Formal Verification of Dynamical Control Systems (Addressing Integral Windup Phenomena Using Model-Checking)

MOHAMMED TLOUL, MICHAEL H. SCHWARZ, JOSEF BÖRCSÖK
Dept. Computer Architecture and System Programming
University of Kassel
Wilhelmshöher Allee 71, 34121 Kassel
GERMANY

Abstract: This paper investigates the utilization of model-checking as a potent method for verifying system designs, emphasizing its early error detection capabilities, reducing failures, increasing safety, and saving costs. The study explores the application of the UPPAAL tool and model-checking techniques within control systems. A case study in the paper concentrates on formally verifying Proportional Integral Derivative (PID) controllers, emphasizing integral windup issues. A model is constructed in UPPAAL for a control system that includes the system dynamics and the actuator limitations. The model's accuracy is validated against the MATLAB/Simulink® model. Formal requirements addressing integral windup are formulated, and a practical model-checking example using UPPAAL illustrates its utility in control system verification.

Key-Words: Model-based Verification, Formal Verification, Model-checking, UPPAAL, Safety-critical Systems, Control Systems Verification, Integral Windup, PID Controller.

Received: January 12, 2023. Revised: September 14, 2023. Accepted: October 17, 2023. Published: November 15, 2023.

1 Introduction

In today's world, the escalating reliance on complex computers, communication networks, and control systems is conspicuous. These multifaceted systems have pervaded diverse spheres of human existence, spanning from handheld devices to nuclear facilities. They constitute the bedrock of modern civilization, underpinning critical functions encompassing power generation, water distribution, transportation systems ...etc. Consequently, the imperative of guaranteeing the accuracy, dependability, and safety of these intricate systems is paramount, [1].

Over the preceding half-century, a procession of accidents and calamities has unfolded due to control systems malfunctioning such as the Three Mile Island Nuclear Accident (1979), the Chernobyl Nuclear Disaster (1986), the Therac-25 Radiation Therapy Accidents (1985-1987), the Ariane 5 Rocket Failure (1996), and the Deepwater Horizon Oil Spill (2010), among others, [1], [2], [3]. Insights from these disasters have to be an integral part of planning new systems, and more measures have to be taken to prevent the repetition of such disasters.

Insights from Health and Safety Executive- Great Britain's study titled " *Out of control: Why controls system go wrong and how to prevent failure*", [4], reveal that a notable 44% of control system failures can be ascribed to inadequate specifications. This percentage can be disaggregated into two segments: specifically, 12% originates from inadequacies in the specification of functional requirements, while a substantial 32% is linked to shortcomings in articulating safety integrity requirements. This observation underscores the compelling reality that a significant portion of control system failures necessitate attention during the initial phases of development.

Addressing this challenge has prompted a shift in the systematic development processes, transitioning away from conventional methodologies like the waterfall model and the widely adopted V-model towards an approach known as model-based development, [5], where the majority of bug and design flaw detection occurs post-product development or prototype creation. For example, under the V-model approach, over 30% of identified errors emerge after unit testing, and more than 40% manifest after system testing, [1].

The approach of model-based development hinges upon the construction of a system model during the developmental process, followed by iterative cycles of simulation and testing across various stages. This contains key phases like Model-In-the-Loop (MIL), Software-In-the-Loop (SIL), and Hardware-In-the-Loop (HIL) testing. By adopting this methodology, there is potential to reduce development expenditures, primarily through early detection of faults, which in turn reduces the expenditures associated with their resolution, [5]. However, it is imperative to acknowledge that this methodology

relies heavily on the utilization of testing and simulation techniques, thereby inheriting their inherent weakness, which predominantly manifests as a reduced level of assurance, particularly in scenarios where a substantial number of test cases are involved, [1]. This can be unacceptable if the control system is used in safety-critical applications.

