid x^T Q^{-1} x \leq 1\}$.

6.2. Restarting in Action

The goal of this experiment is to demonstrate that the proposed switching and recovery method is applicable to a *real* safety critical CPS. We used a minimal implementation of the design with SC on a dedicated processor, a remote operator as CC, and a manual signal as the source of restart. Our testbed consists of an Align T-Rex 450 radio-controlled helicopter equipped with a ArduPilot APM 2.6 board [26] as the main unit and a Bavarian Demon board [27] as the rescue unit[3]. A Futaba FSU-2 [28] is utilized as the failsafe switch (FS).

In this test, during normal operation, the restart module generates a valid PWM signal, which instructs the FS to forward CC-generated commands. When the restart command from the ground control is received, an interrupt is triggered on the main unit that initiates a reset. During the reset, the PWM value in input to the FS becomes invalid, triggering a switch to the rescue unit. After the restart is completed, the main unit outputs a valid PWM pulse again, and control is handed back to the CC. The trace recorded from the helicopter during the flight is depicted in Figure 3. The first and second graphs, depict the altitude and the level angles of the helicopter, respectively. The third graph shows the time frame during which the SC was active. Even though the time required to restart the APM was only 85 ms, we manually forced the system to stay longer in the booting mode for evaluation purposes. The video recorded for this experiment can be found at [29].

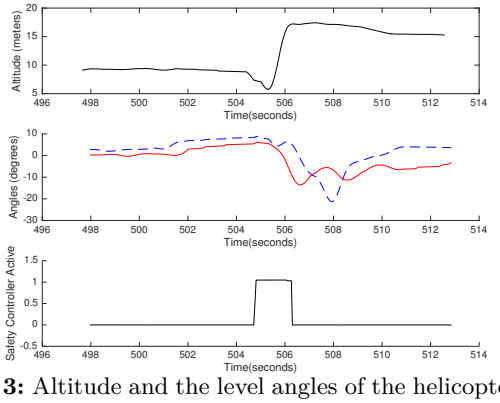


Figure 3: Altitude and the level angles of the helicopter during an in-flight restart

3. It is important to mention that the Bavarian Demon board is not a fully programmable board but it is essentially a tunable PID controller. This unit can easily be replaced with a PID controller implemented on a general purpose micro-controller.

6.3. Evaluation with the Simulated Model

For further analysis and testing of situations that are difficult to implement on the real system, we conducted our analysis using the validated model.

6.3.1. Progress Analysis. Restarting the platform impacts the system's progress towards the CC goal. The two parameters that determine the impact are (i) the frequency of restarts and (ii) time required to complete the restart. Figure 4, depicts the normalized comparison for various restart intervals and reboot lengths for a helicopter system where the CC is designed to keep the helicopter at a fixed altitude and forward velocity. As seen in Figure 4, cases with a small ratio of reboot length to restart interval have an almost negligible progress slowdown.

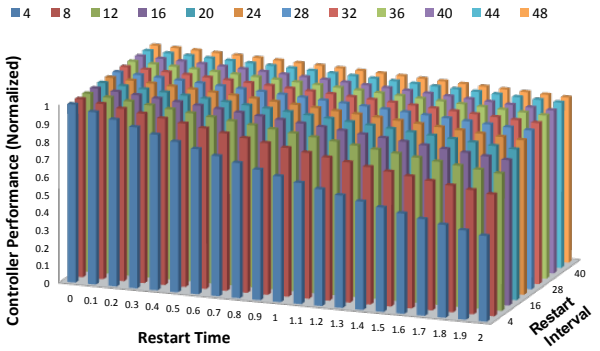


Figure 4: Impact of the restart interval and restart length on the control performance of system.

Next, we measured the time requirements for restarting three embedded platforms used in various applications: Freescale MPC564xL board (designed for automotive applications), Ardupilot APM 2.6 (commonly used in low-end UAVs) and the Intel Edison board (designed to target Internet-of-Things applications). The restart times measured for these platforms are presented in Table 1.

