Zhang *et al. Journal of Cloud Computing* (2024) 13:6
https://doi.org/10.1186/s13677-023-00567-8

Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH

Open Access

Context-aware environment online monitoring for safety autonomous vehicle systems: an automata-theoretic approach

Yu Zhang[1], Sijie Xu[1], Hongyi Chen[1], Uzair Aslam Bhatt[2*] and Mengxing Huang[2]

Abstract

Intelligent Transport System (ITS) is a typical class of Cyber-Physical Systems (CPS), and due to the special characteristics of such systems, higher requirements are placed on system security. Runtime verification is a lightweight verification technique which is used to improve the security of such systems. However, current runtime verification methods often ignore the effects of the physical environment (e.g., the effects of rain, snow, and other weather changes on road conditions), which results in the inability of the monitor to effectively monitor the system according to the changes in the environment. To address this problem, this paper proposes a method for constructing a runtime monitor with environmental context-awareness capability. First, the physical environment factors affecting the system are formally described and constructed into an environment model, then the system statute is transformed into a Büchi automaton, and then a synthesis algorithm combining the environment model and the Büchi automaton is designed based on the network of automatons, and the corresponding monitor is generated. Finally, the proposed method is applied and verified on simulation and real objects. The experimental results show that the monitors generated based on the method of this paper can effectively monitor unsafe events in different environments, thus improving the safety of intelligent driving systems.

Keywords ITS, Runtime monitoring, Environmental modelling, Automata

Introduction

ITS is a complex CPS that combines awareness of the physical environment with intelligent computing. CPS enables interaction and control between physical entities and computing systems by integrating sensors, computing elements and communication networks. CPS has wide applications in various sectors such as transport, healthcare, manufacturing, energy and infrastructure. It relies on advanced technologies such as the Internet of Things (IoT), cloud computing, data analytics and artificial intelligence to achieve real-time data processing, intelligent decision-making and adaptive control. However, due to continuous changes in the system's internal behaviour and environmental conditions, the system's decision-making behaviour may become unpredictable. Therefore, appropriate responses must be made at runtime to ensure system safety [1, 2].

Runtime verification is a lightweight software verification technique that detects anomalies and reports problems by monitoring the system's current trajectory to repair runtime errors. Unlike traditional software reliability assurance techniques, runtime verification is primarily applied to deployed systems and can effectively monitor uncertain context changes [3]. Typically, runtime verification uses temporal logic, such as Linear Temporal Logic (LTL), to describe the properties to be monitored. However, when intelligent systems

*Correspondence:
Uzair Aslam Bhatt
uzairaslambhatti@hotmail.com
[1] School of Computer Science and Technology, Haikou 570228, China
[2] School of Information and Communication Engineering, Haikou 570228, China

face security threats in uncertain environments, real-time monitoring of environmental parameter changes becomes a challenge in generating and operating efficient monitors. Existing runtime verification methods have limited consideration of environmental factors, making it difficult to ensure system safety and reliability [4, 5].

To address the problem of system behaviour mismatch with the actual environment due to environmental influences on intelligent systems, this paper proposes the following approach. The paper abstracts and represents environmental information and transforms the environmental model into a probabilistic finite automaton (PFA). The paper uses LTL to describe system properties and transforms it into a Büchi automaton (BA). Then, the synthesized algorithm for combining environmental modeling with Büchi automata and generating monitors is designed through the idea of synthesis. Finally, the monitor code is instrumented in the executable program to achieve real-time monitoring. The main innovations of our work are as follows:

- We innovatively integrate environment models with system properties, which are accurately represented and translated into monitor form by modeling and quantitatively analyzing the environment.
- By optimizing the monitor insertion algorithm, we succeeded in significantly reducing the average execution time of the monitor, which is only 41.1% of the pre-optimization execution time.

The rest of the paper is structured as follows: Background section introduces the relevant basic theories, which provide the theoretical basis for the runtime verification model and environment modeling method proposed in this paper. Related work section reviews research in related fields. The environment modeling approach is described in detail in Context-aware environmental modeling method section. Monitor generation section describes the monitor build and insert process. In Evaluation section, the validity of the proposed method is demonstrated through simulation and physical experiments. Finally, the work of this paper is summarized in Conclusion section.

Background

This section presents the theoretical foundation for the environment modeling methods used in this paper, by introducing formalization tools for sequential logic, runtime verification techniques, and a framework for runtime verification tools.

Linear temporal logic

Linear Temporal Logic (LTL) is a type of Temporal Logic that represents time as a sequence of states and employs temporal operators to specify constraints that a system must satisfy in the past, present, and future, as well as the temporal relationships between events.

Since its inception, numerous scholars have thoroughly studied and developed LTL. Pnueli [6] first introduced LTL and proposed the syntax and semantics of linear temporal logic. His work laid the foundation for LTL in formal verification and provided a theoretical framework for subsequent research.LTL allows for a more precise expression of the desired system's properties or constraints, given a set of atomic propositions denoted as AP. An LTL formula on the set AP can be recursively defined as follows:

$$\varphi ::= p|\neg\varphi|\varphi_1 \wedge \varphi_2|\varphi_1 \rightarrow \varphi_2|\varphi_1 \leftrightarrow \varphi_2|G\varphi|F\varphi|X\varphi|\varphi_1 U\varphi_2 \mid \varphi_1 R\varphi_2$$

In the above definition, $p \in AP, \varphi, \varphi_1, \varphi_2$ are both LTL formulas.The standard operators logic and ($\wedge$), logic or ($\vee$), non ($\neg$), and implication ($\rightarrow$) are propositional logic tokens. Operators X, U, F, G, and R are temporal operators, representing some properties of time [7].

- Gp: p must be true throughout the timeline
- Fp: p must be true at some point in the present or future
- Xp: p must be true at the next time point
- pUq: q is true at present, or q becomes true at some point in the future before p stops being true

Suppose we have a sensor that periodically measures a certain variable, and we want to check whether that variable fluctuates with a certain periodicity. We can use LTL to represent this periodic behavior. For example, the following LTL formula indicates that the variable should fluctuate every 5 time steps.

$$\mathrm{G(F(((Time\%5) == 0)\&\&(Measurement > Threshold)))}$$

Here, *Time* represents time steps, and *Measurement* is the measured value. This LTL formula ensures the periodic fluctuation property.

Büchi Automaton

Büchi Automaton is a commonly used formal tool for describing and verifying behavioral specifications of systems. In past research, many scholars have extensively studied and applied Büchi Automaton. Vardi and Wolper [8] proposed the theoretical framework of Büchi Automaton and proved its equivalence to Linear Temporal Logic (LTL). Their work laid the foundation for the use of Büchi Automaton in model checking and

runtime verification. Alur and Dill [9] introduced construction methods for deterministic Büchi Automaton and applied them to model checking. Their work has had a profound impact on automata theory and formal verification. Baier and Katoen [10] investigated the modeling of Büchi Automaton for stochastic systems. They explored how Büchi Automaton can be used to describe and verify systems with stochastic properties. Piterman et al. [11] introduced an enhanced version of Büchi Automaton called Aggressive Automaton. Their work aimed to improve the practicality and efficiency of Büchi Automaton in runtime verification. Joël D. Allred et al. [12] proposed a simple algorithm that complements Büchi automata by directly manipulating subsets of state sets (similar to slices) and generating deterministic automata. Büchi automaton is a tuple $B = (Q, Q_0, \Pi, \sigma, F)$, where

- Q is a finite nonempty set of states,
- $Q_0 \subseteq Q$ is a set of initial states,
- $\Pi : \Pi = 2^{AP}$ is the input alphabet (set of atomic propositions),
- $\sigma : \sigma \subseteq Q \times \Pi \times Q$ is nondeterministic transfer relationship,
- $F : F \subseteq Q$ is a set of accepting states.

Runtime verification

Runtime verification is a novel formal verification technique that relies less on the environment. Leucker and Schallhart [13] provide a concise introduction to runtime verification, discussing its foundations, techniques, and challenges. They highlight the role of runtime verification in complementing formal verification and testing methods. Like model checking, runtime verification employs a formal approach to describe the behavior constraints that software systems are expected to satisfy. Linear temporal logic (LTL) is a common formalism used for this purpose. In theory, runtime verification aims to monitor an infinite sequence w of the system and check whether it satisfies the constraint φ. If w satisfies φ, the result is true; otherwise, it is false.

Runtime verification can be divided into two applications based on the type of object being monitored by the monitor. The first application is real-time monitoring of the executing behavior sequence to verify whether the current behavior satisfies the property constraints. In case of a violation, an alert is triggered. This type of runtime verification is referred to as online verification. The second application involves monitoring the historical execution sequence and analyzing the stored execution paths offline. It is used in offline automatic evaluation testing and is known as offline verification [14]. For intelligent systems, it is crucial to detect behaviors that violate safety properties as early as possible, enabling proactive measures to be taken. Therefore, this paper focuses on the online verification approach.

Related work

A monitor is a tool or device used for supervising and managing a system or component. It operates concurrently with the system or component, continuously monitoring and recording their operations and behaviors in real-time [15]. Early monitoring methods primarily involved tracking and logging critical execution paths, function calls, and events during system runtime, along with sampling and data collection [16]. However, these methods are not well-suited for complex distributed systems. Therefore, Snodgrass [17] proposed a more efficient and scalable approach by collecting runtime data from the system and transforming it into instance data in relational schemas to monitor the system's state and behavior, extracting dynamic information from complex systems.