The significance of model-checking has been steadily increasing due to several compelling factors. Primarily, it operates as an a priori verification technique, engaging in the verification process through a system model before the existence of an actual product or prototype. This proactive approach increases the detection of defects even preceding unit and system testing phases, [1]. Moreover, model-checking has substantial coverage and assurance capabilities, [1]. Furthermore, its foundation in mathematics and classification within the realm of formal methods accentuates its prominence. Notably, formal methods such as model-checking are "*Highly recommended*" for verification in safety-critical systems standards like IEC 61508 (Functional safety standard of electrical/ electronic/ programmable electronic safety-related systems), ISO 26262 (Road vehicles functional safety standard), …etc., [6]. Additionally, from the control system perspective, the work of, [5], expounds upon the advantages of incorporating a system model and formal specification within control systems. This integration serves to bridge the gap between continuous-time and discrete-time domains and offers a framework for both model-checking and controller synthesis, aligning with a correct-by-design methodology, [5].

It is worth keeping in mind that control systems are not assigned the safety role in all safety-critical systems as safety standards such as IEC 61508 emphasize the need for a clear separation between the control system and the safety function, if possible, [6]. Normally, a simple and reliable safety function is used for the safety role, However, the controller itself may be a safety-critical system such as the flight controller in the X-29 aircraft, [5].

Note: The main goal of this paper is to study dynamical control systems in a model-checking environment, modelling using hybrid transition systems, and formulating formal requirements to describe such systems. Conventional control system topics such as stability analysis as well as anti-windup measures and techniques from control theory, that are used to minimize the effects of the integral windup, [5], are outside the scope of his paper.

2 Problem Formulation

This study focuses on employing model-checking techniques to verify control systems. It emphasizes the necessity of creating a transition system model that accounts for the system's dynamic behaviours. However, using model-checking tools for this purpose is challenging due to their original design not accommodating dynamic system verification complexities.

One major challenge is the lack of support for floating-point parameter representation in model-checking tools, despite its importance in capturing accurate system dynamics. Unfortunately, this representation can lead to the state space explosion problem during model-checking, [7]. Previous work mentioned in, [7], introduced an ad-hoc data representation-based approach to describe dynamical control systems within the model-checking framework. However, this work doesn't address the integral windup issue.

The study also addresses the challenge of formulating formal requirements to verify aspects related to the integral windup phenomenon. The conventional Computational Temporal Logic (CTL) and Linear Temporal Logic (LTL) languages, designed for state-based systems, are inadequate for describing control system dynamics.

Building upon previous methodologies, [7], the study aims to create a comprehensive control system model, including an actuator model to tackle the integral windup problem. Additionally, it explores using formal languages like LTL and CTL to effectively describe this phenomenon. The goal of this paper is to advance model-checking techniques for verifying control systems, particularly concerning integral windup.

3 PID Controller and Integral Windup Phenomena

3.1 Dynamical Control Systems

The advent of transistors brought about a significant revolution in control systems, leading to the birth of a new field known as digital control in the 1950s. The introduction of embedded systems flexibly enabled the implementation of controllers, allowing for easy updates through software modifications, [7].

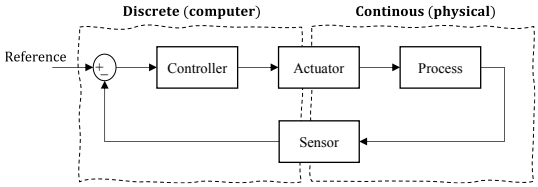

Fig. 1: Simplified Block Diagram of Dynamical Control System, adapted from, [7].

Fig. 1 shows a simplified block physical process regulated by a computer-based controller (without considering the disturbances). The main components of the system are the process, the actuator, the sensor, and the controller.

The processes involved in control systems are often physical and continuous in nature. In these systems, the sensors and actuators play a crucial role as they serve as the interface between the discrete and continuous worlds, [5]. Different types of controllers can be used such as PID controller, Linear Quadratic Regulator (LQR), Model Predictive Control (MPC) …etc. This paper focuses on the PID controller, in particular the PI variant.

The PID controller is the most used in the industry due to its simplicity and effectiveness. It consists of three essential components: the proportional term (P), the integral term (I), and the derivative term (D). The input to the PID controller is the error signal, calculated as the discrepancy between the desired reference value and the measured output obtained from the sensor. The PID controller processes this error signal and generates an output known as the control signal. The control signal is then transmitted to the actuator, which converts it into the appropriate power or action required by the system, [8].

