Runtime Verification with Controllable Time Predictability and Memory Utilization

(Technical Report CS-2013-02)

Ramy Medhat
Department of Electrical and Computer Engineering
University of Waterloo
ON, Canada, N2L 3G1
rmedhat@uwaterloo.ca

Deepak Kumar
Department of Electrical and Computer Engineering
University of Waterloo
ON, Canada, N2L 3G1
d6kumar@uwaterloo.ca

Borzoo Bonakdarpour
School of Computer Science
University of Waterloo
ON, Canada, N2L 3G1
borzoo@cs.uwaterloo.ca

Sebastian Fischmeister
Department of Electrical and Computer Engineering
University of Waterloo
ON, Canada, N2L 3G1
sfischme@uwaterloo.ca

ABSTRACT

The goal of runtime verification is to inspect the well-being of a system by employing a monitor during its execution. Such monitoring imposes costs in terms of resource utilization. Memory usage and predictability of the monitor invocations are among the indicators of the quality of a monitoring solution, especially in the context of embedded systems. In this paper, we propose a novel control-theoretic approach for coordinating time predictability and memory utilization in runtime monitoring of real-time embedded systems. In particular, we design a PID controller and four fuzzy controllers with different optimization control objectives. Our approach controls the frequency of when the monitor should be invoked by incorporating a bounded memory buffer that stores events that need to be monitored. The controllers attempt to improve time predictability and maximize memory utilization, while ensuring the soundness of the monitor simultaneously. Unlike the existing approaches based on static analysis, our approach is highly scalable and well-suited for reactive systems that are required to react to stimuli from the environment in a timely fashion. Our thorough experiments using two case studies (a laser beam stabilizer for aircraft tracking, and a Bluetooth mobile payment system) demonstrate the advantages of using controllers to achieve low variation in the frequency of monitor invocations, while maintaining maximum memory utilization in highly non-linear environments.

1. INTRODUCTION

Given the complexity of today's computing systems, verification techniques such as model checking and theorem proving may not be able to analyze the system's correctness exhaustively. Testing is a best-effort established method to examine correctness. However, testing scrutinizes only a subset of behaviors of the system. *Runtime verification* [4, 15, 11] (RV) is a complementary technique, where a *monitor* checks at run time whether or not the execution of a system under inspection satisfies a given correctness property. If the monitor observes that the system is about to violate a property, it can trigger a steering method, so the system is led to a safe behavior. The ability of a monitor to evaluate the system's properties at run time and take all the system dynamics as well as environment stimuli into account has made RV an excellent technique to ensure the well-being of computing systems, especially in the domain of embedded safety/mission-critical systems.

The inherent cost of RV is runtime overhead. In the context of embedded systems, this cost by itself is not the main obstacle in augmenting a system with RV technology. The more significant problem is the fact that if events that would potentially invoke the monitor do not occur in a time-predictable manner (e.g., periodic), monitoring tasks can severely intervene the normal system execution, thereby, causing deadline misses and unscheduled resource utilization. To tackle this problem, there has recently been an emerging trend on designing *time-triggered* monitors. Such a monitor is invoked within fixed time intervals, while ensuring soundness. The existing methods incorporate static analysis techniques to ensure minimum instrumentation [2] and execution path-aware adjustment of monitor invocation [14] to decrease the overhead. However, deep static analysis techniques suffer from two drawbacks: they (1) may not scale, and (2) are completely blind to system dynamics and environment actions at run time, especially in *reactive* systems.

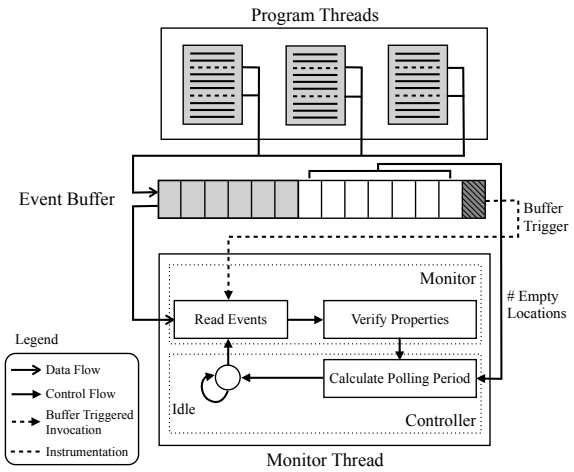


Figure 1: Outline of the proposed controller design.

With this motivation, in this paper, we focus on designing an RV technique, where the monitor should react to system dynamics and environment actions, while taking resource limitations into account. We, in particular, target reactive embedded real-time systems, where time-predictability plays an important role and memory usage is limited by physical constraints. To this end, we require the following:

1. The monitor is not invoked by occurrence of each event that may change the valuation of properties and rather invoked within time intervals, called the *polling period*. In order to enforce property violation detection latency, the polling period cannot be greater than some value given as a system parameter. The monitor is also required to maintain *minimum jitter* in changing the polling period.

2. The monitor must be *sound*; i.e., false-positives and false-negatives are not acceptable.

3. Since the monitor is invoked within time intervals, multiple events of interest may happen between two monitor invocations. Thus, the program under inspection must be instrumented such that the events of interest between the two invocations are buffered. We assume that the program under inspection provides only a bounded-size buffer. This buffer is required to be filled with *maximum utilization*.

In order to achieve the above requirements and make the polling period resilient to non-uniform environment actions, the monitor must be able to learn, predict, and adapt to the environment stimuli at run time. To design such a monitor, we utilize the rich literature of control theory to enforce the three aforementioned requirements. The controller (see Figure 1) executes within the monitor thread. With every invocation of the monitor, the controller determines when the next invocation should occur to satisfy the memory utilization and time predictability objectives. In order to maintain soundness, no events should be dropped from the buffer. Thus, when the buffer is full, the monitor invocation is automatically triggered ahead of its scheduled invocation. We design five controllers:

- A PID feedback controller with variable sampling period for systems in which events of interest are expected to occur linearly. This controller aims at maximizing memory utilization.

- Four fuzzy controllers for handling systems where events of interest occur in a non-linear fashion. Moreover, each controller targets achieving a different objective with respect to our problem:

  – The first fuzzy controller attempts to maximize memory utilization, similar to the PID controller.

  – The second fuzzy controller targets both memory utilization and time predictability. The controller attempts to balance between (1) polling periods that would minimize the empty spaces in the buffer, and (2) choosing intervals as close as possible to the mean of all previous intervals.

  – Fuzzy controllers 3 and 4 attempt to maintain an upper bound on the variance of intervals. Their difference is in their internal decision process.