In addition, runtime verification has become an important topic in achieving accurate monitoring of complex temporal properties. In the context of runtime monitoring, a monitor can be defined more specifically as a device that reads finite execution traces generated by the system at runtime and provides corresponding conclusions or judgments based on these traces [13]. For the design and construction of monitors, Basin et al. [18] proposed a runtime monitoring algorithm for metric first-order temporal logic, overcoming the limitations in the expressiveness of previous algorithms regarding property specification languages. Dong et al. [19] introduced a closed-loop feedback prediction and prevention framework that can monitor system behavior in real-time and dynamically adjust and intervene based on predictive results. Pedro et al. [20] introduced a combined monitoring framework (CMF) that integrates runtime monitoring and static analysis to ensure time isolation and response time guarantees in real-time systems. By using metric temporal logic with durations (MTL-$\int$), they conducted schedulability analysis and static checks on CMF, generating monitors with explicit durations. This represents the first application of combining MTL-$\int$ with the generation of monitors with explicit durations for runtime verification in hard real-time systems. Vierhauser et al. [21] proposed a model-driven framework (GRuM) for generating customized runtime monitoring platforms that support data collection and analysis and can be extended and updated when the monitored system changes.

In recent years, there has been increasing emphasis on runtime monitoring of unmanned systems. Heffernan et al. [22] utilized real-time runtime verification monitors to monitor and verify functional safety requirements of automotive embedded control systems. DONG et al. [7] conducted a series of studies on security threat detection in the UAV domain. Vierhauser et al. [21, 23, 24] focus on achieving flexible and adaptive runtime monitoring in CPS, enhancing monitoring efficiency and adaptability through model-driven techniques and the automated generation of monitoring platforms. Ayhan Mehmed et al. [25] proposed the Safe Driving Envelope-Verification (SDE-V) method for runtime validation of automated driving system (ADS) path planning compliance with safety rules. The approach, which focuses on reducing false positives, assessing monitor quality, designing scalable monitoring architecture, and addressing sensor fusion challenges, introduces notable advances in ADS safety and reliability. Mathilde Machin et al. [26] proposed a versatile method for high-level safety monitor specification generation, incorporating hazard analysis to ensure correctness. The approach, adaptable to complex systems with multiple variables and interventions, lacks real-world application and necessitates further refinement for practical system dynamics and environmental adaptability.

In general, current research still faces limitations in dealing with complex environments, primarily reflected in insufficient exploration of how to accurately describe the environment and ensure the safety and correctness of system execution under uncertain conditions. Most studies focus on specific domains or specific scenarios, without considering highly variable and uncertain environments. The presence of uncertainty and randomness in system behavior increases the complexity of runtime monitoring and poses significant challenges.

Context-aware environmental modeling method

To address the issue of mismatched system data and the actual environment caused by environmental influences in intelligent systems, this paper proposes a combined approach that leverages context-aware information and runtime verification technology.

System framework

Runtime verification is an effective approach for system safety verification, as it enables the evaluation and validation of systems. However, current methods often overlook the influence of the environment, which makes it challenging to ensure the security and reliability of the system in the face of environmental factors. Therefore, this paper aims to address the following issues:

- How to model and quantitatively analyze the system's operating environment.
- How to accurately characterize and describe the task properties of the system.
- How to design an algorithm that combines two automata to generate a monitor.

To tackle these challenges, this study proposes the method framework shown in Fig. 1.

According to the diagram of the validation framework, the environmental information is abstracted in this paper. The environmental abstract model can represent the influence relationship between the environment and the system, and the environmental model is transformed into a finite automaton. This study describes the properties of the system using linear temporal logic and transforms them into Buchi automata. Then the algorithm is designed to combine the two automata by cross-multiplication and optimize the search path to generate the optimal monitor code. Finally, the monitor code is inserted into the executive program to realize real-time monitoring.

Environment modeling

The performance of an intelligent system may be influenced by the physical environment in which it operates. Factors such as temperature, humidity, and pressure can cause variations in system parameters, including input, output, and state, among others. These variations can impact the control system's decision-making process and result in suboptimal actions. To eliminate such effects and enable the system to adapt to environmental changes, it is necessary to define and model the relationship between the environment and the system.

This paper adopts a method of establishing a relationship model between the environment and the system. Firstly, it is necessary to clarify the relationship between environmental changes and the system, delineate the boundaries between environmental information and the system, and subsequently model the environmental factors accordingly. There are various types of environmental information, and to facilitate distinction, this paper refers to the definition of environmental information provided in the reference [27].

Definition 1 The environment information is defined as a quadruple $Enifo = (loc, tab, time, behavior)$, where loc, tab, time, and behavior represent location information, identification information, time feature information, and behavior information, respectively.

Environmental information serves as a crucial foundation for intelligent systems to carry out perception,

decision-making, and control actions. It enables the system to infer the current state and characteristics of the external environment, facilitating better adaptation and response to diverse situations. While the types and sources of environmental information may vary across different application scenarios, they share a common feature of providing real-time feedback on the external environment to intelligent systems, thereby aiding in making more accurate and rational decisions.

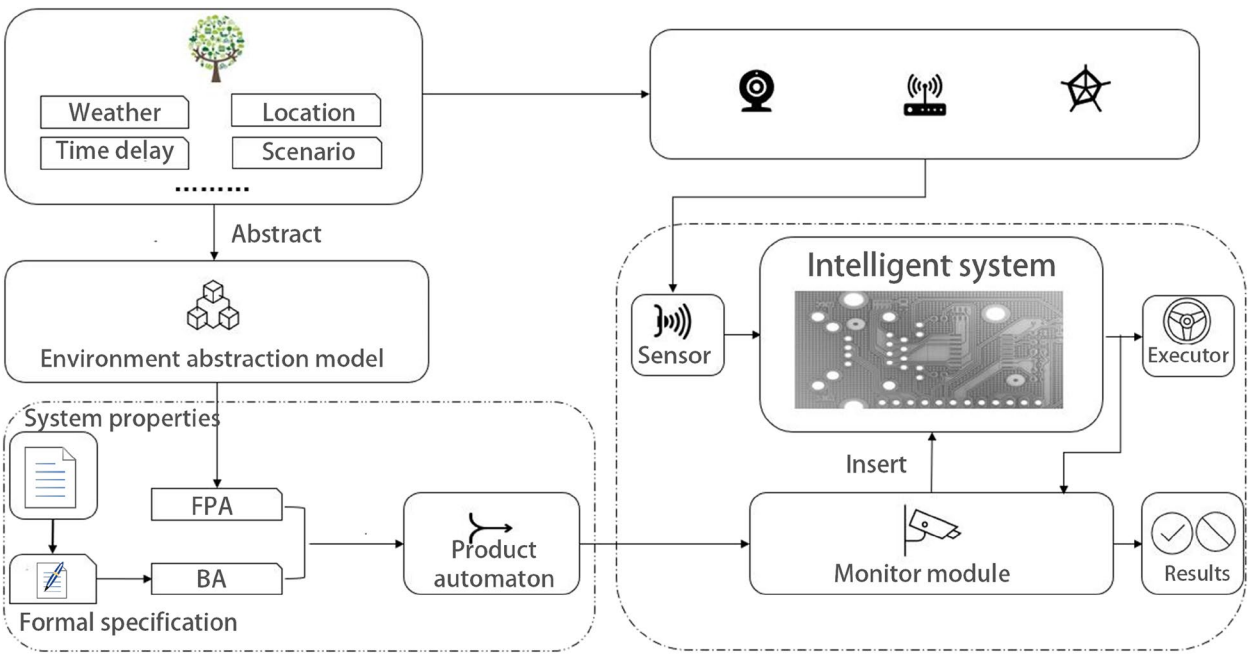


Fig. 1 Runtime verification framework considering environmental factors

To facilitate the identification of data as environmental information, Table 1 presents specific classification methods. These methods assist in determining whether the given data information qualifies as environmental information.

Definition 2 The formal description of the environment abstraction model is a tuple $M = (S_{env}, CE)$. Here, S_{env} represents the static structure formal description of the environment model, which establishes a mapping relationship with the system properties based on the classification information of the environment. CE is a probabilistic finite automaton (PFA) that conforms to the definition and is used to describe the dynamic behavior of the environment model. Each state of CE represents a state of the environment model.

Definition 3 The static structural formal representation of the environment model is given by a triple $S_{env} = (E, S, R)$. Here, $E = \{e_1, e_2, \ldots, e_n\}$ is the set of environment states, $S = \{s_1, s_2, \ldots, s_n\}$ is the set of affected system parameter states, and $R = \{r_1, r_2, \ldots, r_n\}$ is the set of mapping relations between E and S.

Definition 4 The dynamic structural formal representation of the environment model is given by a tuple $CE = \langle D, D_0, T, \Pi, \delta, \mu_0 \rangle$, where

- D : Set of states.
- D_0 : Initial state.
- T : State transition function $D \times T \rightarrow D$.
- Π : Input alphabet, where $\Pi = 2^{AP}$.
- δ : Labeled probabilistic transition function $Q \times \Pi \times Q \rightarrow [0, 1]$.
- μ_0 : Probability distribution over D_0.