3.2 Integral Windup

Integral windup is a nonlinear effect commonly observed in control systems when actuators have limitations, as shown in Fig. 2. These limitations are typically imposed by the physical capabilities of the actuators, such as valves having fully open or closed positions and maximum flow rates, or electric motors having maximum speeds or torque outputs. Additionally, safety considerations may also impose limits to protect the system or equipment, [5].

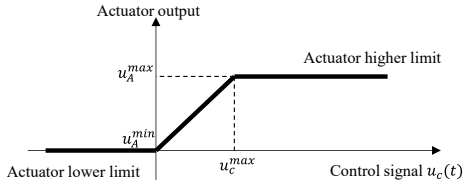

Fig. 2: Saturation Characteristic of an Actuator, [8].

Integral windup occurs when the controller commands a control action that requires the actuator to operate beyond its limits. As a result, the feedback loop is disrupted, and the system operates in an open-loop manner. The actuator output becomes saturated at its limits, rendering it independent of the process output. Since the error does not converge to zero, the integrator continues accumulating the error over time. Consequently, the output of an integral part becomes excessively large, and it takes a considerable amount of time to remove the saturation effect. This phenomenon significantly affects the transient response of the system, resulting in slower performance, [5].

Fig. 3 illustrates the integral windup phenomenon in a first-order system controlled by a PI controller with an actuator that exhibits saturation characteristics. The figure is divided into two halves, the lower part shows the integral control effort, the upper part shows the reference signal in orange and the system output in blue.

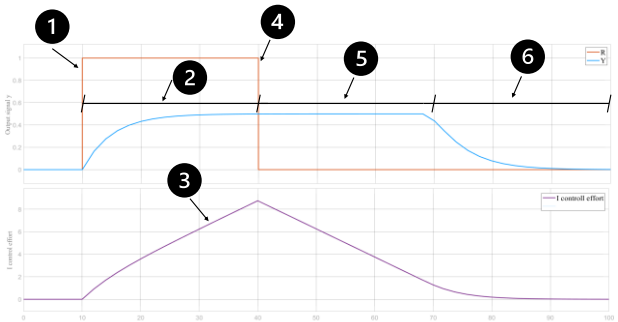

Fig. 3: Simulation of the Integral Windup Phenomena.

The icons shown in Fig. 3 can be summarized as the following:

1. First reference change as a step input.
2. The system is not able to track the input because the actuator is saturated.
3. The integrator is accumulating the error.
4. A second reference change is opposite to the first one.
5. Faulty output of the controller, as it is applying the maximum output, but in the wrong direction.
6. The system tracks the input too late.

4 Model-Checking

Model checking stands as a powerful formal verification technique that systematically and exhaustively analyses all possible states of a system, [1]. To execute model-checking three main things as needed: a model of the system or its components as a transition system (state machines), a set of formal requirements written in languages such as LTL or CTL, and a model-checking tool such as UPPAAL, SPIN or MuSMV as depicted in Fig. 4.

Note: UPPAAL, SPIN and MuSMV are model-checking tools. UPPAAL, the tool that was used for this work, is a model-checking tool developed in collaboration between the Department of Information Technology at Uppsala University in Sweden, and

the Department of Computer Science at Aalborg University in Denmark.

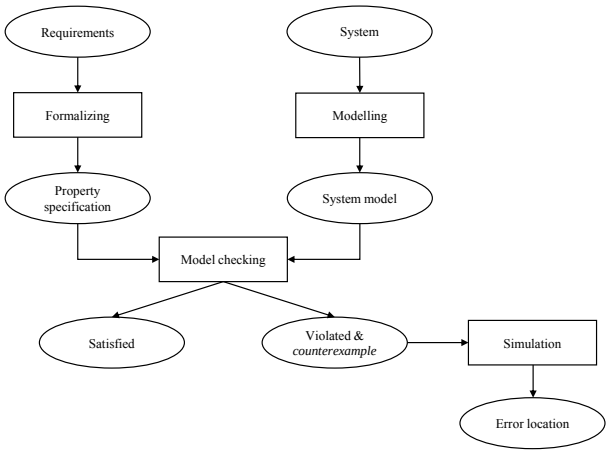

Fig. 4: Schematic View of Model-checking, [1].