We conduct two thorough case studies. The first case study is on a Bluetooth mobile payment system. This system is highly non-linear and our experiments clearly demonstrate the advantages of using our controllers to achieve low variation in the monitor polling period, while maintaining maximum memory utilization in highly non-linear environments. The second case study is on a laser beam stabilizer (LBS) used for aircraft tracking. The RV system for LSB aims at monitoring the location of the laser beam, where aircraft movements are the environment actions. The laser beam control software works periodically and although aircraft movements happen non-linearly, the buffer gets filled up within periods uniformly. This characteristic may unnecessecitate a monitor controller. However, we conduct this experiment to demonstrate that the controlled monitor performs as well as the uncontrolled monitor. That is, our controller introduces negligible disturbance to the system.

Organization. The rest of the paper is organized as follows. In Section 2, we formally state the monitoring objectives. Section 3 recaps the basic concepts on PID and fuzzy controllers. Our controller design choices are explained in Section 4. We present our case studies and experimental results in Section 5. Related work is discussed in Section 6. Finally, we make concluding remarks and discuss future work in Section 7.

2. PROBLEM DESCRIPTION

Expressing logical properties of a system normally involves a set of program variables whose value may change over time. We call such change of value an *event*. Monitoring an event involves invoking a process (called the *monitor*) that evaluates the properties associated with that event at run time. This paper is concerned with the problem of runtime verification of reactive systems, where the monitor is required to exhibit the following features simultaneously:

Soundness For verification to be *sound*, all events should be monitored.

Time predictability Since invocation of the monitor interrupts the program execution, we require that these

interruptions are predictable with respect to time. This requirement assists in achieving more accurate system-wide scheduling.

Resource utilization The monitor may use bounded memory space to buffer events. We require maximum utilization of this buffer.

We now formulate the above constraints. Let R be a reactive system with limited memory under inspection and Φ be a set of logical properties (e.g., in LTL), where R is expected to satisfy Φ. Since, system R has limited memory, we assume that the number of events that it can buffer for monitoring has an upper bound B.

Let $E = e_1 e_2 \cdots e_n$ be a given finite sequence of events that can change the valuation of Φ and $T_e = t_{e_1} t_{e_2} \cdots t_{e_n}$ be the finite sequence of timestamps of occurrence of the events, where $n \in \mathbb{N}$. Also, let $M = m_1 m_2 \cdots m_k$ be the output finite sequence of monitor invocations and $T_m = t_{m_1} t_{m_2} \cdots t_{m_k}$ be the finite sequence of timestamps of monitor invocations, where $k \in \mathbb{N}$. We note that k is a variable to be controlled, meaning that depending upon the monitoring policy, k may change. We denote the start time of the monitor by m_0. Thus, we extend T_m as $t_{m_0} t_{m_1} t_{m_2} \cdots t_{m_k}$.

Let function $between(\tau_1, \tau_2)$ be a function that returns all the events that occur between time τ_1 and τ_2:

$$between(\tau_1, \tau_2) = \{e_i \mid \tau_1 < t_{e_i} < \tau_2\} \tag{1}$$

Based on the above description, we say that the monitor is *sound* iff:

$$\forall\, i \in \{1 \cdots k\} : \left|between\left(t_{m_i}, t_{m_{i-1}}\right)\right| \leq B \tag{2}$$

which implies that at no point in time incoming events will overflow the buffer.

We formalize maximization of *memory utilization* as the following objective:

$$\max \left\{ \frac{1}{k} \sum_{i=1}^{k} \frac{\left|between\left(t_{m_i}, t_{m_{i-1}}\right)\right|}{B} \right\} \tag{3}$$

Thus, the objective is to maximize the average memory utilization across the complete run of the monitor.

Let $X = \{X_i \mid 1 \leq i \leq k\}$ be the set, where

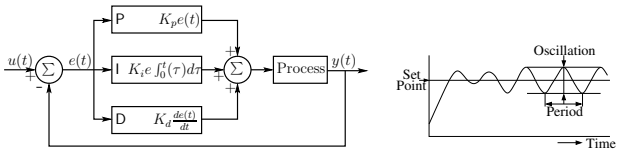
$$X_i = t_{m_i} - t_{m_{i-1}}$$

i.e., each X_i is the amount of time elapsed between monitor invocations m_i and m_{i-1}. Thus, we characterize *time predictability* by the following objective:

$$\min \{V(X) \mid \text{for all possible sets of } X\} \tag{4}$$

where $V(X)$ is the variance of X. In other words, by minimizing the variance of all possible X's, we achieve predictability in the invocation of the monitor.

Observe that the best case minimum variance is zero, which means that for all i, $t_{m_i} - t_{m_{i-1}}$ remains constant. However, if a monitor adopts a constant monitoring frequency, it may be possible to lose soundness in a reactive system, as the rate of occurrence of events depends upon external stimuli, such as environment actions. Furthermore, for memory utilization, the best case is 100% average utilization. However, such a constraint conflicts with the time predictability requirement, since invoking the monitor whenever the buffer is full will result in a variance that is totally controlled by external actions. This discussion clearly illiterates that memory utilization and time-predictability are conflicting requirements.



(a) Structure (b) Tuning through stable oscillation.

Figure 2: PID controller.

Since the sequence of events to be monitored is not given a priori, an optimal monitoring policy that satisfies soundness, time predictability, and high memory utilization cannot be designed before system deployment. In other words, the times and frequency of monitor invocations have to be dynamically adjusted based on the conditions of the system under inspection. Consequently, our goal is to design a runtime *control* mechanism that enforces our objectives (i.e., Equations 2, 3, and 4) simultaneously through identifying T_m (i.e., time of monitor invocations and, hence, k) in a best-effort fashion.

3. BASIC CONTROL THEORY

Since our approach is based on controller design in this section, we recap the concepts of PID controllers in Subsection 3.1 and Fuzzy controllers in Subsection 3.2.

3.1 PID Controllers

A PID feedback controller [16] consist of (1) a *proportional*, (2) an *integral*, and (3) a *derivative* component. An *error signal* $e(t)$ is sampled within fixed time intervals called the *sampling period*. The three components are then applied collectively to $e(t)$ as follows:

$$u(t) = K_P\, e(t) + K_I \int e(t)dt + K_D \frac{d}{dt} e(t) \tag{5}$$

where K_P is the proportional gain, K_I is the integral gain, and K_D is the differential gain. Figure 2(a) demonstrates the structure of a PID controller.

PID controllers are often used to control *linear* systems. One approach to using PIDs is to model the system accurately, so as to deduce ideal gains that ensure controllable behavior. Another method is using experience to configure these controllers; often engineers on site can come up with an initial configuration for PID controllers using well known methods. In this paper, we use the popular Zeigler-Nichols method [20] to tune K_P, K_I, and K_D. We begin by disabling K_I and K_D, and increasing K_P gradually until oscillation begins with a constant amplitude (see Figure 2(b)), where

$$e = \textit{SetPoint} - \textit{Feedback Reading}$$

SetPoint is the desirable set point and *Feedback Reading* is output of the plant. The gain at which oscillation begins is called the ultimate gain K_U. Using K_U and the oscillation period T_U, we can determine the values of K_P, K_I, and K_D by substituting in the Zeigler-Nichols rules.

