Adaptive Runtime Response Time Control in PLC-based Real-Time Systems using Reinforcement Learning

Mahshid Helali Moghadam[1,3], Mehrdad Saadatmand[1], Markus Borg[2], Markus Bohlin[1], Björn Lisper[3]
[1]RISE SICS Västerås, SE-722 13, Västerås
[2]RISE SICS Lund, SE-223 70, Lund
[3]Mälardalen University, SE-721 23, Västerås
Sweden
{mahshid.helali.moghadam, mehrdad.saadatmand, markus.borg, markus.bohlin}@ri.se, bjorn.lisper@mdh.se

ABSTRACT

Timing requirements such as constraints on response time are key characteristics of real-time systems and violations of these requirements might cause a total failure, particularly in hard real-time systems. Runtime monitoring of the system properties is of great importance to check the system status and mitigate such failures. Thus, a runtime control to preserve the system properties could improve the robustness of the system with respect to timing violations. Common control approaches may require a precise analytical model of the system which is difficult to be provided at design time. Reinforcement learning is a promising technique to provide adaptive model-free control when the environment is stochastic, and the control problem could be formulated as a Markov Decision Process. In this paper, we propose an adaptive runtime control using reinforcement learning for real-time programs based on Programmable Logic Controllers (PLCs), to meet the response time requirements. We demonstrate through multiple experiments that our approach could control the response time efficiently to satisfy the timing requirements.

CCS CONCEPTS

• **Computer systems organization** → **Real-time systems**; • **Software and its engineering** → Software performance; • **Computing methodologies** → Machine learning

KEYWORDS

Adaptive response time control; PLC-based real-time programs; Runtime monitoring; Reinforcement learning

ACM Reference format:

M. H. Moghadam, M. Saadatmand, M. Borg, M. Bohlin, B. Lisper. 2018. Adaptive Runtime Response Time Control in PLC-based Real-Time Systems using Reinforcement Learning. In *Proceedings of 13th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, Gothenburg, Sweden, May 28-29, 2018 (SEAMS '18)*, 7 pages.
DOI: 10.1145/3194133.3194153

SEAMS '18, May 28–29, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5715-9/18/05…$15.00
https://doi.org/10.1145/3194133.3194153

1 INTRODUCTION

Real-time control programs implemented on Programmable Logic Controllers (PLCs) are key parts of many time-critical industrial control systems like those in the railway domain. The timing properties in these systems include period of tasks, deadline, worst-case execution time or response time. From the perspective of timing analysis, schedulability analysis methods, statistical and formal timing analysis are common analysis techniques to provide a response time estimation of real-time programs [1-3]. Static analysis-based approaches, in some cases, might not be practical for complex real-time systems. Even if they are feasible, the results might not be valid due to unpredictable factors in runtime and the difference between analysis environment and the realistic one [4].

Generally, there is often a strict set of timing requirements such as deadlines and limits on response time for real-time programs in mission-critical contexts. Correctness of functionality of real-time systems highly depends on satisfying the timing requirements as important features of these systems. Any serious deviation in temporal behavior of real-time programs due to unpredicted runtime events like asynchronous message-passing and runtime changeable priorities, particularly in complex systems, might cause a total failure in the function of system. Thus, providing more robustness against unpredicted varying conditions during runtime is of great importance. In general, robustness could be defined as to which degree the system is tolerable against incorrect inputs or unexpected stressed conditions [5]. In a real-time program, robustness could be defined as the ability to adapt to the varying conditions while satisfying the timing requirements.

An adaptive runtime control in addition to the scheduling capabilities could lead to more robustness in real-time control systems, to cope with changing runtime conditions and unpredicted states [6]. Runtime monitoring could check if the system adheres to the predefined requirements like timing constraints. A control approach based on runtime monitoring could help preserve these timing properties by applying runtime

control operations. Adaptive control strategies are considered as one of the promising solutions to improve robustness through providing adaptation to the varying conditions in dynamic environments. Reinforcement learning (RL) has been frequently applied to address the adaptive control strategy in dynamic environments, in case the environment is stochastic, and the control problem can be formulated as a Markov Decision Process (MDP).