The system model and formal requirements are fed into the model-checking tool, and with a "*push button*" the tool then verifies the model against the requirements, yielding one of three outcomes, [1]:

1. **Satisfied**: The model meets the property.
2. **Violated**: The model fails the property, with a counterexample provided.
3. **Not Enough Memory**: Insufficient resources for model-checking.

4.1 Modelling

Models required for model-checking are parts of the transition systems family. The nodes in transition systems represent states, and the edges represent transitions between states, [7].

UPPAAL tool is used for the work of this paper, and it uses Timed Automata which is a transition system extended with timing notations. A timed automaton is a "*program graph that is equipped with a finite set of real-valued clock variables called clocks*", [1]. Fig. 5 shows an example of a timed automaton of a gate that opens and closes on request, time constraints on nodes and edges and modelled using the clock *x*, [7].

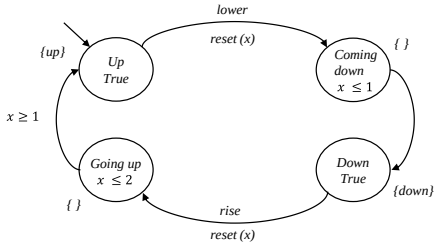

Fig. 5: Timed Automaton Example of a Gate opens and closes on request, [7].

Timed automata is defined as a tuple $(S, I, Act, C, \rightarrow, Inv, AP, L)$, [1], where:

- S is a set of stats,
- $I \subseteq S$ is a set of initial states,
- Act is a set of actions,
- C is a set of clocks,
- $\rightarrow \subseteq S \times CC(C) \times Act \times 2^C \times S$ is a transition relation,
- $Inv : S \rightarrow CC(C)$ is an invariant-assignment function,
- AP is a set of atomic propositions, and
- $L : S \rightarrow 2^{AP}$ is a labelling function.

where $CC(C)$ is a set of clock constraints over clock C, and the invariants are clock constraints on the states.

4.2 Requirements using Formal Languages

Formal languages are essential for describing and verifying system model properties clearly and unambiguously. When it comes to model checking, human languages are not suitable for expressing requirements. Instead, formal languages such as LTL and CTL provide the necessary tools to specify linear-time and branching-time properties for transition systems.

The main distinction between LTL and CTL lies in their treatment of time. LTL assumes linear time, where each state is followed by a single successor state. It enables the expression of properties that hold across all possible system paths. On the other hand, CTL assumes branching time, allowing for multiple successors from each state. This makes CTL suitable for capturing properties that depend on different system branches and choices. Furthermore, CTL permits the explicit inclusion of time values using clocks, which is not possible in LTL. Certain properties can be expressed in one but not the other, and some equivalences exist between properties. For more information on both LTL and CTL, [7].

The syntax of CTL has two stages: state and path formulas. The state formula over a set of atomic propositions is defined as the following, [1]:

$$\Phi ::= true \mid p \mid \Phi_1 \wedge \Phi_2 \mid \neg \Phi \mid \exists \varphi \mid \forall \varphi$$

where $p \in AP$ and φ is the path formula. $\exists$ (*Exist*) is a path quantifier (means for some path), and $\forall$ (*All*) is a state quantifier (means for all states) [1].

The path formulae are defined as the following, [1]:

$$\varphi ::= X\Phi \mid \Phi_1 U \Phi_2$$

where φ is a path formula, X is the next operator, and U is the until operator, [1].

The previous operators can be used to form other operators such as G (*Globally*) and F (*Eventually*). Fig. 6 shows two CTL traces, Lift: $\forall Gp$ (for all states globally the atomic proposition p appears). Right: $\exists Fp$ (At least for one path the atomic proposition p appears eventually).

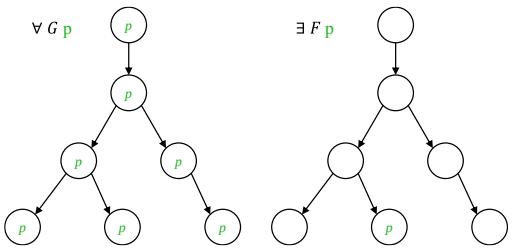

Fig. 6: Traces safeties the CTL requirements $\forall Gp$ and $\exists Fp$.