The main drawback of PID controllers is that their performance in non-linear systems is variable, as they are inherently linear. The engineer is, hence, faced with the trade-off

of decreasing overshoot versus decreasing settling time.

3.2 Fuzzy Controller

A fuzzy controller is often considered as a real-time expert system that relies in part on the system operator's expertise in the form of situation/action rules [7]. This differs from PID controllers in that fuzzy controllers mainly describe what the system's operator would do in different situations based on a set of *fuzzy* conditions, which resembles our human perception of conditions/actions such as the control we employ while driving. This fundamental basis enables fuzzy controllers to outperform PID controllers in non-linear systems.

3.2.1 Fuzzy Logic

The first function of a fuzzy controller is to transform a discrete measured value (called a *crisp* value) to a *fuzzy* value. We first define *fuzzy sets* as sets, whose elements have degrees of membership to that set. For a universe $\mathcal{U}$, each fuzzy set is associated with a *membership function*, which maps each value $u \in \mathcal{U}$ to a value within the interval $[0, 1]$. value That is

$$\mu : \mathcal{U} \rightarrow [0, 1]$$

An *if-then* implication rule is generally of the form "if X is A then Y is B", where X is a fuzzy variable, A is an antecedent fuzzy set, Y is an output fuzzy variable and B is a consequent fuzzy set. In fuzzy logic, there are many methods with which we can perform inference based on this implication. We use *scaled inference*, which has the advantage of preserving the shape of the membership function. In scaled inference, an implication is represented by scaling the consequent membership function with the degree of membership of the crisp value in the antecedent function. Thus, for an if-then rule, scaled inference S is calculated as follows:

$$\mu_S(x, y) = \mu_A(x) \cdot \mu_B(y)$$

where x is the the measured crisp value of the fuzzy variable X and y is the output crisp value of fuzzy variable Y. This process of evaluating the above equation is called *firing*.

Applying scaled inference to support multiple rules is our goal in fuzzy controllers, since we need to control the system using a set of rules that account for the expert's response in different situations. There are two ways to apply scaled inference to multiple rules: (1) composition-based inference, and (2) individual-rule-based inference. The difference between these two methods is that in individual-rule-based inference, each rule is fired individually and then a union is calculated for all rules. Composition-based inference calculates the union first and then fires the resulting set. The output for both methods is the same when using scaled inference. Thus, for a given $u \in \mathcal{U}$, the result of firing the set of rules using individual rule-based inference is obtained by the following equation:

$$\mu_I(u) = \max_k \{\mu_{A^{(k)}}(x) \cdot \mu_{B^{(k)}}(u)\} \quad (6)$$

where k is the enumerator over the set of rules, and x is the crisp input.

3.2.2 Structure of a Fuzzy Controller

Figure 3 shows the structure of a typical fuzzy controller [7]. A fuzzy controller consists of the following components:

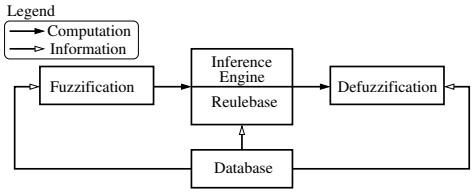


Figure 3: Structure of a Fuzzy Controller [7].

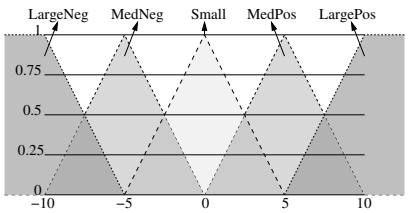


Figure 4: Membership functions of input fuzzy sets.

Fuzzification When a fuzzy controller receives a measured value from the system, this value must be *fuzzified*, so that its membership to the associated fuzzy sets could be determined. As mentioned earlier, in this paper, we use scaled inference for fuzzification.

Knowledge base This component consists of a *rulebase* and a *database*. The rulebase contains the set of rules including the antecedents and consequents. The database contains the membership functions of fuzzy sets. In common practice there are five fuzzy sets for each fuzzy variable: LargeNeg, MedNeg, Small, MedPos, and LargePos. The membership functions for these sets are *lambda-type* functions, with the exception of LargeNeg and LargePos, which are Z-type and S-type respectively [17]. An example of these functions is shown in Figure 4.

Inference engine The inference engine employs either composition-based inference or individual-rule-based inference, described above. The latter is more widely used in fuzzy control since it is computationally more efficient and uses less memory.

Defuzzification This component transforms the output of the inference engine into one single point-wise value. This value is then applied to the system to complete the control loop. The most widely used method for defuzzification is *gravity defuzzification*, which calculates the center of gravity for $\mu_I(u)$ in Equation 6. The output crisp value u^* is calculated as follows:

$$u^* = \frac{\int_{-\infty}^{+\infty} u \cdot \mu_I(u)\, du}{\int_{-\infty}^{+\infty} \mu_I(u)\, du} \quad (7)$$

4. MONITOR CONTROLLER DESIGN

This section presents in detail the design of our controllers based on the objectives in Equations 2, 3, and 4. As discussed in the introduction (see Figure 1), the program under inspection can be multi-threaded running on a single-core machine. We instrument the program, so that it enqueues the events in a bounded-size buffer whenever they are modified. The monitor is a separate thread within the program's

process, that executes at a higher priority than the program threads. It is idle for a period of time while events are being enqueued in the buffer, and once invoked, it preempts the program threads due to having a higher priority. The monitor then reads all events and verifies a set of predefined logical properties. Once the verification is complete, the monitor enters idle mode again, and awaits the refilling of the event buffer.

The controller (see Figure 1) executes within the monitor thread. With every invocation of the monitor, the controller determines when the next invocation should occur to satisfy Equations 3 and 4. In order to maintain soundness, no events should be dropped from the buffer. Thus, when the buffer is full, the monitor invocation is automatically triggered ahead of its scheduled invocation to ensure soundness. This is called a *buffer-triggered* invocation. Subsections 4.1–4.5, describe the design of our PID and four fuzzy controllers.

4.1 PID Controller

Since we deal with reactive systems, overshoots are inevitable. In the context of our problem, an overshoot refers to the event that the buffer overflows before the monitor is invoked. Our design supports a safety threshold for buffer utilization. For instance, a controller with an 80% safety threshold will attempt to keep the buffer 80% utilized in every monitor invocation.