Let's illustrate an example of environment modeling using the Adaptive Cruise Control (ACC) system. ACC is a cruise control system that allows the vehicle to travel at a specified speed while maintaining a reasonable and safe distance from the vehicle ahead. The braking distance required by the vehicle may vary depending on the road and weather conditions. The road and weather conditions are assessed using the Road and Weather Index, which is a weather-based index that provides valuable information on road conditions by analyzing real-time weather data. It is categorized into five levels, as shown in Table 2.

Table 1 Classification and description of environmental information

Category	Description
loc	Information about specific physical positions, such as location data from GPS sensors and the distances measured between two sensors.
tab	Information are identified by sensors, such as temperature and facial recognition.
time	Information about time-related features, such as request time and data read from a clock.
behavior	Information about actions obtained from sensors, such as action information and door opening and closing.

Table 2 Road condition meteorological index

Classification	Description	Recommendation
Level 1	Sunny: visibility greater than 10km, wind force less than level 4	Observe the city traffic speed limit
Level 2	Cloudy: small amount of standing water	Do not exceed 50km/hour
Level 3	Light rain or snow: obvious standing water	Do not exceed 40km/hour
Level 4	Moderate rain or snow: blurry vision, easy to slip	Do not exceed 30km/hour
Level 5	Heavy rain or snow: covered with water or snow, poor visibility	Do not exceed 20km/hour

According to the definition of the environmental model, $E = \{e_1, e_2, \ldots, e_n\}$ represents five states of road conditions, $S = \{s_1, s_2, \ldots, s_n\}$ represents the braking safety distance, and R=(r) represents the different braking safety distances caused by different road conditions. The functional relationship is expressed as $s = v^2/2^*u^*g$.

The finite probabilistic automaton for the environmental model can be represented as shown in Fig. 2. $D = \{d_1, d_2, \ldots, d_n\}$ represents the transitions between five different levels in the environment, where d_1 is the initial state. $T = \{t_1, t_2, \ldots, t_n\}$ represents different triggering transition events, and each transition is marked in the form of $p\backslash t_i$, where p represents the probability of transition between states.

Task formalization description

Traditional task formalization descriptions often use linear temporal logic to express goals, constraints, and

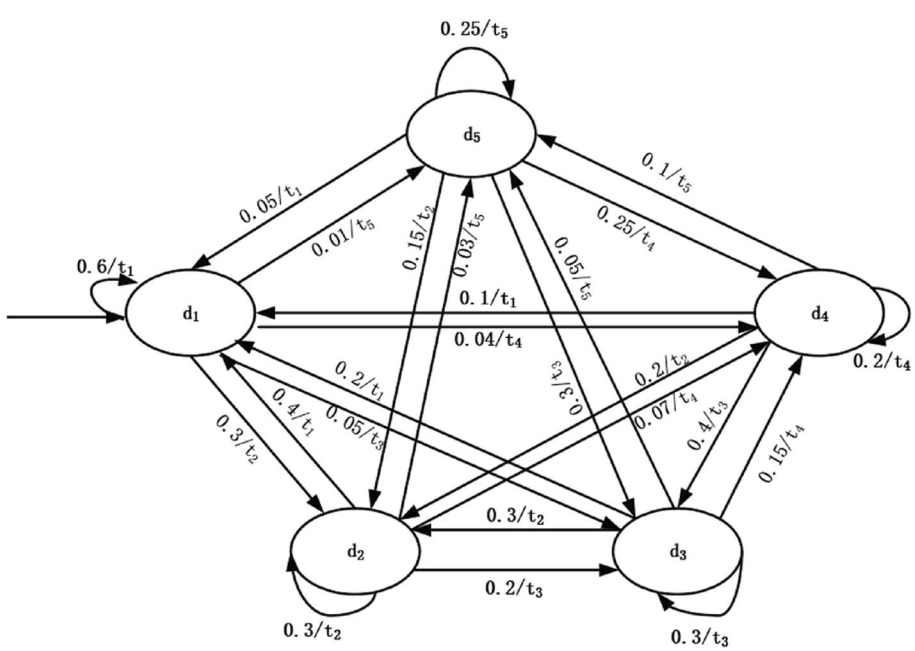


Fig. 2 Automata form of road weather

operability in mathematical or logical expressions. However, traditional binary logic may not provide results when dealing with infinite paths. When monitoring a system in a runtime application, only the system's running state and behavior can be obtained, and what will happen next is not yet determined. Therefore, monitoring the current state is necessary to evaluate whether the subsequent behavior satisfies safety properties. In this case, using ternary logic can more accurately express this description than binary logic. Therefore, extending linear time logic LTL with ternary semantics is necessary. For a property φ and its negation $\neg\varphi$, if the current prefix violates the property, the system does not satisfy the property φ and outputs false; otherwise, it outputs true. If the current prefix satisfies both φ and $\neg\varphi$, no correct judgment can be made, and the output is inconclusive. In summary, properties can be divided into three cases, illustrated with a finite path u and property φ:

- If property φ can be proven on the current observed finite path u, there is no need to consider future events, and the output is *true*.
- If property φ can be proven on the current observed finite path u that it will not hold on any subsequent path, the output is *false*.
- If it is impossible to prove whether property φ holds on the current observed finite path u, the output is *inconclusive*, and continuous monitoring is necessary in the future.

The following is the formal task description of Adaptive Cruise Control (*ACC*). During automatic cruise mode, the vehicle will accelerate to the predetermined cruising speed (*TLV*) if there is no vehicle ahead or if the distance between vehicles is greater than the safe distance. However, if there is a vehicle in front and the distance between vehicles is less than the safe distance, the vehicle will decelerate until it reaches an appropriate cruising speed (*VPL*). The scenario description is shown in Fig. 3. The property specifications can be expressed as follows:

$$G((!exist \vee (dis > ST)) \rightarrow F(|\, speed - TLV \,| < 0.1)) \vee$$
$$G((exist \wedge dis \leq ST) \rightarrow F(|\, speed - VPL \,| < 0.1))$$

The formulas $\varphi_1 = G((!exist \vee (dis > ST)) \rightarrow F(|\, speed - TLV \,| < 0.1))$ and $\varphi_2 = G((exist \wedge dis \leq ST) \rightarrow F(|\, speed - VPL \,| < 0.1))$ ensure that the vehicle's speed can be adaptively adjusted based on the distance and the vehicles ahead in the automatic cruise control mode. In this context, *exist* represents an obstacle ahead, *dis* represents the distance to the vehicle in front, *ST* represents the safe following distance, *TLV* represents the cruising speed, *VPL* represents the appropriate cruising speed, and $|\, speed - VPL \,| < 0.1$ represents increasing the vehicle speed to the cruising speed and maintaining it within a certain threshold range.

Monitor generation

In order to ensure the safety of the system under the influence of environmental factors, the environment model should be used together with the monitoring specification so that the system can meet the specification. Therefore, to generate a runtime monitor that meets the requirements, the defined environment model should be combined with the system specification, which has been formalized into an automaton, and then appropriately inserted at the desired locations. The monitor will be able to effectively monitor and control the system's operating state to ensure its safety and reliability. Figure 4 shows the process of improved monitor generation.

Algorithm for generating monitors

The article proposes a formal verification model that combines the environment model represented by *CE* and the system model represented by *B*, to enable unified verification of the system's response to environmental influences in a feedback loop. The concept of synchronous labeling and synchronous labeling function is introduced.