5 Model of a Dynamical Control System Using Timed-Automata

Model-checking tools do not support floating-point data types, [7]. To include the system dynamics and perform calculations of plant and controller in a model-checking tool, all the necessary parameters shall be represented using a fixed-point data type variable. For example, for the following transfer function, the parameters *K, T, Y* and *U* need to be represented using fixed-point data variables.

$$G(s) = \frac{Y(s)}{U(s)} = \frac{K}{Ts+1} \quad (1)$$

To use the transfer function in a model-checking tool, [7], introduces Algorithm 1, shown in Table 1, to convert a continuous Single Input Single Output (SISO) Linear Time-Invariant (LTI) transfer function into a suitable form for model-checking tools.

Moreover, [7], introduces model bases for a control system consisting of three time automatons: plant, controller and observer, as shown in Fig. 7. The plant has states such as *Settled, Transient, Rise time* and *Overshoot.* It is responsible for performing the plant calculations necessary for calculating the dynamics of the plant. The controller has states such as *Settled* and *Transient*, and it is responsible for performing the controller calculation required to calculate the controller output from an error input. In addition, an observer is used to synchronize the calculation between the plant and the controller, [7].

Table 1. Algorithm to Convert SISO LTI Transfer Function to be Used in a Model-checking Tool.

Algorithm 1 Convert a SISO Linear Time Invariant Model to be used in a Model-Checking Tool	
Input: Continuous SISO LTI model, sampling time T_s, K_U gain. *Output:* Discrete SISO LTI model, K_{ab} coefficients scaling gain, K_S input/output scaling gain.	
1	Discretize the SISO LTI model using one of the discretization methods and sampling time T_s.
2	Define the input/ output ranges
3	Select the digit numbers of the fixed-point representation (I, F) based on step 2. Where I is the number of decimal digits and F is the number of fractional digits.
4	Select K_S based on step 3.
5	Based on I and F in step 3, round off the coefficients in the described SISO LTI model in step 1.
6	Verify that the non-zero fractional digits in the model coefficients can be recovered by scaling them.
7	Based on step 3, calculate K_{ab}: closest value to 10^F which is a power of 2.
8	Recalculate coefficients using K_U gain
9	Apply the final value theorem for the generated transfer function.
10	Select the transfer function with the highest and lowest final value theorem.
11	Scale coefficients using K_{ab} gain and round off to integer values.

Note: using Algorithm 1 required identifying the number of the decimal digits *I*, and the number of fractional digits *F*. For example, choosing (*I*=1, *F*=4) results in a limited range of the *U* and *Y* values between $(-3.2767 \cdots 3.2767)$ and it shall be mapped to $(-32767 \cdots 32767)$ in UPPAAL, [7].

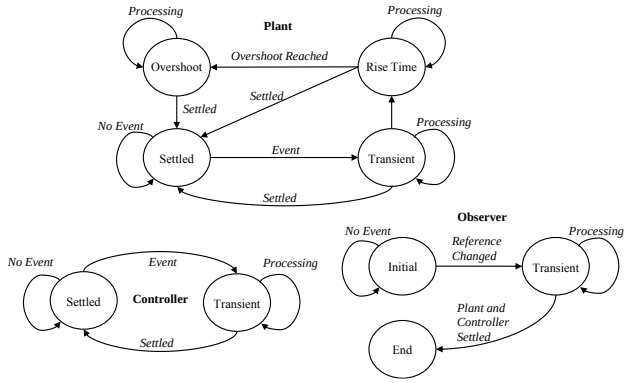

Fig. 7: Bases of Modelling a Control System in Timed-Automata, [7].

6 Case Study (PID Controller and Integral Windup)

6.1 Modelling

Using the models, model bases and the models provided in, [7], as a reference, a control system consisting of a first-order plant, PI controller, observer and actuator, was constructed in the UPPAAL tool as shown in Fig. 8. The model was tailored with the focus on the integral windup issue.