- **Input.** In order to achieve maximum memory utilization (Equation 3), the controller should target maintaining a completely full buffer up to the safety threshold at every invocation of the monitor. Thus, the input error signal to the controller is the number of empty locations in the buffer at the moment the monitor is invoked. The safety threshold is also a configuration parameter of the controller that can be altered depending upon the system requirements. Hence, the input error signal is formally the following:

$$e(t_{m_i}) = B \times S - \left|between\left(t_{m_i}, t_{m_{i-1}}\right)\right|$$

  where B is the buffer size, S is the safety threshold percentage, t_{m_i} is the timestamp of the current invocation of the monitor, $t_{m_{i-1}}$ is the timestamp of the last invocation of the monitor, and $between\left(t_{m_i}, t_{m_{i-1}}\right)$ is the set of events received between the two timestamps (defined in Equation 1).
- **Output.** Initially the controller schedules the monitor to run after a predefined idle period. The goal of the controller is to change this initial period dynamically to maintain zero error. We refer to this period as the *polling period*, i.e. the period with which the monitor *polls* the application for new events. Thus, the output of the controller is the offset (positive or negative) with which to change the polling period to maintain zero error.
- **Tuning.** The controller is tuned using the Ziegler-Nichols method. The proportional, integral and derivative gains are $0.6K_u$, $2K_p/T_u$, and $K_pT_u/8$, respectively, where T_u is the period of constant oscillation and K_u is the proportional gain at which oscillation occurs (see Figure 2(b)).

The controller updates are not periodic due to the fact that the period depends on the output of the controller itself, and also due to buffer triggered invocations. Thus, the integral component is calculated as in a *variable sampling period* PID [9].

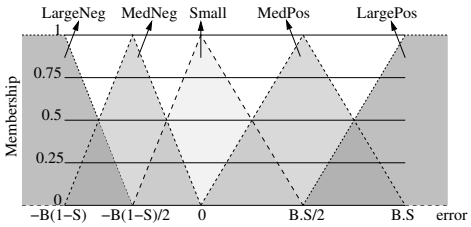


Figure 5: Membership functions of the error fuzzy sets.

4.2 Fuzzy Controller 1

The first fuzzy controller attempts to maximize memory utilization, similar to the PID controller.

- **Input.** The input to the controller is the fuzzy variable E_B representing the number of empty locations in the buffer. The crisp value for this variable is calculated the same way $e(t)$ is calculated in the PID controller:

$$E_B = B \times S - \left|between\left(t_{m_i}, t_{m_{i-1}}\right)\right|$$

  There are 5 fuzzy sets for the error variable based on lambda-type functions as shown in Figure 5. The *Small* set has a peak at zero error, with the left x-intercept at $\frac{-B(1-S)}{2}$ and the right x-intercept at $\frac{B\times S}{2}$. The reason these points are not symmetric is that the largest positive error that could be reached is $B \times S$, which denotes that the buffer is completely empty. However, the largest negative error is $-B(1-S)$, since buffer triggering will prevent the error from exceeding that value.
- **Output.** The output of the controller is the offset value from the current polling period, which we denote as Δ_X. The membership functions for the output variable are standard lambda-type functions similar to those in Figure 4, with centers at -1, -0.5, 0, 0.5, and 1, respectively. The output is multiplied by a factor depending on the nature of the system.
- **If-then rules.** The *if-then* rules for the controller are as follows:
  - if E_B is LargeNeg, Δ_X is LargeNeg
  - if E_B is MedNeg, Δ_X is MedNeg
  - if E_B is Small, Δ_X is Small
  - if E_B is MedPos, Δ_X is MedPos
  - if E_B is LargePos, Δ_X is LargePos
- **Fuzzification, inference, and defuzzification.** The fuzzification module uses scaled inference and the inference engine uses individual rule based firing. The defuzzification module uses the center of gravity method to calculate the output value. The calculations involved in applying these methods are minimal, with the advantage that most of the calculations can be precomputed before the system executes, thus decreasing the processing overhead of the controller in run time.

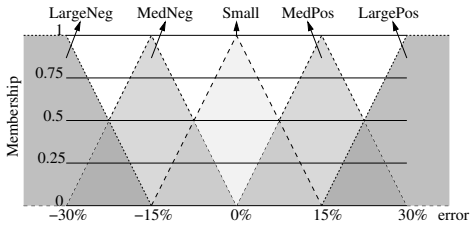


Figure 6: Membership functions of $E_{\bar{X}}$

4.3 Fuzzy Controller 2

Fuzzy controller 2 targets both memory utilization and time predictability. The approach of this controller is to balance between choosing a polling period that would minimize the error in the buffer, and choosing a polling period of a value as close as possible to the mean of all previous polling periods. The second condition ensures that the variance of the polling period is minimized.

- **Input.** In addition to E_B, we introduce a new fuzzy variable $E_{\bar{X}}$ to control the polling period variance. $E_{\bar{X}}$ represents the difference between the current polling period and the mean of all previous polling periods. The crisp values of $E_{\bar{X}}$ is calculated as follows:

$$E_{\bar{X}} = \frac{X - \bar{X}}{\bar{X}} \tag{8}$$

  where X is the current polling period and $\bar{X}$ is the mean of all previous polling periods. $E_{\bar{X}}$ is a percentage so as to make the controller computations independent of the time scale at which the system operates. The membership functions for this variable are standard lambda-type, as shown in Figure 6. These values are configuration parameters and can be changed according to the user requirement. The choice of the range -30% to 30% produces low variation in polling periods, and consequently high time predictability.
- **Output.** The output of the controller is the same as Fuzzy 1 (i.e., the offset value from the current polling period).
- **If-then rules.** Since the controller is now targeting two simultaneous goals involving two fuzzy variables (E_B and $E_{\bar{X}}$), with 5 fuzzy sets each, there are 25 possible if-then rules. Table 1 shows the consequent fuzzy set of each rule based on the combination of the two antecedent fuzzy sets, where the columns are $E_{\bar{X}}$ fuzzy sets, the rows are E_B fuzzy sets, and LN, MN, S, MP, and LP are abbreviations of LargeNeg, MedNeg, Small, MedPos, and LargePos, respectively. The mapping above is symmetric, meaning that no variable has a more significant effect on the output than the other.This mapping is a configuration parameter and could be changed according to the system requirements.

4.4 Fuzzy Controller 3

Instead of minimizing the variance, fuzzy controller 3 attempts to maintain an upper bound on the variance. Thus, this controller adds a configuration parameter to fix that upper bound. However, since the mean of polling period is not known a priori and changes during the program's execution, the value that the user chooses as an upper bound on

		$E_{\bar{X}}$ *Fuzzy sets*				
		LN	MN	S	MP	LP
E_B *Fuzzy sets*	LN	S	MN	LN	LN	LN
	MN	MP	S	MN	LN	LN
	S	LP	MP	S	MN	LN
	MP	LP	LP	MP	S	MN
	LP	LP	LP	LP	MP	S

Table 1: Symmetric mapping of input variables in if-then rules.

the variance does not represent the actual variation in the polling period. For instance, a variance of 10 for a polling period mean of 1000 is an indicator for very high predictability and low variation. However, the same variance when the mean is 10 shows very high variation in polling times. This has led to using the coefficient of variation as the metric that has an upper bound. The coefficient of variation is calculated as $c_v = \sigma_X/\bar{X}$, where σ_X is the standard deviation of all previous polling periods, and $\bar{X}$ is the mean. Since the polling period mean will never be zero, c_v is a safe metric. The coefficient of variation enables the user to dictate the required shape of the distribution of polling periods; i.e. whether to have a broad or narrow curve around the mean.