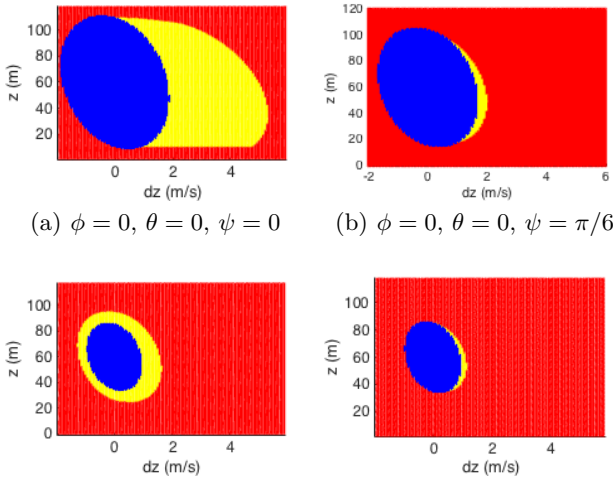
Platform Name	OS Type	Restart Time
Freescale MPC564xL	ERIKA RTOS	45 ms
Ardupilot APM 2.6	ArdOS	80 ms
Intel Edison	Yocto Linux	2031 ms

TABLE 1: Time required for full system restart

The conclusion that arises from the results in Figure 4 and Table 1, is that the impact of restarts on the progress of an embedded system with a typical fault rate can remain negligible. It follows that, if specific properties about the state of the system can be inferred after a reset, controlled periodic resets could also be introduced as a low-overhead strategy to "refresh" a live CPS and prevent the occurrence of unexpected faults.

6.3.2. Stabilizable Region Comparison. In this experiment, we compare the size of the operational region of the helicopter system under the original LMI-based Simplex [1]–[4], face-lifting real-time reachability [5] and our proposed modified RTR.

LMI-Simplex and RTR techniques cannot provide guarantees on the overrun constraints. Thus this experiment only considers hard constraints. The projection of the stabilizable region, for z and $\dot{z}$ is shown in Figure 5. In order to demonstrate an adverse case behavior, we have changed the projection plane such that we can observe the behavior near the boundaries of the stability region. In Figure 5(a), 5(b), 5(c) and 5(d) the level angles of helicopter were increased, which resulted in a reduction in the size of the stability region. As seen in the figures, when only the hard constraints of the system are considered, the obtained stability region via face-lifting real-time reachability and our modified RTR are identical. These figures, highlight the benefit of using real-time reachability and modified real-time reachability by the larger provably safe recoverable region (yellow).



(a) $\phi = 0$, $\theta = 0$, $\psi = 0$ (b) $\phi = 0$, $\theta = 0$, $\psi = \pi/6$

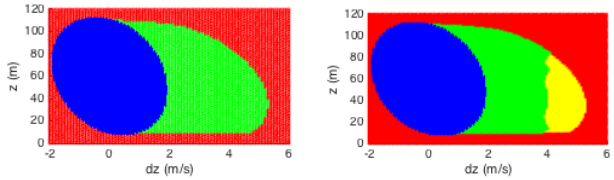
(c) $\phi = 0$, $\theta = \pi/6$, $\psi = \pi/6$ (d) $\phi = \pi/6$, $\theta = \pi/6$, $\psi = \pi/6$

Figure 5: Projection of stabilization region. Blue: LMI-Simplex; yellow: RTR and modified RTR; and red: unrecoverable. In all the figures we have $\dot{\phi} = 0$, $\dot{\theta} = 0$, $\dot{\psi} = 0$.

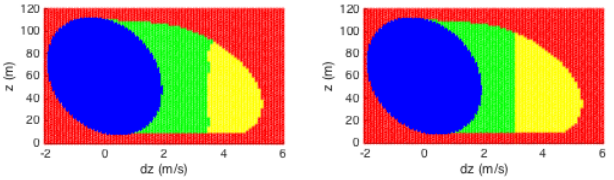
6.3.3. Modified RTR with Overrun Constraints. Next, we demonstrate that providing further guarantees on the overrun constraints can limit the operational region of system. The overrun constraint considered here was formulated in Section 6.1. Whether the overrun constraints is satisfied depends not only on the current state, but also on the trajectory followed by the system. Therefore, we project the stabilization region under increasing levels of accumulated stress over the past time window. Figures 6(a) to 6(d) depict the stability regions of the system from a given state for different values of accumulated stress. From left to right, top to bottom, the considered amount of accumulated stress is 12, 14, 14.5 and 15. It can be noted that in Figure 6(a) classic face-lifting RTR and our modified RTR produce an identical region because 3 seconds are sufficient for the SC to reduce $v_x = \dot{z}$ below the 3 m/s threshold. As the accumulated stress increases, the size of the green region decreases. Finally, in Figure 6(d), where the accumulated stress is already 15, the $\dot{z} = 3$ boundary cannot be crossed at any time.