The following first-order transfer function was taken as an example of the first-order plant:

$$G(s) = \frac{Y(s)}{U(s)} = \frac{1}{15s+1} \quad (2)$$

It was converted to be used in UPPAAL using Algorithm 1 to:

```
y= (15327*Y_1+1057*U_1)/16384;
// system output
```

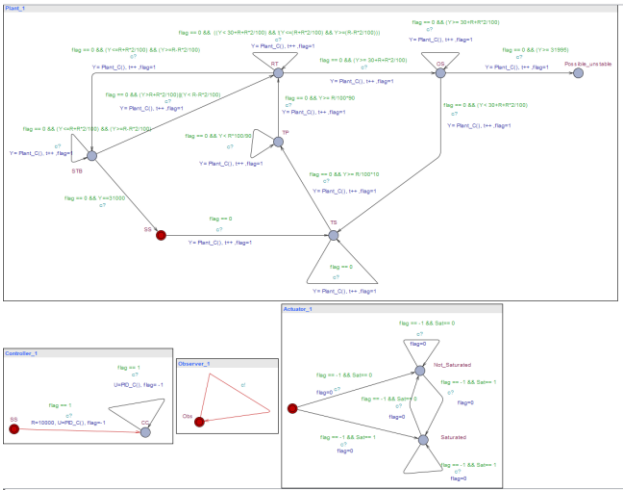

Fig. 8: Model of a Control System Consists of First Order Plant, PI Controller, Observer and Actuator in UPPAAL.

The controller transfer function was:

$$G_{PI}(s) = \frac{U(s)}{E(s)} = K_P + \frac{K_I}{s} \quad (3)$$

with $K_P = 2$ and $K_I = 0.2$ as an example.

The controller transfer function was converted to the following piece of code using Algorithm 1 to be used in UPPAAL:

```
int u_P= 2*E;
// Proportional controller output

int u_I =
(1638*E/16384)+(1638*E_1/16384)+U_1I;
//Integral controller output

int u= u_P + u_I;
//PI controller output
```

The actuator model is essential for the integral windup issue. It was modelled as three states: the initial state, the state *Not_Saturated* which represents the actuator in its linear range before reaching its maximum or minimum limits, and the state *Saturated* which represents the state of the actuator when it reaches its higher or lower limits. Fig. 9 shows the model of the actuator in UPPAAL.

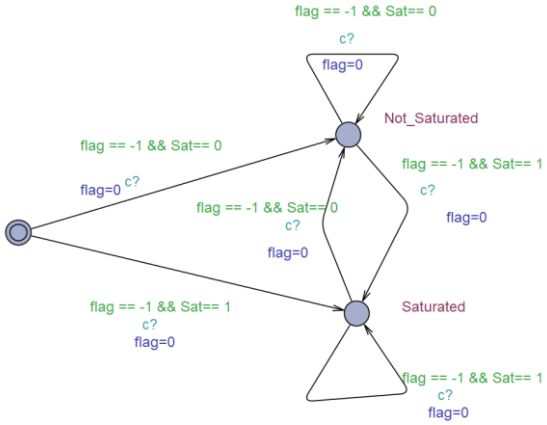

Fig. 9: Model of the Actuator in UPPAAL.

6.2 Simulation

Before initiating the model-checking process, it was imperative to execute a comparative analysis between the UPPAAL model and its counterpart implemented in MATLAB/Simulink®. This serves the dual purpose of validating the outcomes derived from the abstraction method and ensuring the precision of the system dynamics calculations. The outcomes of this analysis are presented in Fig. 10.

Fig. 10 shows the simulation results generated by both MATLAB/Simulink® and UPPAAL tools are juxtaposed. Remarkably, the findings reveal that the disparities between the data obtained from these two tools manifest predominantly in the fifth digit, signifying variations within the fourth fractional digit.

It is significant to understand that when the transfer function parameters or controller gains representation requires a greater number of decimal digits, the level of accuracy attainable through UPPAAL results is subject to degradation, as the highest accuracy can be achieved using the number of decimal digits I to be one, and the number of fractional digits F to be four.