Fuzzy controller 3 adopts a fuzzy variable E_{c_v} which is simply the last polling period of the controller. Let all polling periods since the start of execution be the sequence $X = X_1X_2X_3\cdots X_N$, where X_N is the last polling period. Since the upper bound of c_v is fixed by a constant k, the controller needs to determine the best X_{N+1} that guarantees k as the coefficient of variation. We expand the coefficient of variation formula, so that we can obtain the value of X_{N+1}. This led to deriving a quadratic equation whose roots are the values for X_{N+1} that produce $c_v = k$. To simplify the equation, we define γ as the following quantity:

$$\gamma = \left(\frac{N + 1 + k^2N}{N\,(N+1)^2}\right) \tag{9}$$

where N is the number of polling periods in the sequence X. The quadratic equation to calculate X_{N+1} is as follows:

$$\left(\frac{1}{N} - \gamma\right) X_{N+1}^2 - \left(2\,\gamma \sum_{i=1}^{N} X_i\right) X_{N+1} + \left(\frac{1}{N}\sum_{i=1}^{N} X_i^2 - \gamma \sum_{i=1}^{N} X_i\right) = 0 \tag{10}$$

If Equation 10 has complex roots, then it is not possible for X_{N+1} to lower the coefficient of variation down to k. In this case, fuzzy controller 3 falls back to Fuzzy 2, attempting to minimize the variance all together. This will continue until the coefficient of variation is low enough that it can be controlled within the upper bound.

If the two roots of Equation 10 are real values, the mean is a number between these two roots. The membership functions for E_{c_v} are designed in such a way that it tries to keep the polling period between the two roots, with preference to the mean. Figure 7 shows how these functions are defined, where r_1 and r_2 are the roots of Equation 10. The *Small* membership function has a peak at the mean μ, has a left x-intercept at r_1, and a right x-intercept at r_2. *MediumNeg* and *MediumPos* are centered around r_1 and r_2 with intercepts at half the distance between the mean and the roots. This maintains fairness in treating the polling period

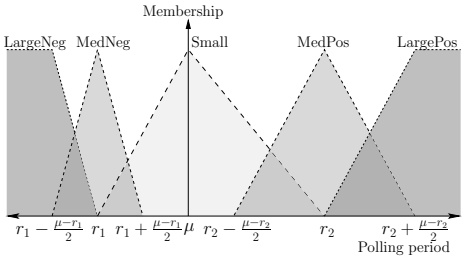


Figure 7: Membership functions of E_{c_v}

regardless of which root it is closer to.

The mapping in the *if-then* rules in this controller is similar to the mapping in Fuzzy 2 as shown in Table 1, which maintains a balanced trade-off between memory utilization and time predictability.

4.5 Fuzzy Controller 4

Fuzzy controller 4 is essentially the same as Fuzzy 3, with the exception of the mapping for the *if-then* rules. In this controller, the mapping gives preference to controlling memory utilization when E_B is a large negative value, even if that contradicts with the time predictability requirement. Table 2 shows the modified mapping. This mapping enables Fuzzy 4 to react faster to large overshoots in the error, thereby maintaining stability and giving room for the controller to work on a balanced trade-off.

5. IMPLEMENTATION AND EXPERIMENTAL RESULTS

In order to analyze the performance of our controllers, we have conducted two case studies: (1) a Bluetooth mobile payment, and (2) a Laser beam stabilizer for aircraft tracking. Each case study involves using different controllers with different configurations.

Our experiments are designed based on three factors:

1. **Controller type.** We incorporate seven controllers in our experiments: PID, Fuzzy 1, Fuzzy 2, Fuzzy 3 with target coefficient of variation $c_v = 0.4$, Fuzzy 3 with $c_v = 0.2$, Fuzzy 4 with $c_v = 0.4$, and Fuzzy 4 with $c_v = 0.2$.
2. **Buffer size (B).** We experiment with three different buffer sizes: 20, 40, and 60 events.
3. **Safety threshold (S).** We experiment with two safety thresholds: 80%, and 90%.

Hence, there is a total of 42 configurations to test all different combinations of the above three factors. For both case studies, we carried out multiple runs with randomization to provide statistical confidence and remove any hidden effects.

The five measurement metrics that we observe are:

		E_{c_v} Fuzzy sets				
		LN	MN	S	MP	LP
E_B Fuzzy sets	LN	MN	LN	LN	LN	LN
	MN	S	MN	MN	LN	LN
	S	LP	MP	S	MN	LN
	MP	LP	LP	MP	S	MN
	LP	LP	LP	LP	MP	S

Table 2: Asymmetric mapping of input variables in if-then rules.

1. **Error mean.** This is the mean number of empty buffer locations at every invocation of the monitor. This value is a measure of the memory utilization of the monitor, i.e. the lower the value, the more utilized the memory.
2. **Polling period coefficient of variation (C_v).** This value is a measure of time predictability, i.e. the lower the value, the closer polling periods are to their mean, and hence, more time predictability.
3. **Context switches.** This is the number of invocations of the monitor during a run of an experiment. This value is a measure of the overhead introduced by the monitor.
4. **Buffer triggers.** This is the number of buffer-triggered monitor invocations. This value is a measure of the quality of the controller in the sense that a well designed controller should not overshoot frequently causing many buffer triggers.

5.1 Bluetooth Mobile Payment (BTP)

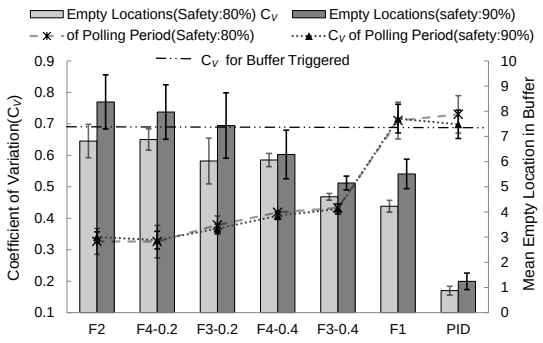
Mobile payment is becoming increasingly popular and gaining assurance about the soundness of such a system is an essential requirement. Whether payment is through WiFi, Bluetooth, or NFC, the process relies on a payment hub that communicates with smartphones to process payments, which includes establishing a connection with devices. The hub establishes a connection with these devices and sends/receives messages. We monitor these messages at the operating system level to ensure that every message gets a response and no error.