In this paper, we propose a self-adaptive response time control for real-time programs in PLC-based systems using reinforcement learning. In our previous work [7], we presented the initial idea on how a learning-based solution can be used to provide assurance of timing properties; here in this work we extend that initial idea and provide an industrial evaluation of our proposed approach. We present the evaluation experiments of the proposed approach on sample programs inspired from our collaboration with Bombardier Transportation in Sweden. The proposed approach formulates the response time control problem as an MDP and uses Q-learning as a model-free RL to provide adaptive control of response time while meeting the timing requirement. We show the efficacy of the proposed approach through multiple experiments based on simulating real-time programs in a PLC-based control system. Our approach mostly keeps the programs adhering to the response time constraints despite the occurred time deviations during the run time. Based on the evaluation results, the proposed approach with ε-*greedy,* ε*=0.5,* and α*=0.1* and γ *=0.5* provided better satisfaction of the response time threshold without any programs ending with medium or high deviation.

The rest of this paper is organized as follows; Section 2 discusses briefly the motivation and background concepts of RL. The technical details of the proposed approach are discussed in Section 3, while Section 4 presents the evaluation experiments and results. Section 5 provides a review of the related works and background techniques. Conclusions and future directions are provided in Section 6.

2 MOTIVATION AND BACKGROUND

2.1 Motivation

Runtime monitoring is considered as a principal means for real-time systems. Providing an adaptive control for satisfying the timing requirements such as constraints on response time/execution time based on runtime monitoring could improve the robustness of the system. Model-driven control approaches may require precise knowledge of the system and environment. The complexity of real-time systems, for example, intricate temporal dependencies between real-time tasks and the dynamism of the environment are major barriers which motivate towards model-free learning-based control. Learning-based control can find an adaptive control policy to varying conditions regardless of having a precise model of the environment. Reinforcement learning-based control techniques have been used for runtime control of non-functional properties to satisfy the performance and timing requirements in many application contexts.

Reinforcement learning [8] is a learning mechanism working based on interaction with the environment. In RL, the agent senses the state of the environment continuously, takes a possible action and in return, receives a reward signal from the environment which shows the desirability and effectiveness of the applied action. During the learning, the agent follows a policy which maximizes the long-term received reward. The agent learns this policy through an action selection strategy which is based on selecting an action randomly (exploration) or selecting an action with a high utility value (exploitation). Q-learning [8] is a model-free RL algorithm in which the agent learns the value function of the long-term expected reward associated to the pairs of states and actions. It is an off-policy learning as the optimal policy is learnt independently of the action selection strategy being used by the agent. Once the learning converges, the agent replays the learned policy.

2.2 PLC-based Industrial Control Programs

Many of the real-time industrial control systems like those ones in the transportation domain, are implemented based on IEC 61131-3 [9] which is one of the main programming language standards for programmable controllers. According to the proposed software structure in IEC 61131-3, Programmable Organization Units (POU) are the building blocks of a PLC program. They are hardware-independent and programmable in a flexible fashion facilitating the reusability and modularization in this context.

There are mainly three unified types of POUs: program, function block and function. A function block has its own data record to remember the state of the information, while a function always produces the same result based on the same input. A program may consist of zero or multiple function and function blocks. A real-time task can execute one or multiple programs or a set of function blocks. Timer function blocks are widely used as one of the main constituent POUs in PLC-based real-time programs. Their basic functions involve providing their output after a preset controllable/programmable time interval. There are three types of timers as standard PLC timer blocks, i.e., TP (Timer Pulse), TON (Timer On-Delay) and TOF (Timer Off-Delay). Timer TP is a pulse generator which supplies a constant pulse on output upon detecting a rising edge at input. TON supplies the value of input at output with a delay upon detecting a rising edge at input. TOF has an inverse functionality to TON. Figure 1 shows a sample schema of a real-time control program in Function Block Diagram format, as an integration of multiple functions and function blocks. The number of POUs in each control program depends on the complexity of the program.