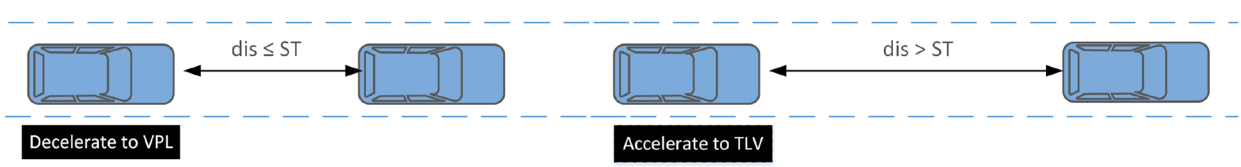


Fig. 3 Scenario Description

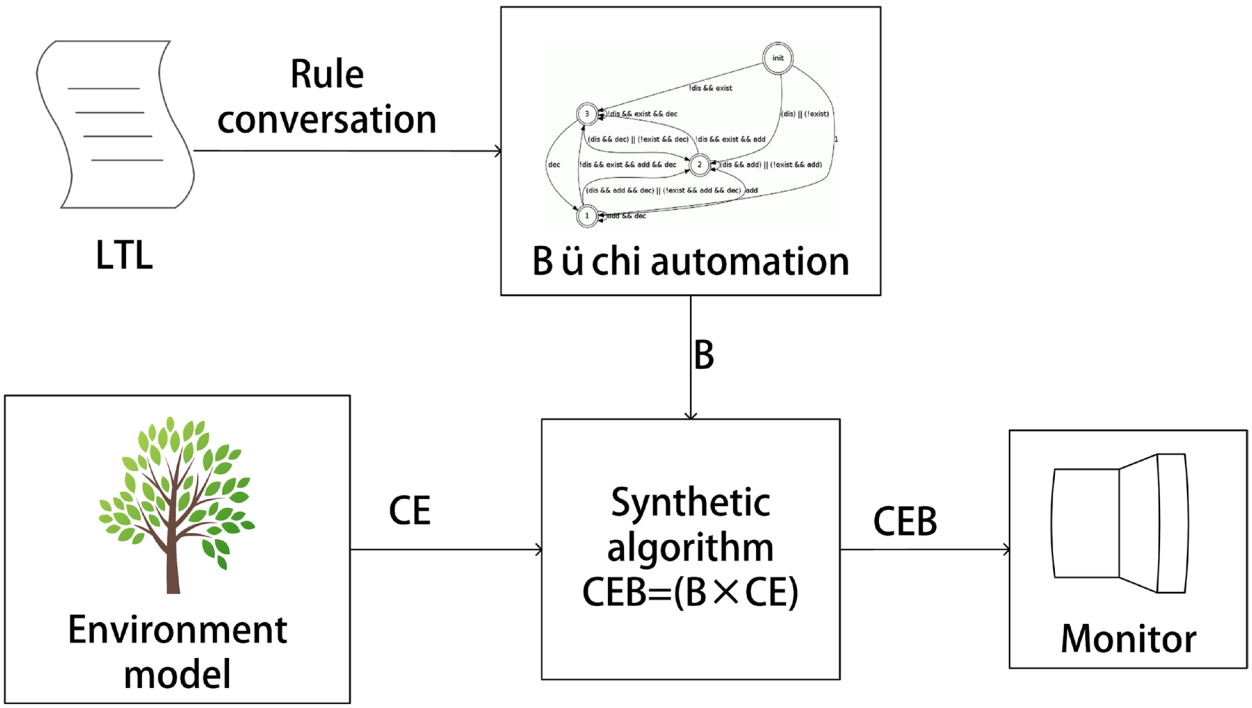


Fig. 4 Improved monitor generation process

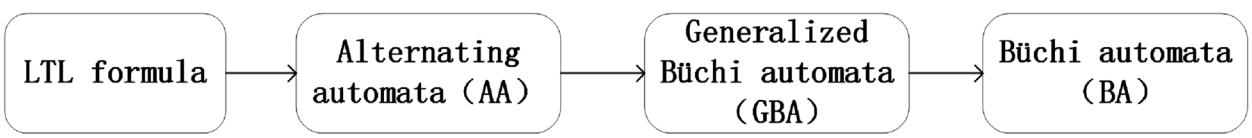


Fig. 5 LTL to Büchi automata process

Converting LTL to Büchi automaton can be divided into three steps as shown in Fig. 5. First, it is necessary to convert the LTL formula to an alternating automaton, which can represent all the semantics of the LTL formula. Secondly, by using the equivalence between generalized Büchi automata and alternating automata, the alternating automaton is converted to a generalized Büchi automaton. Finally, the generalized Büchi automaton needs to be converted to a Büchi automaton for more efficient runtime monitoring.

LTL to AA

Definition 5 Alternating Automaton(AA) is represented by the quintuple $A_\varphi = (S, \Sigma, \delta, I, F)$,where

- S is the set of states,
- Σ represents an alphabet,
- $I \subseteq P_f(S)$represents the initial state(s), and $P_f(events)$ represents events that can occur simultaneously,
- $F \subseteq S$ represents a set of accepting states,
- $\delta : S \to P_f\big(P_f(S \times \Sigma)\big)$ represents a transition function.

Before performing the transformation from the LTL formula to alternating automaton, it is necessary to define the subformula form ψ of property φ. Both φ and ψ should accept the alternating automaton A_φ, and their accepting language should have the initial state $I = \bar{\psi}$.

Given an LTL formula φ over a set of atomic propositions AP, $A_\varphi = (S, \Sigma, \delta, I, F)$ is transformed into an alternating automaton. After the transformation is completed, the accepting language of the alternating automaton A satisfies the LTL formula φ.

AA to GBA

To transform the alternating automaton $A_\varphi = (S, \Sigma, \delta, I, F)$ into a generalized Büchi automaton $B_G = (S, \Sigma, \delta', I, F)$, the key is that the accepting states are ordered. Each state in A_φ is split into two states: one state represents all even rounds, and the other represents all odd rounds. For each transition

in A_φ, two transitions in B_G are created: one goes from the even state of the initial state to the odd state of the target state, and the other goes from the odd state of the initial state to the even state of the target state. All terminating states in A_φ are marked as terminating states in B_G, regardless of parity. A special initial state is added to B_G, which includes the even and odd states of the initial state in A_φ. After completion, the automaton is made equivalent. To transform the alternating automaton $A_\varphi = (S, \Sigma, \delta, I, F)$ into a generalized Büchi automaton $B_G = ()$, the key is that the accepting states are ordered. For each state in A_φ, it is split into two states: one state represents all even rounds, and the other state represents all odd rounds. For each transition in A_φ, two transitions in B_G are created: one goes from the even state of the initial state to the odd state of the target state, and the other goes from the odd state of the initial state to the even state of the target state. All terminating states in A_φ are marked as terminating states in B_G, regardless of parity. A special initial state is added to B_G, which includes the even and odd states of the initial state in A_φ. After completion, the automaton is made equivalent. The generalized Büchi automaton transformed from the ACC task property specification φ given in Section Task formalization description is shown in Fig. 6.

GBA to BA

To construct the monitor, using the Büchi automaton B is the most convenient way. B is easier to implement and optimize compared to B_G, which can simplify the structure of the automaton and improve readability and maintainability. The method of converting B_G into B has been widely researched and implemented and can be referred to the theoretical research to provide the conversion approach [28].

The state transition function δ' is represented as

$$\delta'(q,j) = \left\{ \left(\alpha, \left(q', j'\right)\right) \mid \left(\alpha, q'\right) \in \delta(q), j' = \text{next}\left(j, \left(q, \alpha, q'\right)\right) \right\}$$

and the *next* function is defined as follows:

$$\text{next}(j,f) = \begin{cases} \max\left\{j \le i \le r \mid \forall j < k \le i, f \in F_k\right\} j \ne r \\ \max\left\{0 \le i \le j \mid \forall 0 < k \le i, f \in F_k\right\} j = r \end{cases}$$

After the transformation process described above, the non-deterministic generalized Büchi automaton B_G can be generalized to the Büchi automaton B. The LTL formula can be transformed into a Büchi automaton $B = (Q, Q_0, \Pi, \sigma, F)$ to determine whether the input sequence satisfies the task requirements. The Büchi automaton obtained from the ACC task property specification φ, as given in Section Task formalization description, is shown in Fig. 7.

The integrated algorithm for synthesizing the environment model and Büchi automaton

One method for combining two automata is through synchronous composition. Synchronous composition involves the merging of two automata in such a way that the resulting automaton can only be in an accepting state when both original automata are simultaneously in an accepting state. This process can be achieved through the following steps:

Firstly, the state sets of the two automata are used to compute the Cartesian product, yielding a new set of states. Then, a new transition function is defined based on the transition functions of the original automata. This new function ensures that the composite automaton only transitions when both automata satisfy the transition conditions simultaneously.

In addition, the initial and accepting states are defined by combining the initial states of the two original automata. Accepting states are determined by states where both automata are accepting simultaneously.

By following this systematic process, two valid-state automata can be effectively combined to create a new automaton. The resulting composite automaton seamlessly integrates the specifications of both original