Our experimental platform is single core machines running under the QNX real-time operating system hosting a Bluetooth 2.1 adapter. Our implementation follows the outline in Figure 1, with the exception that there is a single program thread responsible for extracting events using the QNX TraceEvent API and queuing them into the buffer. We use an experimental dataset that has been collected in a shopping mall [10]. It includes Bluetooth contact traces from employee devices around the cashier area of a certain store. To provide statistical confidence in the results, we run 9 replicates of a trial, where each trial consists of running all 42 possible combinations of the experimental factors.

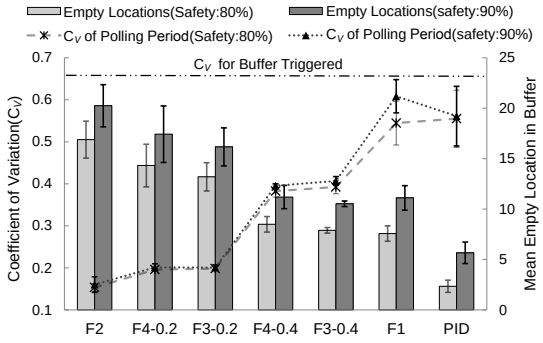
5.1.1 Analysis of Time Predictability

Figure 8 shows the average polling period coefficient of variation C_v across all 9 replicates for buffer sizes 20, 40, and 60. As can be seen, Fuzzy 2 exhibits the lowest C_v, since it is designed to control the polling period within $\pm 15\%$ of the mean (see Section 4.3). Fuzzy 3 targets $C_v = 0.2$ (denoted F3-0.2 in the figure) and Fuzzy 4-0.2 show low C_v due to having an aggressive $C_v = 0.2$ goal. In Figure 8(a), Fuzzy 3-0.2 and Fuzzy 4-0.2 fail to meet their goals, scoring a C_v of 0.32 and 0.37. This is due to the 0.2 goal being too aggressive to reach in a buffer of size 20. Note that at higher buffer sizes, these controllers meet their goals, as shown in Figures 8(b) and 8(c)). However, Fuzzy 3-0.4 and Fuzzy 4-0.4 consistently meet their goal ($C_v = 0.4$) across all configurations. Since Fuzzy 1 and PID do not attempt to control C_v, they have the highest values.

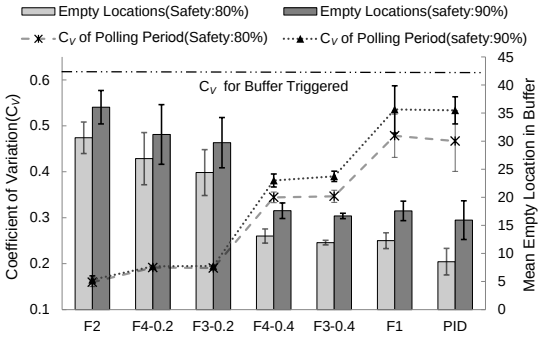
An interesting observation is that for a purely buffer triggered implementation, where no control is involved, the C_v is almost always higher than any controller across all configurations (shown as a horizontal line in all three graphs). In



(a) Buffer size 20



(b) Buffer size 40



(c) Buffer size 60

Figure 8: Polling period coefficient of variation versus error mean for all 7 controllers and both safety threshold values for BTP.

fact, for Fuzzy 2, C_v is less than a third of pure buffer triggered for buffer size 40. This shows the advantage of using controllers to improve time predictability of the monitoring system.

Figure 9 shows the box-plots of the polling periods for different controllers for buffer size $B = 20$ and safety threshold $S = 80$. The figure shows that a purely buffer triggered implementation exhibits the highest variability. This is expected since this implementation responds transparently to the non-linearity of the system. The second highest variability is present in the PID controller, explained by the inability of the PID to adapt to a non-linear system. Again, it can be seen that using Fuzzy 1, which has the same goal as the PID, can drastically improve the stability of the controller. The lowest variability is - as expected - due to Fuzzy 2, Fuzzy 3-0.2, and Fuzzy 4-0.2.

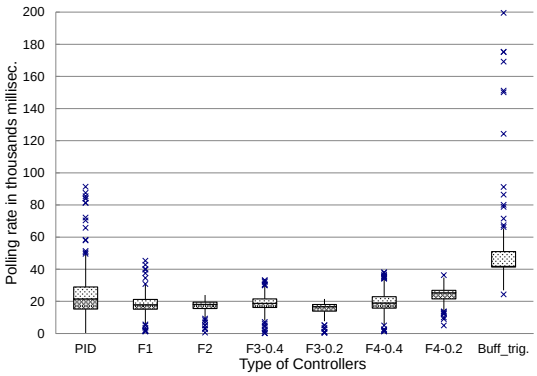


Figure 9: Box-plot of polling periods for different controllers of BTP.

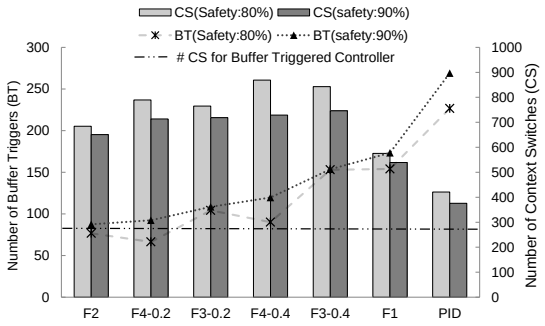
5.1.2 Memory Utilization

Figure 8 shows the mean number of empty buffer locations (error) across all 9 replicates for buffer sizes 20, 40, and 60. The 95% confidence intervals for the error mean are also shown. As can be seen, the PID controller consistently has the lowest error mean, and thus provides the highest memory utilization. The error mean for Fuzzy 1 is also low, and comparable to that of the PID when the buffer size increases (see Figures 8(b) and 8(c)). Fuzzy 2 exhibits a consistently high error mean. This is due to the fact that Fuzzy 2 is designed to be aggressive in maintaining a low polling period C_v, which comes at the cost of error. This also applies to Fuzzy 3 aggressively targeting $C_v = 0.2$ (denoted F3-0.2) and Fuzzy 4 targeting $C_v = 0.2$. However, Fuzzy 3-0.4 and Fuzzy 4-0.4 perform comparably to PID and Fuzzy 1, especially with increased buffer size. This stems from the fact that these controllers have a relaxed goal (i.e., $C_v = 0.4$) and are thus more capable of maintaining a low error mean. The error mean of a 90% safety threshold controller is consistently higher than that of 80% simply due to having more space to control in the buffer.