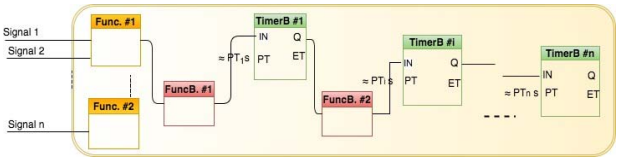


Figure 1: A sample schema of a PLC-based control program

The time delay of timer function blocks in time-critical programs are the target entities supposed to be tuned in urgent conditions by our control approach to satisfy the response time requirements.

3 ADAPTIVE RESPONSE TIME CONTROL USING Q-LEARNING

In this section, we present the technical details of the proposed runtime response time control using reinforcement learning for real-time programs running on PLC-based systems. This control method is incorporated into the control scan program which is responsible for executing the building blocks and preserving their execution orders within real-time programs. Timer function blocks are one of the standard function blocks which are widely used and play a key role in many time-critical industrial control programs.

The proposed control strategy is supposed to use the capability of tuning the time delay of timer function blocks to control the response time of real-time programs. The main objective of the proposed runtime response time control is to meet the response time requirements in abnormal conditions when time deviations happen, by optimally tuning the time delay parameters. The proposed approach uses Q-Learning as a model-free RL to learn the optimal tuning of delay parameters to preserve the program responsive within the target response time threshold.

The learning task in the proposed control approach mainly involves the following steps:

1) Detecting the *State* of the system. Based on the interactive characteristic of the reinforcement learning, the control agent/controller observes the state of the program at discrete time steps. After each execution cycle, the controller measures the execution time until the current time point. The actual execution time until the end of the n^{th} function block execution, ET_n, is classified under four classes. This is done based on the amount of compliance with the desired/target execution time until the end of the n^{th} function block (e.g. from requirements/constraints), T_n, calculated as follows:

$$T_n = \sum_{i=1}^{n} T_i^f \quad (1)$$

Where T_i^f is the desired response time of the i^{th} function block. The class values representing the state of the program, *s*, are *Required*, *Low*, *Medium* and *High*, as shown in Figure 2. They represent the acceptable state, and the states with low, medium and high deviation, respectively. We defined the acceptable state based on a target execution time characterized by a tolerance region $[T_n, T_n^{'}]$ where $T_n^{'} = T_n + \tau$. where τ in *ms* is defined based on the characteristics of the system.

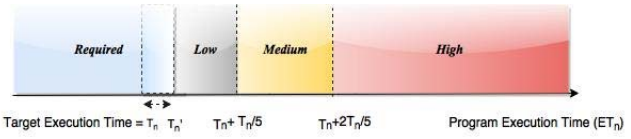


Figure 2: States of the program

2) Selecting a *Control Action.* We defined the control actions as tuning operations for the time delay of the next running function block, D_f^{n+1}.

For providing a safety margin, we also considered a required minimum delay, D_m ,for function blocks.

Then, the time delay of function blocks could not be set to a value less than D_m. Regarding the minimum time delay, we specified a set of control actions for tuning the time delay as follows:

$$Actions = \{(1 - f_d)D_f^{n+1} + f_d D_m : f_d \in K\} \quad (2)$$

$$K = \{0, \tfrac{1}{5}, \tfrac{2}{5}, \tfrac{3}{5}, \tfrac{4}{5}, 1\} \quad (3)$$

Where f_d is a decreasing factor.

3) Receiving the *Reward* signal and updating the stored experience. After applying the selected action, the system will go to the next state and the controller will receive a reward signal representing the effectiveness of the applied action. We derived a utility function based on a Normal probability density function with $\mu = T_n$ and $\sigma = T_n/10$ which is as follows:

$$r_n = \begin{cases} \frac{1}{\frac{T_n}{10}\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{ET_n - T_n}{T_n/10}\right)^2}, & T_n < ET_n \\ 1, & ET_n \le T_n \end{cases} \quad (4)$$

The computed reward values will be in the range (0, 1].