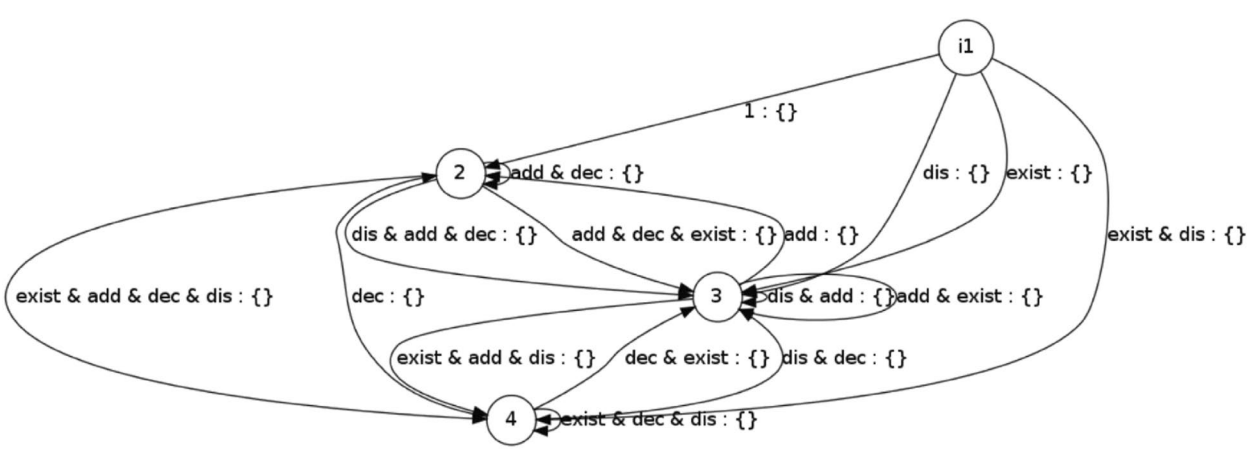


Fig. 6 Transformed Generalized Büchi Automata

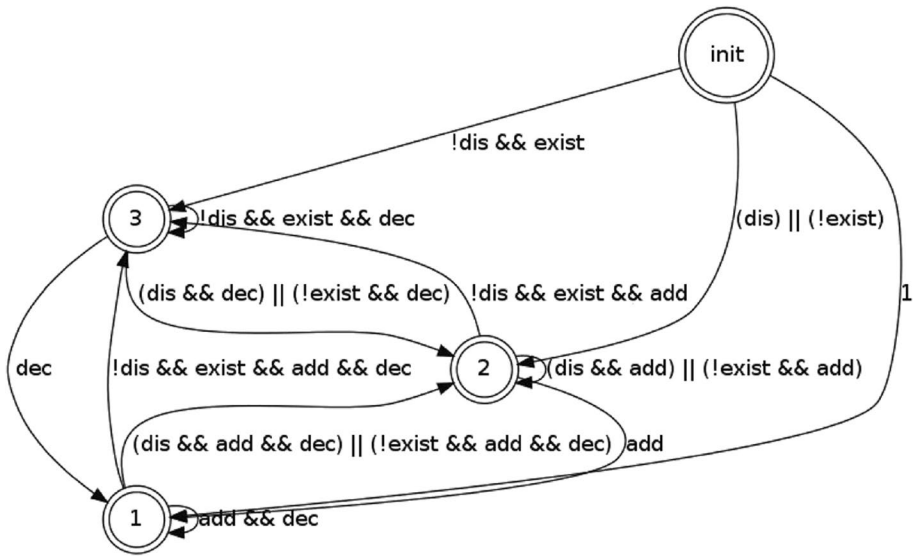


Fig. 7 Transformed Büchi automata

automata, requiring both automata to be in sync for the composite automaton to reach an accepting state.

Building upon this theoretical foundation, this section presents a comprehensive algorithm that combines the environment model *CE* and the Büchi automaton *B*, allowing verification within a feedback loop. To construct the comprehensive algorithm, the first step is to define synchronous labels.

Definition 6 The synchronous label ε of the environment model *CE* is defined as a set of transition condition events on the transitions of the Büchi automaton *B*.

Synchronous labels serve as interaction indicators between the environment model and the Büchi automaton, determining whether a state transition can occur at a specific time point. By defining synchronous labels, it becomes possible to combine the environment model and the Büchi automaton into a new representation. In this new model, the combination of states from the environment model and the Büchi automaton forms the states, and the synchronous labels define the transition conditions between states. Additionally, formal verification techniques can be applied to validate the interaction behavior between the environment model and the Büchi automaton and ensure compliance with specific properties.

Below is the definition of the combination of the environment model $CE = \langle D, D_0, T, \Pi, \delta, \mu_0 \rangle$ and the Büchi automaton $B = (Q, Q_0, \Pi, \sigma, F)$.

Definition 7 The product of the environment model CE and the Büchi automaton B denoted as CEB can be represented by a tuple as follows:

$$CEB = CE \otimes B = \left(Q^{\xi}, Q_0^{\xi}, \Pi, \delta, \sigma, F, \mu_0\right)$$

- Q^{ξ} is the set of states,
- $Q_0^{\xi} : Q_0^{\xi} = (Q_0, D_0) \in Q^{\xi}$ is the set of initial states,
- $\Pi : \Pi = 2^{AP}$ is the input alphabet (set of atomic propositions),
- $\delta : Q \times \Pi \times Q \rightarrow [0, 1]$is the labeled probabilistic transition function
- $\sigma = \sigma \times D \times T$ represents the transition relationship,
- F is the set of accepting states.
- μ_0 is the probability distribution over Q_0^{ξ}

Combining the environment model *CE* and Büchi automaton *B* results in *CEB*, which contains both the information of the operating environment and the property constraints of the LTL formula.

The combination of the environment model *CE* and Büchi automaton *B* is not strictly a Cartesian product because a simple Cartesian product may lead to invalid transitions. The synthesis algorithm described in Algorithm 1 is designed for a given environment model *CE* and Büchi automaton *B*. The algorithm initializes the current state node and the previous state node and then iterates over all nodes in the state set. The transition relationship between the current node and the previous state node is considered. If it does not satisfy the synchronization mapping, it is regarded as unreachable, and the automaton form and transition are modified. Otherwise, a new transition form is added.

Data: $CE = \langle D, D_0, T, \Pi, \delta, \mu_0 \rangle$, $B = (Q, Q_0, \Pi, \sigma, F)$, synchronous label ε
Result: $CEB = \left(Q^{\xi}, Q_0^{\xi}, \Pi, \delta, \sigma, F, \mu_0\right)$
$Init\,(Q_{\text{pre}} = Q_0, Q_{cur})$;
for Q_{cur} *to* Q **do**
  if $Visit(Q_{cur})$ **then**
    $ans = (Q_{\text{pre}}, Q_{cur}, \sigma, D, \varepsilon)$;
  end
  else if $!ans$ **then**
    /* relax model constraints entirely */
    $\sigma = \text{RelaxAll}\,(Q_{\text{pre}}, Q_{\text{cur}}, \sigma, D, \varepsilon)$;
  end
  else
    /* adapt model constraints entirely */
    $\sigma = \text{Adapt}\,(Q_{\text{pre}}, Q_{cur}, \sigma, D, \varepsilon)$;
  end
  $Update\,(Q_{\text{pre}}, Q_{\text{cur}})$;
end

Algorithm 1 Synthetic algorithm

Insertion process

In the context of formal system verification, the insertion of monitors is an important step. Inserting monitors allows for dynamic monitoring and verification of the system by inserting a runtime monitor into specific transitions of the system. The process of insertion is typically achieved by defining insertion points and insertion rules. The insertion points specify where the monitor should be inserted, while the insertion rules specify how the monitor is inserted at those points.

In the previous section, a monitor was constructed using the synthesis algorithm that combines the environmental model *CE* and the Büchi automaton *B*. To monitor the entire system, it is necessary to insert the monitor at appropriate locations. However, a generic insertion method that inserts the monitor after every executed statement would result in significant redundancy and overhead. In reality, many program statements, such as parameter initialization and variable assignments, do not affect the system's behavior. Therefore, it is crucial to simplify the insertion algorithm. To reduce the complexity of verification, this paper introduces the concept of visible variables *VisibleVar*.

Definition 8 For a system being monitored, let's refer to it as System. If the current statement does not cause any changes in the states of the monitor ($L(system) \cup L(CEB) = \varnothing$), the current execution point is considered invisible to the monitor. However, if the current code has the potential to alter the state of the monitor, it is considered visible to the monitor.

$$\text{VisibleVar} = \begin{cases} \text{false} & \text{When and only when } L(\text{system}) \cap L(CEB) = \varnothing \\ \text{true} & \text{otherwise} \end{cases}$$

$L(system) \cup L(CEB) = \varnothing$ represents that when the value at the current execution point of the system changes, it does not affect the values in the monitor. By using visible variables to identify the visible set of the program, it clearly defines which data and which parts of the program are necessary. The monitor can only access them in specific contexts, thereby avoiding erroneous data access, reducing the time required for evaluations, and improving efficiency and safety.

Through optimization, the goal is to assess the impact of runtime verification on system performance while maintaining accuracy and completeness in the verification process. Removing unnecessary insertion points and only instrumenting at critical locations makes it possible to significantly reduce the size of the state space and improve the system's runtime speed. Specifically, the instrumentation process is depicted in Fig. 8. The insertion points specify where the monitor is inserted, allowing instrumentation only in the corresponding intersecting portions, which greatly reduces the scale of instrumentation.

Instrumenting monitors make it possible to dynamically monitor and verify the system, thus enhancing its reliability and security. Moreover, inserting the monitors

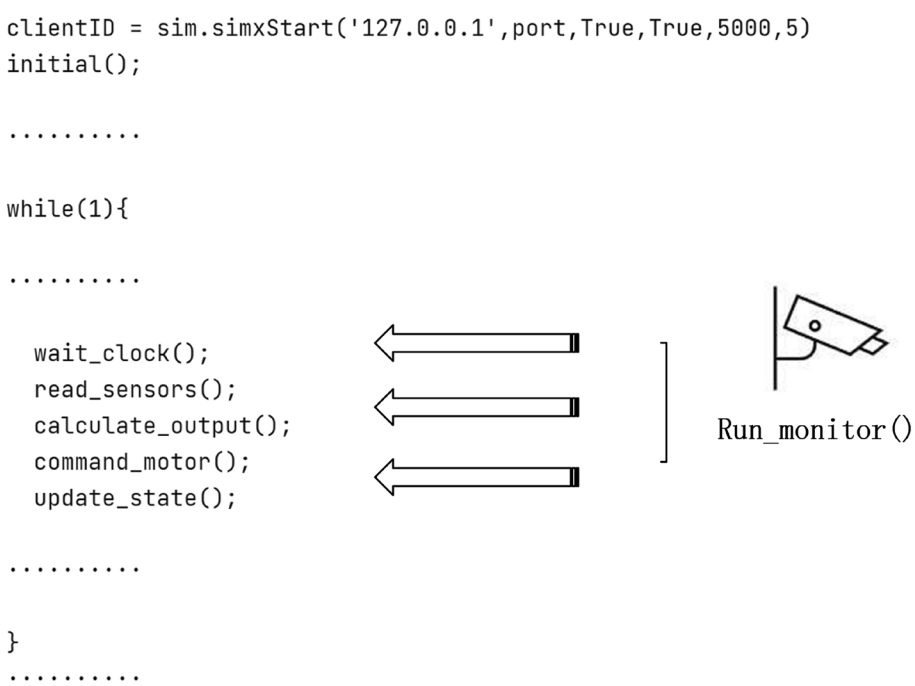


Fig. 8 Insertion form

only at appropriate locations within the system and evaluating properties based on violation cases means there is no need to insert the monitor program after every statement. This approach avoids unnecessary computations, significantly reducing the scale of verification and better addressing the complexity and practical requirements of the system.