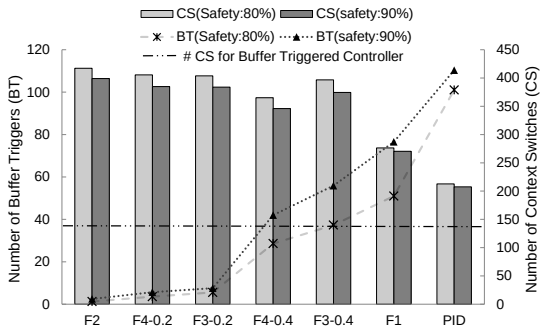
The error mean trend is further clarified in Figure 10. This figure shows the number of buffer triggers occurred for every controller. It appears that the reason PID has such a low error mean is because it consistently has the highest number of buffer triggers. This is an indication that the PID controller is unable to adapt to the non-linear nature of the system, and as a result is overshooting considerably more than any other controller. This also shows that Fuzzy 1, although having a slightly higher error mean, is more capable of adapting to the change in the system without frequently overshooting. The other fuzzy controllers have a low number of buffer triggers due to their tendency to remain stable.

5.1.3 Execution Time

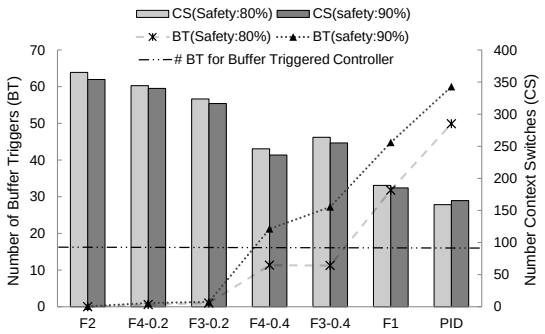
We next study the effect of using different controllers on the execution time of the program. The execution time includes the CPU time, time of kernel calls, CPU time by child processes, and time of kernel calls made by child processes. We compare this time to the execution time of the program without any monitoring functionality. Note that for this comparison, there is no verification overhead included in the calculation. We assume that the verification overhead



(a) Buffer size 20



(b) Buffer size 40



(c) Buffer size 60

Figure 10: Number of buffer triggers vs. number of context switches for 7 controllers and both safety threshold values for BTP

can be offloaded to a seperate processing unit.

Since execution time results are subject to many factors affecting variability, we attempt to estimate the worst case overhead introduced by our controllers based on the maximum execution time of the program with our controllers relative to the minimum execution time of the program without any monitoring. This comparison shows that in the worst case, PID and F1 introduce a 19% increase in execution time. However, other fuzzy controllers average around 10%.

5.1.4 Additional Observations

- **Thread context switching.** A high number of buffer triggers indicates that the system is overshooting frequently and, thus, is more frequently filling the buffer completely. This results in a lower number of context switching. Figure 10 illustrates the number of context switches and a trend that is related to the number of buffer triggers. The figure also shows a horizontal line denoting the number of context switches performed by a purely buffer triggered solution, which is expected to be the lower than any controller-based approach.
- **Time-predictability vs. memory utilization.** The trend of polling period coefficient of variation C_v versus error mean magnifies the trade-off between time predictability and memory utilization. The results show that Fuzzy 3-0.4 and Fuzzy 4-0.4 exhibit the best balance between the two goals consistently across configurations.
- **Resilience to overshoots.** Figure 10 shows that all fuzzy 4 controllers present an advantage over fuzzy 3 in terms of number of buffer triggers. Since these controllers are designed to be more aggressive when an overshoot occurs or is about to occur, their behavior demonstrates a more conservative approach with respect to buffer triggers.

5.2 Laser Beam Stabilization (LBS)

LBS technology is used in aircraft targeting, surveillance and laser-based communication systems. A control system stabilizing a laser beam is required to maintain safety properties, such as ensuring that the offset of the laser from the target should not exceed a certain value. In this case study, we use the Quanser laser beam stabilization system with a mounted motor that produces undesirable vibrations affecting the stability of the laser. When the photodetector registers the laser at an offset larger than 0.01mm, an event is queued into the buffer. Our experiments are based on 9 replicates and we target $C_v = 0.6$ for Fuzzy 3 and 4 controllers. This demonstrates how a more relaxed constraint affects the response of the controller.

5.2.1 Time predictability

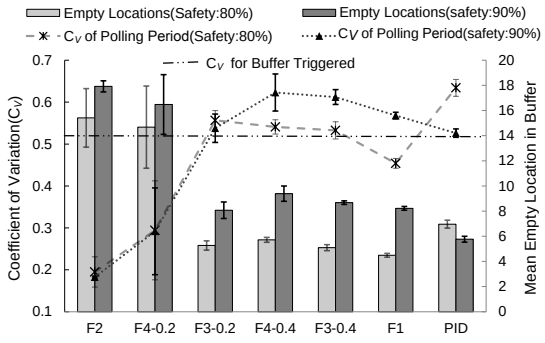
Figure 11 shows the results of the experiments on buffer size of 40. The trend of C_v for polling period versus error mean in Figure 11(a) is similar to that of the Bluetooth experiment. Fuzzy 3-0.2 scores a much higher C_v than its goal (0.6 vs. a goal of 0.2). However, Fuzzy 4-0.2 is closer to its goal, achieving $C_v = 0.3$. This is due to the periodic nature of the oscillations introduced by the motor, which coupled with the aggressiveness of Fuzzy 4 at high errors, enables it more quickly to reach low error and focus on controlling the coefficient of variation. An interesting observation is that C_v of a purely buffer triggered implementation is on average 0.59, which is less than all controllers except for Fuzzy 4-0.2 and Fuzzy 2. This is due to the periodic nature of the events, which enables a purely buffer-triggered solution to naturally produce a lower C_v. Fuzzy 2 and fuzzy 4-0.2, however, are more aggressive in maintaining a low C_v and, thus, they outperform pure buffer triggered.

5.2.2 Memory Utilization

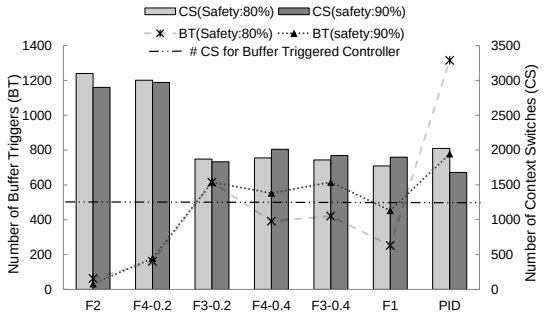
Figure 11(a) shows that low C_v comes at the cost of the error mean. This is contrasted with the number of buffer triggers in Figure 11(b), which shows that PID has the highest number.

5.2.3 Additional Observations

- **Tuning cost.** In Figure 11(b), the difference between the number of buffer triggers for PID when safety



(a) Polling period coefficient of variation versus error mean



(b) Number of buffer triggers versus number of context switches

Figure 11: Results of all 7 controllers at buffer size 40 and both safety threshold values for LBS

is 80% versus 90% is large. This is due to the sensitivity of PID controllers to tuning. Compared to Fuzzy 1 which attempts the same objective, Fuzzy 1 appears to be more consistent. This is further supported by our results for different buffer sizes not shown here.