The final objective of the learning is to find a policy π, a mapping between the states and actions, which maximizes the expected long-term reward defined as follows [8]:

$$R_n = r_{n+1} + \gamma r_{n+2} + \cdots + \gamma^k r_{n+k+1} \quad (5)$$

Where $\gamma \in [0,1]$ is a discount factor specifying the importance of future rewards compared to the immediate reward. The long-term expected return of selecting action *a* in state *s*, based on policy π, is specified by a utility value $Q^{\pi}(s,a)$ defined as follows [8]:

$$Q^{\pi}(s,a) = E^{\pi}[R_n | S_n = s, A_n = a], \quad (6)$$

The q-values stored in a look-up table, *Q-table*, form the experience of the agent. The controller relies on q-values to make decision on actions. During the learning, the q-values are updated incrementally via temporal differencing. The agent updates the associated $Q^{\pi}(s,a)$ for each experienced *(s, a)* through the following rule:

$$Q(s_n, a_n) = Q(s_n, a_n) + \alpha[r_{n+1} + \gamma \max_a Q(s_{n+1}, a) - Q(s_n, a_n)] \quad (7)$$

Where $\alpha \in [0,1]$ is the learning rate parameter. It specifies to what extent new information impacts the q-values. The all steps of the adaptive control procedure are described in List 1. Eventually, after multiple learning cycles, the controller finds the optimal policy of selecting the action which maximizes the q-value in a given state.

learning performance. Different action selection strategies could be used during the learning. The agent can use a random action selection method or select greedily an action with the highest utility value according to the Q-table. *ε-greedy* is an action selection strategy which allows the agent to make a trade-off between the exploration and exploitation in the action space. In *ε*-greedy, with probability *ε*, a random action is selected and with probability *1-ε*, an action based on the utility value is selected. However, RL-based approaches might generally suffer slow convergence due to the need for exploring the state space. To alleviate this effect, we also introduced an initial control mapping in Q-table by specifying some invalid pairs of state and action to guide the agent not to explore specific actions in a specific state. For example, when it is in acceptable state, no need to change the time delay parameter.

Algorithm: Adaptive Response Time Control in PLC-based Real-time programs

Required: S, A, ε, α, γ, ϕ (invalid state-action pairs)
Initialize q-values, $Q(s,a) = -1\ if\ (s,a) \in \phi$ else $0\ \forall s \in S, \forall a \in A$
1. Detect the program current state, s_n
2. Select an action using the action selection policy
(e.g. ε-greedy: select $a_n = argmax_{a \in A}\, Q(s_n, a)$ with probability
(1- ε) or a random action with probability ε)
3. Apply the selected action, let the system continue running and execute the next function block
4. Detect the new state of the system
5. Compute the reward (reinforcement) signal.
6. Update the q-value by
$Q(s_n, a_n) = Q(s_n, a_n) + \alpha[r_{n+1} + \gamma \max_a Q(s_{n+1}, a) - Q(s_n, a_n)]$
7. Repeat for every observed state at the start of each function block execution

4 RESULTS AND DISCUSSION

This section presents the results of the early stage evaluation experiments addressing the performance of the proposed approach in terms of meeting the predefined response time threshold. The main objective of the experiments is to assess to which degree the learning-based control can work adaptively on varying conditions and untimely behavior of function blocks in a realistic environment.

4.1 Evaluation Setup

In this study, we implemented the proposed approach based on three action selection strategies. We incorporated it into an environment which simulates multiple real-time programs consisting of various timer function blocks. The simulation environment emulates the temporal behavior of the function blocks, their responses in realistic environments and the corresponding control scan program for controlling the execution order of the function blocks. The learning-based control has been integrated into the control scan thread to provide a runtime control of the response time of real-time programs.

The proposed approach has been evaluated through two analysis scenarios. In the first scenario, concerning response time analysis, the performance of the learning-based control based on using three action selection algorithms has been studied. In this scenario, the performance of the proposed approach after 100 learning episodes (interaction with various real-time programs) has been demonstrated. The real-time programs have been characterized with different numbers of function blocks, predefined response time requirements and minimum required delay time (safety margin). The second analysis scenario, sensitivity analysis, analyzes the sensitivity of the learning-based approach to the learning parameters. This scenario involves investigating the effects of the learning parameters by systematically changing the values of one parameter while keeping the other one constant.