Evaluation

To validate the effectiveness of the proposed method, we conducted experimental verification of simulation scenarios using the CoppeliaSim simulation software. Furthermore, we performed validation studies in real-world scenarios using the physical robot RoboMasterEP. By modeling and monitoring environmental factors, we demonstrated the feasibility and practicality of our proposed method.

Simulation experiment

CoppeliaSim is a powerful general-purpose robot simulation platform used in various fields such as robotics, mechanics, physics, and electronics. It provides a range of APIs and scripting interfaces, including Python, Lua, and C/C++, allowing developers to easily write their control programs or algorithms and interact with the simulation environment.

CoppeliaSim provides the ability to control each node in the vehicle (e.g., engine, shaft) by assigning names to them. By modifying the relevant kinematic parameters and calling the corresponding nodes, the vehicle can be controlled for motion. The kinematic model of the vehicle is shown in Fig. 9. The map coordinates of the vehicle's location can be directly obtained from CoppeliaSim and represented as (x, y). The coordinates of the detected object are represented as (x_i, y_i). In the model, d represents the distance between the center point of the vehicle and the object, α represents the angle between the forward direction and the object, β represents the angle between the object direction and the horizontal axis, and γ represents the angle between the forward direction and the horizontal axis.

To create an accurate simulation environment in CoppeliaSim, it is necessary to set the parameters of the entities based on real-world values. Therefore, in the experiment, the simulation vehicle utilizes the intelligent car model. To improve the accuracy of the simulation, infrared sensors, and vision sensors are added to the vehicle during the experiment to simulate environmental perception in the real world. These measures enable better simulation of the vehicle's movement and facilitate various tests and experiments. The simulation scene is depicted in Fig. 10.

The controller designed in this section is implemented using the Python programming language. By modifying the wheel dynamics parameters of the vehicle in the simulation environment through Python code, the vehicle's motion can be controlled. The interaction between modules is illustrated in Fig. 11. After obtaining sensor data,

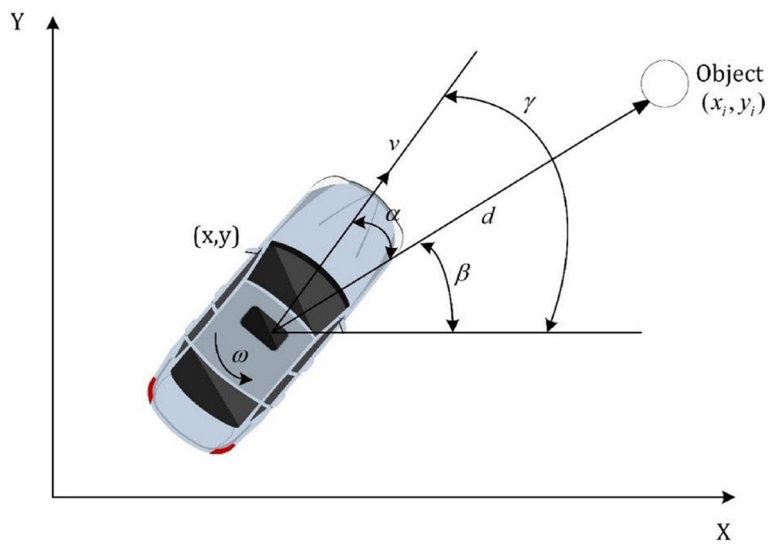


Fig. 9 Vehicle dynamics parameters

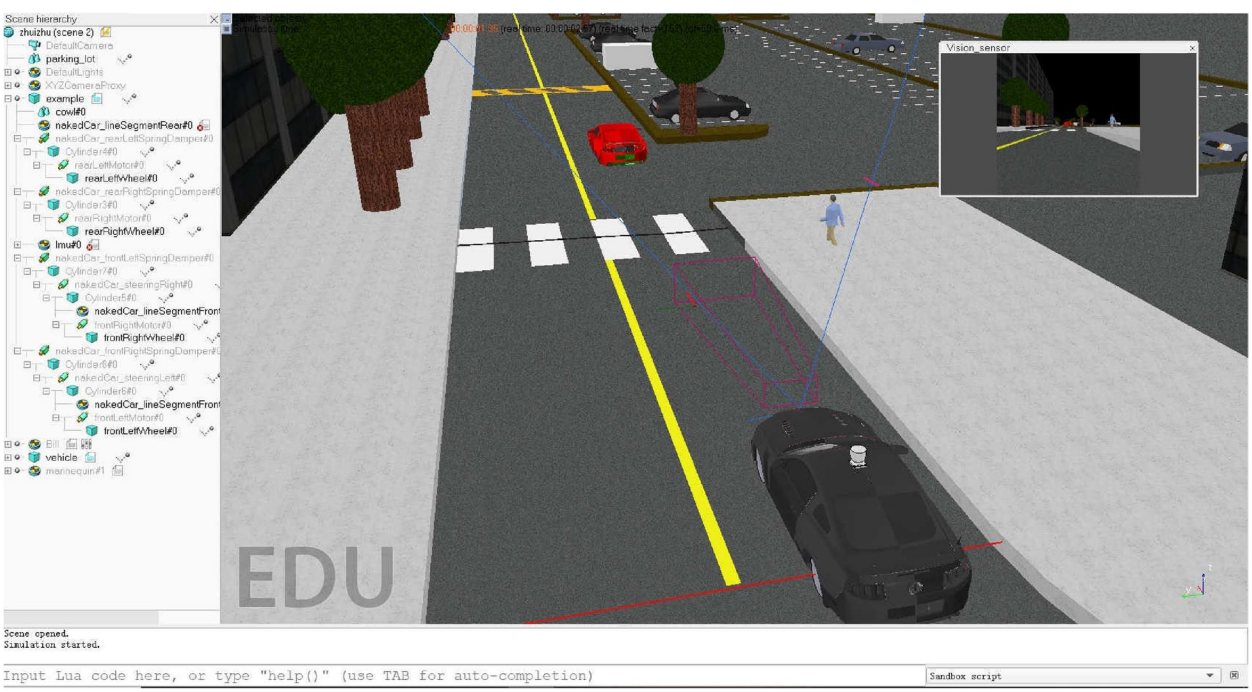


Fig. 10 CoppeliaSim Simulation scenario

it is processed, and control commands are generated to make decision control for the simulated vehicle.

The experimental design involves the operation of a moving vehicle in an automatic cruise mode, where each execution utilizes vision sensors and infrared sensors to perceive the surrounding environment. The designed safety property states that if there are no vehicles in front of the vehicle, or if there is a vehicle ahead and the distance is greater than the safety distance, the vehicle accelerates until it reaches the maximum cruise speed *TLV*. If there is a vehicle ahead and the distance is less than the safety distance, the ACC system controls the vehicle to decelerate until it reaches the safe cruising speed *VPL*.

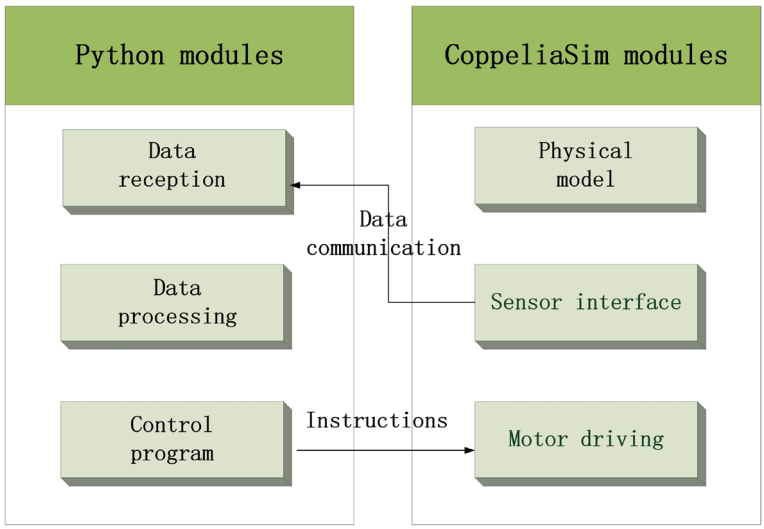


Fig. 11 Interaction between calculation and control

$$G((!\text{exist} \vee (\text{dis} > ST)) \rightarrow F(|\text{ speed} - TLV| < 0.1)) \vee$$
$$G((\text{exist} \wedge \text{dis} \leq ST) \rightarrow F(|\text{ speed} - VPL| < 0.1))$$

Analyze the influence of different environmental factors on the safety braking distance in the safety property. Establish an environmental model with road and weather conditions that affect the safety braking distance and integrate it into the system model. Table 3 shows an experiment comparing the simulation results with and without the monitor and different environmental factors. The symbols ✓ and × in the table indicate whether the monitor was added. It can be concluded from the results that the addition of the monitor can effectively detect system violations and provide timely feedback and responses. The reasons for the occurrence of *lessthansafedistance* and *collision* in the table are that the simulated vehicles did not respond differently to different environments when detecting vehicles in front, resulting in the same strategy being applied, which led to inadequate braking distance.