7. Conclusions

In this paper, we enable continuously-actuated CPS using Simplex design to (i) to recover from the faults in a timely manner by restarting at runtime. Moreover, (ii) we propose a novel technique to guarantee a more complex category of safety constraints with a temporal aspect. And, (iii) through a proof-of-concept minimal implementation on a small unmanned helicopter and simulation-based system modeling, we show the effectiveness of proposed recovery



(a) Cumulative stress = 12 (b) Cumulative stress = 14



(c) Cumulative stress = 14.5 (d) Cumulative stress= 15

Figure 6: Blue: LMI-Simplex; yellow: RTR; green: modified RTR; and red: unrecoverable. Yellow: only hard constraints satisfied. Green: both overrun and hard constraints satisfied.

architecture under the assumed fault model. In the future we plan to investigate alternative real-time reachablity algorithms such as backward reachability analysis [30] which allows for near real-time verification of safety properties.

Acknowledgment

The material presented in this paper is based upon work supported by the Air Force office of Scientific Research (AFOSR) and the National Science Foundation (NSF) under grant numbers CNS-1302563, and CNS-1219064. Any opinions,findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the sponsors. DISTRIBUTION A. Approved for public release; Distribution unlimited. (Approval AFRL PA #88ABW-2016-1304, 23 JUN 2016)

References

[1] L. Sha, "Dependable system upgrade," in *Real-Time Systems Symposium, 1998. Proceedings., The 19th IEEE*. IEEE, 1998, pp. 440–448.

[2] L. Sha, "Using simplicity to control complexity." IEEE Software, 2001, pp. 20–28.

[3] L. Sha, R. Rajkumar, and M. Gagliardi, "Evolving dependable real-time systems," in *Aerospace Applications Conference, 1996. Proceedings., 1996 IEEE*, vol. 1. IEEE, 1996, pp. 335–346.

[4] T. L. Crenshaw, E. Gunter, C. L. Robinson, L. Sha, and P. Kumar, "The simplex reference model: Limiting fault-propagation due to unreliable components in cyber-physical system architectures," in *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*. IEEE, 2007, pp. 400–412.

[5] S. Bak, T. T. Johnson, M. Caccamo, and L. Sha, "Real-time reachability for verified simplex design," in *Real-Time Systems Symposium (RTSS), 2014 IEEE*. IEEE, 2014, pp. 138–148.

[6] S. Z. Bak, "Industrial application of the system-level simplex architecture for real-time embedded system safety," 2009.

[7] S. Bak, D. K. Chivukula, O. Adekunle, M. Sun, M. Caccamo, and L. Sha, "The system-level simplex architecture for improved real-time embedded system safety," in *Real-Time and Embedded Technology and Applications Symposium, 2009. RTAS 2009. 15th IEEE*. IEEE, 2009, pp. 99–107.

[8] G. Candea and A. Fox, "Recursive restartability: Turning the reboot sledgehammer into a scalpel," in *Hot Topics in Operating Systems, 2001. Proceedings of the Eighth Workshop on*. IEEE, 2001, pp. 125–130.

[9] G. Candea and A. Fox, "Crash-only software," 2003.

[10] G. Candea, E. Kiciman, S. Zhang, P. Keyani, and A. Fox, "Jagr: An autonomous self-recovering application server," in *Autonomic Computing Workshop. 2003. Proceedings of the*. IEEE, 2003, pp. 168–177.

[11] G. Candea, S. Kawamoto, Y. Fujiki, G. Friedman, and A. Fox, "Microreboot — a technique for cheap recovery," in *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, ser. OSDI'04, 2004, pp. 3–3.

[12] K. Vaidyanathan and K. S. Trivedi, "A comprehensive model for software rejuvenation," *Dependable and Secure Computing, IEEE Transactions on*, vol. 2, no. 2, pp. 124–137, 2005.