4.2 Experiments and Results

Timing Analysis. In the timing analysis scenario, the efficacy of the learning-based approach was evaluated in terms of adaptation to changeable behavior while meeting the timing requirement. The simulated real-time programs have different numbers of function blocks in the range [5, 25]. The predefined response time requirements of function blocks and associated safety margins in *ms* have been initialized with values in the range [1000, 6000] and [1000, 2000], respectively. A maximum deviation at most equal to 25 percent of the upper bound of the response time requirement was allowed during the simulation. The default acceptable tolerance value was considered as 500 *ms*. Time deviations were injected into the programs randomly. Figure 3 shows a sample pattern for injecting time deviations to function blocks within three sample programs. ε-greedy was used in the proposed approach as an action selection strategy with ε= 0.1, ε=0.5 and ε=0.9. The ε-value determines to what extent exploration and exploitation are weighted during the action selection procedure.
Figure 4 shows the observed response time plots of real-time programs after applying the learning-based control approach based on different values of *ε* parameter in the action selection strategy. Clearly, the learning-based control approach tries to adapt well to the varying temporal behaviors of the function blocks while meeting the response time thresholds of the programs. Results in Figure 4 describe the efficacy of the learning-based control approach based on the number of programs ended with medium or high deviations from the timing requirements and also the achieved average deviations. According to the results, the performance of the proposed approach with different action selection strategies is described as follows:

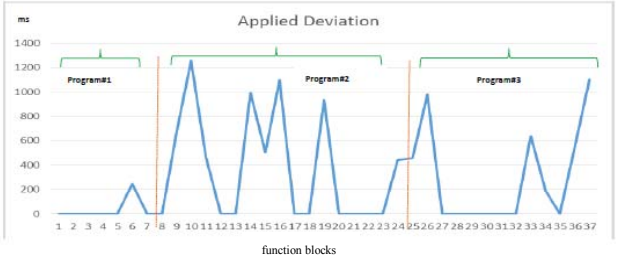


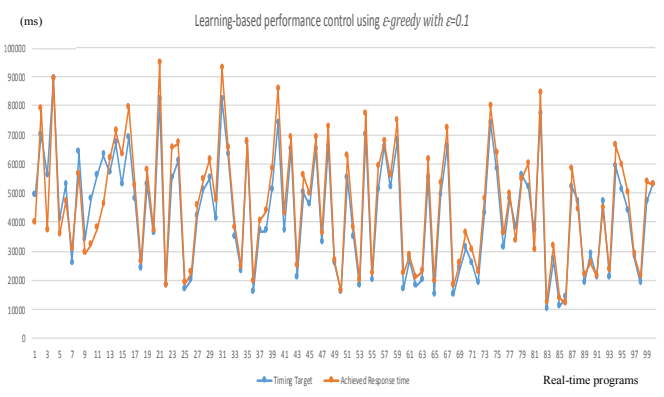
Figure 3: A sample pattern of time deviation

1) *ε-greedy with ε= 0.1* makes the controller trust most on its stored experience, rather than exploring new actions. The learning-based approach based on this action selection strategy, showed less efficiency in terms of optimizing the response time and also the number of programs which ended with medium or high deviations. In this case, the experience of the controller has not been extended well and needs more exploration to be improved.

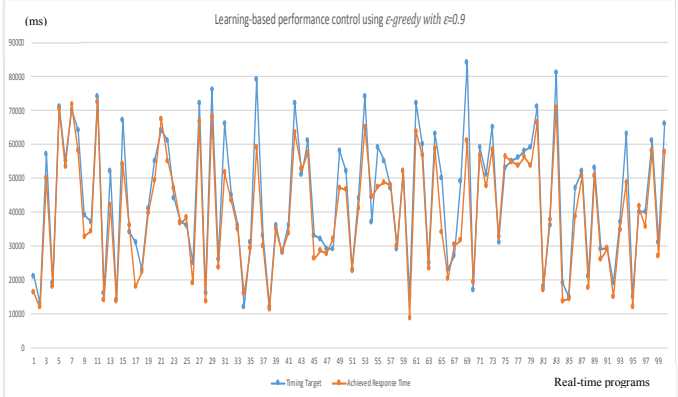
2) ε-greedy with ε= 0.9 provides more opportunities towards the exploration of the action space. It provided partially better performance in terms of optimizing the response time and preventing the programs from exceeding the predefined thresholds with medium or high deviations compared to *ε*-greedy with ε= 0.1.