6.3 Verification through model-checking

The verification phase was executed through the formulation of formal requirements that required to be checked. Diverse sets of requirements were formulated to assess aspects, including the plant's susceptibility to overshooting or progressing into an unstable state, as well as addressing the intricate phenomenon of integral windup.

As depicted in Fig. 11, the results of the model-checking verification process, facilitated by UPPAAL, are presented herein. Within this visual representation, requirements that have been successfully met are denoted by a reassuring green colour, while those that remain unfulfilled are distinctly highlighted in red. Furthermore, it is worth

noting that instances of an orange may arise when sufficient computational resources are not available.

The main written requirement regarding the integral windup issue was:

E<>(Actuator_1.Saturated)&&((E>E_1)||(E==E_1))

The preceding requirement is formulated to check the presence of integral windup within the system. It systematically examines whether there exists at least one path in which the actuator operates in the *Saturated* state while concurrently maintaining or surpassing the current error value compared to its prior state. In other words, it checks if the actuator is saturated while the error remains constant or increases.

The model-checking result of the previous requirements is green, which means that the requirement is satisfied, and the system's actuator reached its limits. These results were also validated with MATLAB/Simulink® simulations, and both results were the same.

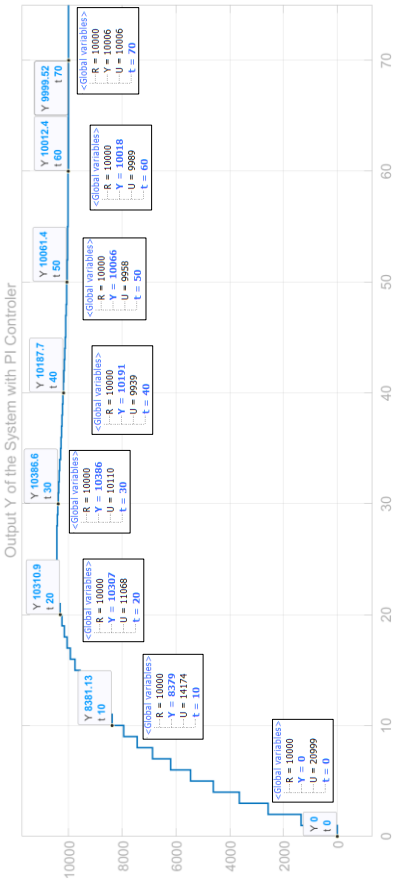

Fig. 10: Simulation Results from MATLAB/Simulink® and UPPAAL. The Results from MATLAB/Simulink® are Represented Using the Blue Plot and the Corresponding Data Tips. The Results from UPPAAL are Represented Using 8 Screenshots Labelled with (Global variables).

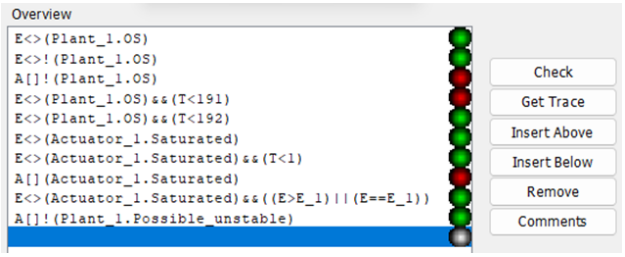

Fig. 11: Model-Checking Results in UPPAAL.

7 Conclusion

In conclusion, this scientific paper confirms the importance of model-checking as an effective approach for verifying system designs, offering benefits such as early error detection, enhanced safety, and cost savings, particularly for hybrid systems. Model-checking provides extensive coverage and addresses limitations associated with testing and simulation. However, challenges exist in accurately constructing models and expressing requirements in formal languages, particularly when dealing with phenomena like integral windup.

The UPPAAL tool is highlighted as a powerful and user-friendly choice, with its visual modelling feature offering a significant advantage over other model-checking tools relying on textual modelling. This visual approach reduces systematic errors and aids in bug detection within models. Nevertheless, the effectiveness of modelling and verification in UPPAAL depends on system complexity and specific requirements, which can be challenging for complex systems.