[13] S. Garg, A. Puliafito, M. Telek, and K. S. Trivedi, "Analysis of software rejuvenation using markov regenerative stochastic petri net," in *Software Reliability Engineering, 1995. Proceedings., Sixth International Symposium on*. IEEE, 1995, pp. 180–187.

[14] Y. Huang, C. Kintala, N. Kolettis, and N. D. Fulton, "Software rejuvenation: Analysis, module and applications," in *Fault-Tolerant Computing, 1995. FTCS-25. Digest of Papers., Twenty-Fifth International Symposium on*. IEEE, 1995, pp. 381–390.

[15] R. Hajisheykhi, M. Roohitavaf, and S. S. Kulkarni, "Auditable restoration of distributed programs," in *34th IEEE Symposium on Reliable Distributed Systems, SRDS 2015, Montreal, QC, Canada, September 28 - October 1, 2015*, 2015, pp. 37–46. [Online]. Available: http://dx.doi.org/10.1109/SRDS.2015.24

[16] R. Hajisheykhi, M. Roohitavaf, and S. S. Kulkarni, "Bounded auditable restoration of distributed programs," *To be appeared at IEEE Transaction on Computers.*, 2016.

[17] C. Zimmer and F. Mueller, "Fault resilient real-time design for noc architectures," in *Cyber-Physical Systems (ICCPS), 2012 IEEE/ACM Third International Conference on*. IEEE, 2012, pp. 75–84.

[18] E-Flite Inc., "Power 10 brushless outrunner motor datasheet," http://www.e-fliterc.com/ProdInfo/Files/EFLPower10OutrunnerInstructions.pdf, accessed: 2015-10-10.

[19] B. Venkataraman, B. Godsey, W. Premerlani, E. Shulman, M. Thaku, and R. Midence, "Fundamentals of a motor thermal model and its applications in motor protection," in *Protective Relay Engineers, 2005 58th Annual Conference for*. IEEE, 2005, pp. 127–144.

[20] D. Seto and L. Sha, "A case study on analytical analysis of the inverted pendulum real-time control system," DTIC Document, Tech. Rep., 1999.

[21] F. Abdi, S. Bak, R. Mancuso, O. D. Dantsker, and M. Caccamo, *Technical Report: Reset-Based Recovery for Real-Time Cyber-Physical Systems with Temporal Safety Constraints*, http://rtsl-edge.cs.illinois.edu/reset-based/techrep.pdf, Feb 2015.

[22] F. Mazenc, R. Mahony, and R. Lozano, "Forwarding control of scale model autonomous helicopter: a lyapunov control design," in *Decision and Control, 2003. Proceedings. 42nd IEEE Conference on*, vol. 4, Dec 2003, pp. 3960–3965 vol.4.

[23] O. Shakernia, Y. Ma, T. J. Koo, and S. Sastry, "Landing an unmanned air vehicle: Vision based motion estimation and nonlinear control," *Asian journal of control*, vol. 1, no. 3, pp. 128–145, 1999.

[24] E. Licéaga-Castrol, "A liouvillian systems approach for the trajectory planning-based control of helicopter models," *Int. J. Robust Nonlinear Contrul*, vol. 10, pp. 301–320, 2000.

[25] I. A. Raptis and K. P. Valavanis, *Linear and nonlinear control of small-scale unmanned helicopters*. Springer Science & Business Media, 2010, vol. 45.

[26] 3D Robotics, "Ardupilot apm2.6," http://3drobotics.com/kb/apm-2-6/, accessed: 2015-9-24.

[27] CAPTRON Electronic GmbH, "Bavarian demon datasheet, 3x/3xs series," http://www.bavariandemon.com/fileadmin/user_upload/downloads/bavarianDEMON-Instructions-3SX-3X_V6.1.pdf, accessed: 2015-9-24.

[28] Futaba Inc., "Fsu2 instruction manual," http://manuals.hobbico.com/fut/fsu2-manual.pdf, accessed: 2015-10-11.

[29] Fardin Abdi, "Flight demo video," https://youtu.be/tHcJUvBKd8Q.

[30] S. N. Ahmadyan and S. Vasudevan, "Reachability analysis of nonlinear analog circuits through iterative reachable set reduction," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, March 2013, pp. 1436–1441.