To further analyze the reasons for the different simulation results, this paper will further elaborate on the simulation scenario situation based on Fig. 12, and obtain detailed data through the *simxGetObjectPosition* interface.

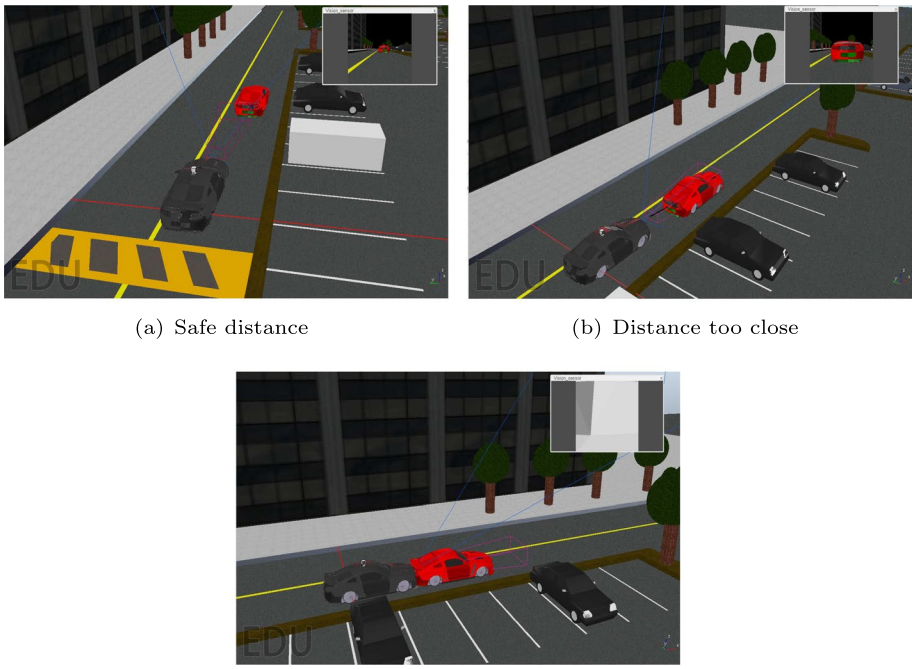
Table 3 Monitor effectiveness analysis

Environmental parameters	Monitor	Simulation result
Index 1, Friction Coefficient 0.65	✓	Maintain Distance
Index 1, Friction Coefficient 0.65	×	Maintain Distance
Index 3, Friction Coefficient 0.41	✓	Maintain Distance
Index 3, Friction Coefficient 0.41	×	Less than Safety Distance
Index 5, Friction Coefficient 0.3	✓	Maintain Distance
Index 5, Friction Coefficient 0.3	×	Collision Occurs

Figure 12a shows a schematic diagram in which the black car can maintain a safe distance from the red car after adding the monitor. The minimum distance between the red car and the black car is 19.831m, which always meets the minimum interval for a safe distance. Figure 12b shows that due to changes in environmental parameters, the black car brakes just in time when it reaches a close distance from the red car. At this time, the distance is already 10.386m, which is not enough to ensure a safe braking distance. Figure 12c shows a collision situation where the black car fails to adapt to changes in environmental parameters, leading to a collision with the front car due to the delayed braking.

As the monitor is inserted as a program into the system, this will cause more consumption during system operation. If the performance and efficiency of the monitor itself are not high, it will lead to inadequate monitoring or inaccurate monitoring, and thus cannot provide useful monitoring data and warning information, failing to effectively ensure the execution efficiency of the system.

As shown in Fig. 13, the time consumption issue caused by adding the monitor is demonstrated. The experiment was designed to run a fixed path length while obtaining the system's relative time ticks and interpolating and smoothing the discrete points. It can be seen that as the number of monitors increases, the required time consumption will increase exponentially, and optimization of the monitor can effectively reduce the time consumption. Initially, adding an unoptimized monitor code



(a) Safe distance (b) Distance too close

(c) Collision

Fig. 12 Interaction between calculation and control

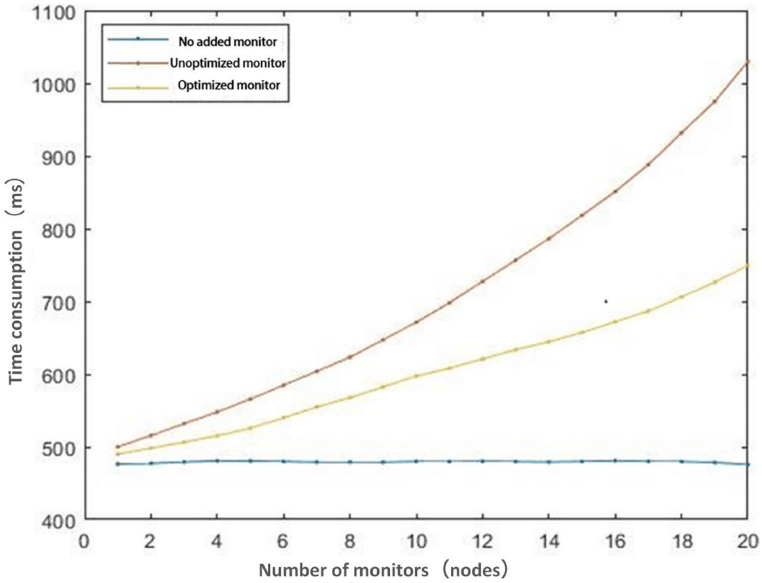


Fig. 13 Time consumption of monitor

requires about 23.6ms of time consumption, while adding 20 monitors requires 554.1ms. As the number of monitors increases, the time consumption does not increase linearly with a constant but shows an exponential trend. This is because when the number of monitors increases, the computational complexity of each monitor's attribute verification will also increase, and storing data related to system behavior will increase memory requirements and reduce performance. After optimization, adding an optimized monitor takes about 13.9ms, and the average time consumption of an optimized monitor is about 41.1% of that before optimization.

Experimental analysis shows that after modeling the environment, the monitor formed by combining the environment model and system specification can still effectively monitor the system and make corrections in complex environments. This ensures the safety and reliability of the system when facing environmental influences.

Physical experiments

In this paper, the RoboMaster EP launched by DJI Innovation Technology Co., Ltd. is adopted as the physical simulation platform. As shown in Fig. 14, The EP robot is equipped with open interfaces, sensor interfaces, and programmable components. Its control system is based on the Linux operating system, using ROS as middleware, providing a complete platform for robot control and programming. The experiment utilizes a platform communication mode, establishing a TCP/IP connection with the EP robot and enabling communication through a plaintext SDK. The running code is transmitted to the intelligent central controller. The monitoring algorithm proposed in this study is implemented in Python for monitoring through code instrumentation. All experiments were conducted on a machine with an Intel Xeon W-2225 CPU and 16GB RTX5000.

Figure 15 depicts the physical experimental setup. Due to the susceptibility of the robot's operational state to environmental factors, it is crucial to monitor the robot's behavior to promptly detect and address any issues that may arise. The experiment aims to assess whether the robot can maintain stability and make correct decisions when subjected to environmental changes. Therefore, the formulated property is $G(V < speed)$, where V represents the robot's velocity and *speed* denotes the desired speed threshold. This property ensures that the robot's velocity remains consistently below the specified speed threshold, indicating that it is maintaining a stable and controlled motion. In this experiment, the environmental factors in the scene are set to random values representing different conditions such as slippery, normal, and sandy. Each weather condition is assigned acceptance values (f_1,f_2,f_3) such that $f_1 + f_2 + f_3 = 1$. A section of slippery terrain is introduced along the path of the EP robot to simulate the effects of environmental interference. Based on the contextual information, a monitor is constructed and inserted into the executing code.

Fig. 15 Experimental scene

Fig. 14 EP robot

Figure 16 illustrates the variations in the robot's driving speed under the influence of environmental factors, considering the presence or absence of the monitor constraint. Since the vehicle's speed can be sampled periodically, the global speed changes can be observed intuitively. To capture the speed changes within shorter time intervals, the experiment adopts a fixed step size for sampling. The arrows in the figure represent moments when the environmental information changes, impacting the robot's driving speed. From the graph, it can be observed that the EP robot initially travels at a limited speed. However, upon entering the slippery terrain, the robot's actual speed increases beyond the safe speed threshold. It is only after passing through the slippery terrain that the robot returns to a safe speed. With the addition of the monitor, safety property violations are promptly detected, allowing corrective actions to be taken regarding the behavior of the EP robot.