While UPPAAL was not originally designed for modelling dynamic control systems, this paper demonstrates its adaptability for building hybrid control system models and performing simulation and model-checking verification. UPPAAL's strength lies in its formal verification capabilities, which are not typically found in traditional control system simulators like MATLAB/Simulink®.

A notable weakness of UPPAAL is the absence of libraries containing pre-existing functions or examples to assist users in learning the tool, necessitating a "start from scratch" approach for modelling and specifying requirements for specific systems.

Using UPPAAL requires constructing system models, which can be challenging and demands prior knowledge about the system's components. Additionally, the omission of factors like noise, time delay, or disturbances in the model adds potential sources of modelling errors.

The process of simulating models in UPPAAL is straightforward and valuable, allowing for the

identification and correction of modelling faults and the improvement of model accuracy. The comparison of UPPAAL results with those from MATLAB/Simulink® demonstrates UPPAAL's ability to produce accurate outcomes, although precision may be influenced by variable value ranges.

In principle, the verification process in UPPAAL is a "push-button" activity once the system is properly modelled. However, challenges arise in translating requirements from human-written languages into temporal logic languages and ensuring the correctness and completeness of both the model and the requirements, which can impact the accuracy of model-checking results.

In summary, model-checking stands as a powerful technique within the realm of formal verification. However, it is essential to recognize that model-checking verifies a model of the system rather than the system itself. To fully leverage the benefits of model-checking, it should be combined with testing, simulation, safety analysis, and other verification techniques. This comprehensive approach becomes even more crucial when dealing with safety-critical systems. In such cases, employing a variety of verification methods and techniques becomes imperative to ensure the utmost safety assurance for the system.

References

[1] C. Baier and J.-P. Katoen, *Principles of model checking*. Cambridge, Mass., London: MIT, 2008.

[2] "Deepwater Horizon – BP Gulf of Mexico Oil Spill," [Online]. Available: https://www.epa.gov/enforcement/deepwater-horizon-bp-gulf-mexico-oil-spill#:~:text=On%20April%2020%2C%202010%2C%20the,of%20marine%20oil%20drilling%20operations (accessed: Jun. 12 2023).

[3] *Chernobyl | Chernobyl Accident | Chernobyl Disaster - World Nuclear Association.* [Online]. Available: https://www.world-nuclear.org/information-library/safety-and-security/safety-of-plants/chernobyl-accident.aspx (accessed: Jun. 12 2023).

[4] Great Britain. Health and Safety Executive, *Out of control: Why controls system go wrong and how to prevent failure,* 2nd ed. Sudbury: HSE Books, 2003.

[5] K. J. Åström and R. M. Murray, *Feedback systems: An introduction for scientists and engineers / Karl Johan Åström, Richard M. Murray*. Princeton, New Jersey: Princeton University Press, 2021.

[6] *Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems (E/E/PE, or E/E/PES).*, IEC 61508, 2010.

[7] Pablo Armando Ordóñez Aguileta, "Formal Design and Verification of Digital PID Gain Scheduling Controllers: A Model Checking Approach," PhD thesis, University of Sheffield, 2018. Accessed: Feb. 22 2023. [Online]. Available: https://etheses.whiterose.ac.uk/19465/

[8] M. A. Johnson, M. H. Moradi, and J. Crowe, *PID control: New identification and design methods / Michael A. Johnson and Mohammad H. Moradi (editors); with J. Crowe, K.K. Tan, T.H. Lee, R. Ferdous, M.R. Katebi, H.-P. Huang, J.-C. Jeng, K.S. Tang, G.R. Chen, K.F. Man, S. Kwong, A. Sánchez, Q.-G. Wang, Yong Zhang, Yu Zhang, P. Martin, M.J. Grimble and D.R. Greenwood*. New York, N.Y., London: Springer, 2005.

Contribution of Individual Authors to the Creation of a Scientific Article (Ghostwriting Policy)

The authors equally contributed in the present research, at all stages from the formulation of the problem to the final findings and solution.

Sources of Funding for Research Presented in a Scientific Article or Scientific Article Itself

No funding was received for conducting this study.

Conflict of Interest

The authors have no conflicts of interest to declare that are relevant to the content of this article.