3) ε-greedy with ε= 0.5 provides a trade-off between the exploration and exploitation of the action space. It showed a better adaptation to the varying conditions and tried to preserve the response time close to the requirement threshold. In some cases where a sharp satisfaction of the timing requirement is needed, e.g. airbag control systems of automotive products, this is the desired performance which is required.

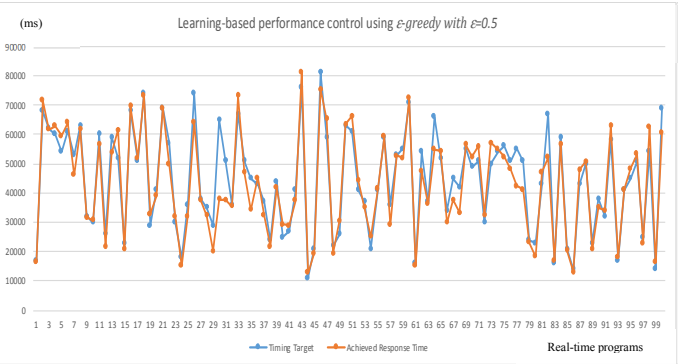
4) ε-greedy with decaying ε, is an action selection strategy during which the ε parameter gradually decreases. It causes more exploration during the first steps of the learning and more exploitation at the last steps. Using this strategy, the performance controller first explores the action space, then tends towards using the achieved experience. The learning-based approach based on ε-greedy with decaying ε, showed the most promising results, i.e., it outperformed the other ε-strategies both in terms of optimizing response time and preventing medium or high deviations from the predefined timing thresholds.



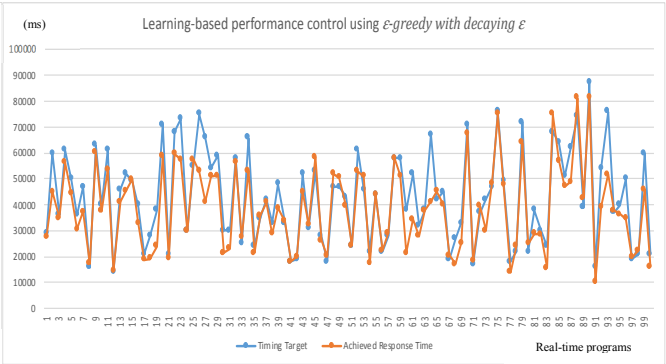
Average injected time deviation per program: *5504 ms*	*LR-based using ε-greedy, ε= 0.1*	*Uncontrolled*
#RT programs with highly exceeded predefined threshold	6	13
Achieved average deviation	2585	5504



Average injected time deviation per program: *5911 ms*	*LR-based using ε-greedy, ε= 0.9*	*Uncontrolled*
#RT programs with highly exceeded predefined threshold	1	7
Average deviation	-3804	5911



Average injected time deviation per program: *5731 ms*	*LR-based using ε-greedy, ε= 0.5*	*Uncontrolled*
#RT programs with highly exceeded predefined threshold	0	10
Average deviation	-1228	5731



Average injected time deviation per program: *5395 ms*	*LR-based using decayingε-greedy*	*Uncontrolled*
#RT programs with highly exceeded predefined threshold	0	8
Average deviation	-4531	5395

Figure 4: response time plots of real-time programs and the performance of the adaptive learning-based control