- **Controller instability.** In Figure 11(a), the trend of C_v for 80% versus 90% safety thresholds is reversed for PID. This is due to the instability of the PID, causing it to revert more to buffer triggers (see Figure 11(b)). This causes it to actually produce a lower C_v at 90% because, in that case, it is closer to a pure buffer triggered controller. This is why the resulting C_v is almost the same as that of pure buffer triggered.

6. RELATED WORK

The main focus of classic event-based runtime verification [13, 3, 18] is to reduce the monitoring overhead through improved instrumentation [8, 5], static analysis [1], and efficient monitor generation [6]. Huang, et al. [12] propose a control-theoretic based software monitoring technique. In this work, cascaded PID controllers are used to temporarily disable monitors in order to keep monitoring overhead below a user-defined threshold. This results in an unsound monitor, since events are being dropped when a particular monitor is disabled. To tackle this problem, Bartocci et al. [19] augment the method presented in [12] with a hidden Morkov model (HMM) to fill in the gaps in event sequences. However, both these methods require tuning of PID controllers and training of HMM in [19]. While the context of our work is different, our approach utilizes fuzzy controllers that are more suited to non-linear systems and require less efforts in tuning.

In the context of time-predictability, in time-triggered runtime verification [2] a monitor periodically samples the system state and verifies critical properties of the system. The time-triggered approach involves the problem of finding an optimal sampling period (equivalent to polling rate in this paper) to minimize the size of auxiliary memory required, so that the monitor can correctly reconstruct the sequence of program state changes. [14] uses symbolic execution to compute the sampling period at run time. However, static analysis is impractical to use in large systems, and the technique is inapplicable in reactive non-linear systems, which are the focus of our work.

7. CONCLUSION

Gaining assurance about the correctness of embedded systems has always been an active and challenging area of research in computing technology. In this paper, we concentrated on designing a scalable approach for runtime verification of reactive embedded systems with three objectives: soundness, minimum jitter in monitor invocation frequency, and maximum memory utilization. To this end, we leveraged the rich literature of control theory. In particular, we designed a PID and four different fuzzy controllers, each targeting a different set of objectives. Our experiments on two embedded systems (a laser beam stabilizer and a Bluetooth payment system) show that our controller-based approach is quite effective and scalable with minimal runtime overhead. In particular, we observed that for reactive systems, where the environment stimuli occur non-linearly, a fuzzy controller reacts the best with respect to achieving time-predictability in monitor invocations. Moreover, the fuzzy controller that attempts to maintain an upper bound on the variance monitor invocation frequency, in most cases provides the best balance between time-predictability and memory utilization. In addition, the runtime overhead of our approach is on average 10%, which is highly reasonable.

For future work, we are planning to investigate employing static analysis techniques such as symbolic execution, so our controllers are also aware of the structure of the system under inspection. Another interesting research direction is to design controllers for monitoring distributed embedded systems.

8. REFERENCES

[1] E. Bodden, L. Hendren, and O. Lhoták. A staged static program analysis to improve the performance of runtime monitoring. In *Proceedings of the 21st European conference on Object-Oriented Programming*, ECOOP'07, pages 525–549, Berlin, Heidelberg, 2007. Springer-Verlag.

[2] B. Bonakdarpour, S. Navabpour, and S. Fischmeister. Time-triggered runtime verification. *Formal Methods in Systems Design (FMSD)*, 2013. To appear.

[3] F. Chen and G. Roşu. Java-MOP: A monitoring oriented programming environment for java. In *Tools and Algorithms for the construction and analysis of systems (TACAS)*, pages 546–550, 2005.

[4] S. Colin and L. Mariani. *Run-Time Verification*, chapter 18. Springer-Verlag LNCS 3472, 2005.

[5] M. d'Amorim and K. Havelund. Event-based runtime verification of java programs. *SIGSOFT Softw. Eng. Notes*, 30(4):1–7, May 2005.

[6] M. d'Amorim and G. Roşu. Efficient monitoring of ω-languages. In *Proceedings of the 17th international conference on Computer Aided Verification*, CAV'05, pages 364–378, Berlin, Heidelberg, 2005. Springer-Verlag.

[7] D. Driankov, H. Hellendoorn, and W. Reinfrank. *An introduction to fuzzy control*. Springer-Verlag New York, Inc., New York, NY, USA, 1993.

[8] Matthew B. Dwyer, Alex Kinneer, and Sebastian Elbaum. Adaptive online program analysis. In *Proceedings of the 29th international conference on Software Engineering*, ICSE '07, pages 220–229, Washington, DC, USA, 2007. IEEE Computer Society.

[9] P. Galan. Temperature control based on traditional pid versus fuzzy controllers.

[10] A. Galati and C. Greenhalgh. Human mobility in shopping mall environments. In *Proceedings of the Second International Workshop on Mobile Opportunistic Networking*, pages 1–7. ACM, 2010.

[11] D. Giannakopoulou and K. Havelund. Automata-Based Verification of Temporal Properties on Running Programs. In *Automated Software Engineering (ASE)*, pages 412–416, 2001.

[12] X. Huang, J. Seyster, S. Callanan, K. Dixit, R. Grosu, S. A. Smolka, S. D. Stoller, and E. Zadok. Software monitoring with controllable overhead. *Software tools for technology transfer (STTT)*, 14(3):327–347, 2012.

[13] O. Kupferman and M. Y. Vardi. Model Checking of Safety Properties. In *Computer Aided Verification (CAV)*, pages 172–183, 1999.

[14] S. Navabpour, B. Bonakdarpour, and S. Fischmeister. Path-aware time-triggered runtime verification. In *Runtime Verification (RV)*, pages 199–213, 2012.

[15] A. Pnueli and A. Zaks. PSL Model Checking and Run-Time Verification via Testers. In *Symposium on Formal Methods (FM)*, pages 573–586, 2006.

[16] D. E. Rivera, M. Morari, and S. Skogestad. Internal model control: Pid controller design. *Industrial & engineering chemistry process design and development*, 25(1):252–265, 1986.

[17] T. J. Ross. *Fuzzy logic with engineering applications*. Wiley, 2009.

[18] O. Sokolsky, S. Kannan, M. Kim, I. Lee, and M. Viswanathan. Steering of real-time systems based on monitoring and checking. In *Proceedings of the Fifth International Workshop on Object-Oriented Real-Time Dependable Systems*, WORDS '99, pages 11–, Washington, DC, USA, 1999. IEEE Computer Society.

[19] S. Stoller, E. Bartocci, J Seyster, R. Grosu, K. Havelund, S. Smolka, and E. Zadok. Runtime verification with state estimation. In *International Conference on Runtime Verification (RV)*, 2011.

[20] J. G. Ziegler and N. B. Nichols. Optimum settings for automatic controllers. *trans. ASME*, 64(11), 1942.