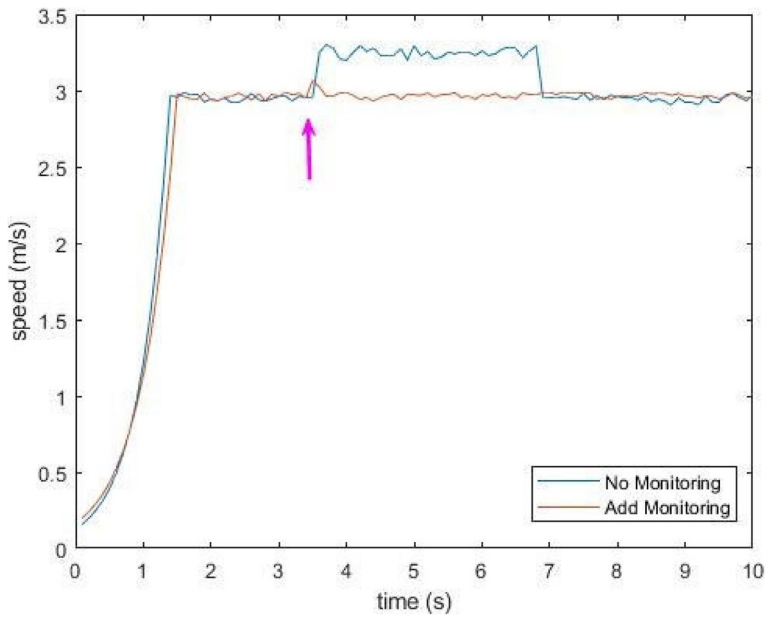


Fig. 16 Experiment result

Conclusion

This paper presents a kind of runtime monitor with environment awareness for ITS. By modelling and quantitative analysis of the environment, combined with runtime verification techniques, we have successfully inserted precisely expressed monitors into the target program. The construction of these monitors is based on the conversion process from LTL to Büchi automata, using an algorithm that combines the environment model with the Büchi automata. We also simplify the insertion algorithm by defining the visible variable *VisibleVar*. Simulation experiments in the CoppeliaSim environment and physical experiments with the actual EV robot verify the effectiveness of the proposed monitor. The experimental results show that the monitor can accurately monitor and control the system behaviour in real time and ensure the correct operation of the system under uncertain environments. This provides an effective solution for the safety and reliability of unmanned systems.

However, this paper employs LTL for descriptive purposes; nevertheless, limitations exist in characterising certain properties. Further research should aim to broaden the attributes' description to accommodate multifarious variations in intricate environments.

In conclusion, the context-aware environment online monitoring method proposed in this paper provides an effective solution for the safety and reliability of ITS. The real-time monitoring of system behaviour can ensure the normal operation of the system under uncertain environments and improve the performance and efficiency of the system. Future research will further advance the development and application of monitors to address more complex and diverse environmental challenges and promote the development of unmanned systems.

Authors' contributions

Yu Zhang ,Sijie Xu and Hongyi Chen wrote the main manuscript text. Uzair Aslam Bhatt and Mengxing Huang mainly responsible for revising and checking articles. All authors reviewed the manuscript.

Funding

This work was supported in part by the National Natural Science Foundation of China (Grant #: 62062030), in part by the Key R &D Project of Hainan province (Grant #: ZDYF2021SHFZ243), in part by the Major Science and Technology Project of Haikou (Grant #: 2020-009).

Availability of data and materials

No data were used to support this study. Data availability is not applicable to this article as no new data were created or analyzed in this study.

Declarations

Ethics approval and consent to participate

This article does not contain any studies with human participants or animals performed by any of the authors.

Consent for publication

The authors read and approved the fnal manuscript.

Competing interests

The authors declare no competing interests.

Received: 27 October 2023 Accepted: 7 December 2023
Published online: 03 January 2024

References

1. Cheng M, Li D, Zhou N, Tang H, Wang G, Li S, Bhatti UA, Khan MK (2023) Vision-motion codesign for low-level trajectory generation in visual servoing systems. IEEE Trans Instrum Meas 72:1–14
2. Bhatti UA, Huang M, Neira-Molina H, Marjan S, Baryalai M, Tang H, Wu G, Bazai SU (2023) Mffcg-multi feature fusion for hyperspectral image classification using graph attention network. Expert Syst Appl 229:120496
3. Liu K, Li P, Zhang Y, Ren J, Wang X, Bhatti UA (2023) Self-awakened particle swarm optimization bn structure learning algorithm based on search space constraint. Comput Mater Continua 76(3):3257–3274
4. Bhatti UA, Tang H, Wu G, Marjan S, Hussain A (2023) Deep learning with graph convolutional networks: An overview and latest applications in computational intelligence. Int J Intell Syst 2023:1–28
5. Bhatti UA, Marjan S, Wahid A, Syam M, Huang M, Tang H, Hasnain A (2023) The effects of socioeconomic factors on particulate matter concentration in china's: new evidence from spatial econometric model. J Clean Prod 417:137969
6. Pnueli A (1977) The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science (sfcs 1977). pp 46–57. https://doi.org/10.1109/SFCS.1977.32
7. Yang D, Shi H, Dong W, Liu ZL, Zhou G (2018) Security and safety threat detection method for unmanned aerial system based on runtime verification. J Softw 29(5):1360–1378. http://www.jos.org.cn/1000-9825/5508.htm
8. Vardi MY, Wolper P (1986) An automata-theoretic approach to automatic program verification. In: Proceedings of the First Symposium on Logic in Computer Science. IEEE Computer Society, New York, p 322–331
9. Alur R, Dill DL (1994) A theory of timed automata. Theor Comput Sci 126(2):183–235
10. Baier C, Katoen JP (2008) Principles of Model Checking. MIT press, Cambridge
11. Bloem R, Jobstmann B, Piterman N, Pnueli A, Sa'Ar Y (2012) Synthesis of reactive(1) designs. J Comput Syst Sci 78(3):911–938
12. Allred JD, Ultes-Nitsche U (2018) A simple and optimal complementation algorithm for büchi automata (LICS '18). Association for Computing Machinery, New York, pp 46–55
13. Leucker M, Schallhart C (2009) A brief account of runtime verification. J Logic Algebraic Program 78(5):293–303
14. Wang Z (2014) Research on runtime verification of real-time systems. Master's thesis, Huazhong Normal University, Wuhan, in Chinese with English abstract
15. (1990) IEEE Standard Glossary of Software Engineering Terminology. In: IEEE Std 61012-1990. pp 1–84. https://doi.org/10.1109/IEEESTD.1990.101064
16. Bayat B, Crasta N, Crespi A, Pascoal AMS, Ijspeert AJ (2017) Environmental monitoring using autonomous vehicles: a survey of recent searching techniques. Curr Opin Biotechnol 45:76–84. https://api.semanticscholar.org/CorpusID:4312879
17. Snodgrass R (1988) A relational approach to monitoring complex systems. ACM Trans Comput Syst 6(2):157–195. https://doi.org/10.1145/42186.42323
18. Basin D, Klaedtke F, Müller S, Zälinescu E (2015) Monitoring metric first-order temporal properties. J ACM 62(2). https://doi.org/10.1145/2699444
19. Zhao C, Dong W, Qi Z (2010) Active monitoring for control systems under anticipatory semantics. In: 2010 10th International Conference on Quality Software. pp 318–325. https://doi.org/10.1109/QSIC.2010.82
20. Matos Pedro A, Pereira D, Pinho LM, Pinto JS (2014) A compositional monitoring framework for hard real-time systems. In: Proceedings of the 6th International Symposium on NASA Formal Methods - Volume 8430. Springer-Verlag, Berlin, pp 16–30. https://doi.org/10.1007/978-3-319-06200-6_2
21. Vierhauser M, Garmendia A, Stadler M, Wimmer M, Cleland-Huang J (2023) Grum - a flexible model-driven runtime monitoring framework and its application to automated aerial and ground vehicles. J Syst Softw 203:111733. https://doi.org/10.1016/j.jss.2023.111733
22. Heffernan D (2014) Runtime verification monitoring for automotive embedded systems using the iso 26262 functional safety standard as a guide for the definition of the monitored properties. IET Softw 8:193–203(10). https://digital-library.theiet.org/content/journals/10.1049/iet-sen.2013.0236
23. Vierhauser M, Wohlrab R, Stadler M, Cleland-Huang J (2023) Amon: A domain-specific language and framework for adaptive monitoring of cyber-physical systems. J Syst Softw 195(C). https://doi.org/10.1016/j.jss.2022.111507
24. Stadler M, Vierhauser M, Garmendia A, Wimmer M, Cleland-Huang J (2022) Flexible model-driven runtime monitoring support for cyber-physical systems. In: Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings (ICSE '22). Association for Computing Machinery, New York, pp 350–351. https://doi.org/10.1145/3510454.3528647
25. Mehmed A (2020) Runtime monitoring for safe automated driving systems. PhD thesis, Mälardalen University
26. Machin M, Dufossé F, Blanquart JP, Guiochet J, Powell D, Waeselynck H (2014) Specifying safety monitors for autonomous systems using model-checking. In: Proceedings of the 33rd International Conference on Computer Safety, Reliability, and Security - Volume 8666 (SAFECOMP 2014). Springer-Verlag, Berlin, pp 262-277. https://doi.org/10.1007/978-3-319-10506-2_18
27. Luo C, Wang R, Guan Y, Li X, Shi Z, Xiaoyu S (2019) Integrated modeling method of cps for real-time data. J Softw 30(7):1966–1979. http://www.jos.org.cn/1000-9825/5753.htm
28. Gastin P, Oddoux D (2001) Fast ltl to büchi automata translation. In: Berry G, Comon H, Finkel A (eds) Proc. of the 13th Int'l Conf. on Computer Aided Verification, LNCS, vol 2102. Springer-Verlag, Heidelberg, pp 53–65

Publisher's Note