Sensitivity Analysis. The behavior of the proposed learning-based control approach could be impacted by the learning parameters including learning rate (α) and discount factor (γ). In the sensitivity analysis, two sets of experiments were done to study the effects of varying learning parameters. Each set of experiments involves changing the value of one parameter while keeping the other one constant. ε-greedy with ε= 0.5 was used as a baseline action selection strategy during the sensitivity analysis experiments. Table 1 shows the performance of the learning-based approach regarding the number of real-time programs which ended with medium or high deviations from the predefined response time thresholds and also the achieved average deviation in response time, during the sensitivity analysis experiments. In Table 1 the bold column represents the baseline parameter setting which was used in each sensitivity analysis experiment. We set the learning rate to 0.1 and the discount factor to 0.5 at the first and second experiments, respectively. It seems that setting the learning rate to 0.1, which provides a slower learning, leads to good performance, particularly in adaptation to varying behaviors and preventing the real-time programs from exceeding the timing thresholds. Increasing the learning rate towards 0.5, which aims at balancing between learning new information and saving previous experience, causes improvement in optimizing the response time of the programs. The proposed approach also does not show as much performance improvement as when we set the discount factor to values other than 0.5.

5 RELATED WORK

We classify the relevant works on timing properties of real-time systems under modeling, verifying and some approaches to preserve and satisfy the timing requirements. Many of the verification and preservation/control approaches are based on runtime monitoring of the properties. Real-time Specification for Java (RTSJ) was introduced to provide a real-time scheduler with the facility of monitoring deadlines and enforcing the execution cost [10]. Mezzetti et al [11] used the Ada Ravenscar Profile for preserving the timing properties of real-time systems. Saadatmand et al [12, 13] developed an extra scheduler taking the temporal properties including period, execution time and deadline of the tasks and scheduled them using the underlying scheduler of the operating system. A model synthesis approach for timing properties of real-time systems based on monitoring the running system was proposed in [14]. A runtime framework for monitoring the runtime constraints such as timing constraints and detecting the violations of timing properties was presented in [15]. The related issues on runtime monitoring of properties in real-time systems were discussed in [16]. Goodloe et al [6] surveyed different runtime monitoring techniques including off-line and on-line techniques for distributed real-time systems, in particular hard real-time systems. Das et al [17] presented a tool environment which provided runtime monitoring, animating the development and analysis of the components to support model-driven development of real-time embedded systems. In [18] a runtime monitoring approach for checking the system properties in embedded systems was presented. It used a control method to coordinate the time predictability and memory utilization in the monitoring solution.

6 CONCLUSION

Runtime monitoring of system properties remains as a principal need for real-time systems. A runtime control approach based on runtime monitoring could improve robustness of the system. In this paper, we present an adaptive runtime response time control based on reinforcement learning for PLC-based real-time programs, to satisfy the timing requirements. In this study, we formulate the control problem as an MDP and apply Q-learning to provide a control technique to preserve the response time according to the timing requirements. We evaluate the efficacy of the approach through multiple experiments. The learning-based approaches generally require multiple learning trials to converge and stabilize the learned policy. Regarding this issue and the characteristics of soft and hard real-time systems, it is supposed that the proposed learning-based approach in its incremental learning fashion could be used in soft real-time systems. While the controller with the converged policy, after training based on simulation environment, could be integrated into the hard real-time systems. Furthermore, the result values (the tuned values) of the control policy could be used as a feedback to correct the initial model of the system. Future directions of this study will be evaluating the efficacy of the approach in the industrial platforms, improving the training time and adaptation precision of the approach by modeling the state space as fuzzy state space and using cooperative agents to speed up the learning.

ACKNOWLEDGMENTS

This research has been funded by the ITEA3 initiative TESTOMAT Project (www.testomatproject.eu) through Vinnova and by KKS through the TOCSYC project).

Table 1: Impacts of varying learning parameters on the performance of control approach

	LR-based performance control using ε= 0.5, Discount factor γ=0.5			*LR-based performance control using ε= 0.5, Learning rate α= 0.1*		
	α= 0.1	*α= 0.5*	*α= 0.9*	*γ = 0.1*	***γ = 0.5***	*γ = 0.9*
#RT programs with highly exceeded predefined threshold (Uncontrolled Condition)	**0 (10)**	0 (6)	0 (3)	3 (8)	**0 (10)**	0 (7)
Average deviation (Uncontrolled Condition)	**-1228 (5731)**	-6475 (5378)	-4395 (5455)	-1052 (5484)	**-1228 (5731)**	-1210 (5744)

REFERENCES

[1] G. A. Kaczynski, L. L. Bello, and T. Nolte. 2007. Deriving exact stochastic response times of periodic tasks in hybrid priority-driven soft real-time systems. In *Proceeding of IEEE Conference on Emerging Technologies and Factory Automation* (ETFA). IEEE, 101-110.

[2] S. Manolache, P. Eles, and Z. Peng. 2004. Schedulability analysis of applications with stochastic task execution times. ACM Transactions on Embedded Computing Systems (TECS) 3, no. 4 (2004): 706-735.

[3] E. Fersman, P. Krcal, P. Pettersson, and W. Yi. 2007. Task automata: Schedulability, decidability and undecidability. Information and Computation 205, no. 8 (2007): 1149-1172.

[4] M. Saadatmand, Antonio Cicchetti, and Mikael Sjödin. 2012. Design of adaptive security mechanisms for real-time embedded systems. In *Proceeding of International Symposium on Engineering Secure Software and Systems*. Springer, Berlin, Heidelberg, 121-134.

[5] Standard Glossary of Software Engineering Terminology (ANSI), The Institute of Electrical and Electronics Engineers Inc. 1991.

[6] A. E. Goodloe, and Lee Pike. 2010. Monitoring distributed real-time systems: A survey and future directions. NASA/CR-2010-216724, (2010).

[7] M. H. Moghadam, M. Saadatmand, M. Borg, M. Bohlin, B. Lisper. 2018. Learning-Based Self-Adaptive Assurance of Timing Properties in a Real-Time Embedded System, In *Proceeding of* 2nd International Workshop on Testing Extra-Functional Properties and Quality Characteristics of Software Systems (ITEQS'18), IEEE.

[8] R. S. Sutton, A. G Barto. 1998. Reinforcement learning: An Introduction. Vol. 1. MIT press Cambridge.

[9] International Standard IEC 61131-3, Programmable controllers-part 3 Programming languages, 2013.

[10] A. Wellings, G. Bollella, P. Dibble, and D. Holmes. 2004. Cost enforcement and deadline monitoring in the real-time specification for Java. In *Proceeding of the Seventh IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*. IEEE, 78-85.

[11] E. Mezzetti, M. Panunzio, and T. Vardanega. 2010. Preservation of timing properties with the ada ravenscar profile. In *Proceeding of International Conference on Reliable Software Technologies*. Springer, Berlin, Heidelberg, 153-166.

[12] M. Saadatmand, M. Sjodin, and N.U. Mustafa. 2012. Monitoring capabilities of schedulers in model-driven development of real-time systems. In *Proceeding of 17th IEEE Conference on Emerging Technologies Factory Automation* (ETFA). IEEE, 1–10.

[13] N. Asadi, M. Saadatmand, and M. Sjödin. 2013. Run-Time Monitoring of Timing Constraints: A Survey of Methods and Tools. In *Proceeding of the Eighth International Conference on Software Engineering Advances*.

[14] J. Huselius and J. Andersson. 2005. Model Synthesis for Real-Time Systems. *In Proceedings of the Ninth European Conference on Software Maintenance and Reengineering* (CSMR '05). IEEE, 52–60.

[15] F. Jahanian. 1995. Run-time monitoring of real-time systems. Advances in Real-Time Systems. Prentice Hall (1995).

[16] H. Thane. 2000. Design for Deterministic Monitoring of Distributed Real-Time Systems. Technical Report ISSN 1404-3041 ISRN MDHMRTC- 23/2000-1-SE, Mälardalen University, May 2000.

[17] N. Das, Suchita Ganesan, Leo Jweda, Mojtaba Bagherzadeh, Nicolas Hili, and Juergen Dingel. 2016. Supporting the model-driven development of real-time embedded systems with run-time monitoring and animation via highly customizable code generation. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*, ACM, 36-43.

[18] R. Medhat, Borzoo Bonakdarpour, Deepak Kumar, and Sebastian Fischmeister. 2015. Runtime monitoring of cyber-physical systems under timing and memory constraints. ACM Transactions on Embedded Computing Systems (TECS) 14, no. 4 (2015): 79.