Proceedings of the
2nd Workshop on Artificial Intelligence and Formal Verification, Logics, Automata and Synthesis (OVERLAY),
September 25, 2020

Formal Runtime Monitoring Approaches for Autonomous Vehicles*

Saumya Shankar[1], Ujwal V R[1], Srinivas Pinisetty[1], and Partha Roop[2]

[1]Indian Institute of Technology, Bhubneswar
[2]University of Auckland, New Zealand
[1]{ss117,vru10,spinisetty}@iitbbs.ac.in
[2]p.roop@auckland.ac.nz

Abstract

Consumer interest for autonomous vehicles is growing around the world. Formal verification techniques are needed for thorough verification and validation of such safety-critical systems. Applying static verification techniques for such complex systems that are also increasingly designed and developed using Artificial Intelligence based approaches is challenging and has limitations. In this work, we propose the use of light-weight dynamic formal verification approaches, runtime verification and enforcement. We prototype a self-driving car and propose to apply runtime monitoring to ensure safety of the vehicle. For the development of the prototype, we use Raspberry pi as a master device and Arduino Uno as a slave for steering the vehicle. We use various image processing methods to develop a working hardware prototype model. The output of the controller is fed to the monitor (generated using formal runtime monitor synthesis approach), which enforces desired safety policies on the output of the system. We propose that these formal dynamic monitoring approaches can also be used on Neural Network based controllers. The developed hardware model can act as a test bed to illustrate practical applicability of formal runtime monitor synthesis theory and tools in the context of cyber-physical systems such as autonomous vehicle.

1 Introduction

An Autonomous Vehicle (AV), also known as a self-driving car is capable of sensing its environment and moving safely with little or no human intervention [1]. Surveys [2] say that by the end of 2030 most of the cars on the road will be level 2 autonomous (or partial cruise control where the steering and acceleration/deceleration are automated). This replacement of traditional cars by autonomous cars will cause a growth in fleet financing which will decrease in auto loans and leasing. Also, as we all know that a learning system becomes more efficient with more miles driven, hence the autonomous cars will be more efficient than humans. Moreover, a fatal accident happens every 26 seconds [3] in the world, out of which 93.5% accident happen due to human error [11]. Thus, there is a need for automating the vehicles.

For AV, to drive in all possible conditions like a human, they come up with different type of sensors used for sensing the environment like, acoustic sensor, camera, radar, LIDAR, car2X communication,

*This work has been partially supported by IIT Bhubaneswar Seed Grant (SP093), and The Ministry of Human Resource Development, Government of India (SPARC P#701).

[1]Autonomous vehicle. *https* : *//en.wikipedia.org/wiki/Self* − *driving*$_c ar$
[2]Frost and sullivan survey on automotive vehicles. *https* : *//ww*2.*frost.com/files*/9114/3620/7521/$Sarwant_I M_d eck_V$10.*pdf*
[3]Fatality rate. *https* : *//www.asirt.org/safe* − *travel/road* − *safety* − *facts/*

sophisticated algorithms, and powerful processors to execute the software. There are hundreds of such sensors and actuators which are situated in various parts of the vehicle, being driven by a highly sophisticated system whose designing is a real challenge. Also, AV is a safety-critical system, i.e. the safety of the driver, passengers, pedestrian and other infrastructure present around is important. Therefore it becomes important that the AV should not malfunction at any given point of time. Moreover, there is increasing use of neural networks (NNs) and other Artificial Intelligence (AI) approaches in developing controllers for AVs. Convolutional Neural Networks (CNNs), are gaining importance in the development of AVs [4]. Thus, novel approaches are needed to thoroughly verify and validate the safety of AVs.

Formal methods and model driven development approaches are widely used for designing and developing safety-critical systems such as AV [2, 25]. Various formal verification techniques such as model checking[5] involve the formal modeling of computing systems and verification of policies on the models, including safety and timing policies. These model-based techniques are well suited for verifying complex safety-critical systems, because they guarantee absence of errors.

Model checking is an automated static verification approach to check that the formal specification of the system is satisfied by an abstract formal model of the system. Abstract model of the system is typically described as automata (or its extensions/variants). Policies are usually expressed in some form of Temporal Logic [21]. The abstract model and policies are taken as input by the model checking tools (e.g.[3, 9]) to answer whether the model satisfies the desired specification.

There are several challenges with using static formal verification approaches for systems such as AV developed using AI based approaches. For instance, creating a formal model of the system is an expensive task, where the system changes frequently (e.g., systems that use AI approaches changes frequently). Static verification of NN-based complex controllers is very challenging and time consuming [23].

Proposed work: There are several formal Runtime Verification (RV) and Runtime Enforcement (RE) monitor synthesis approaches proposed [1, 24, 17, 22, 10, 7, 16, 19, 20, 13] and the area of automatic synthesis of monitors from high-level specification that are correct by construction is actively under research. Such formal dynamic verification approaches are suitable for complex safety-critical systems where verifying the complete system statically is infeasible or difficult to achieve in practice. Runtime verification and enforcement techniques are lightweight and do not require a formal model of the system since only a single execution of the system is considered, hence are suitable for monitoring, verifying and enforcing critical properties of systems such as AV. There are also several tools and frameworks developed based on these proposed formal theories [12, 6, 13, 15, 18]. In this work we aim to illustrate the practical applicability of formally based runtime monitoring approaches and tools for systems such as AV. The proposed work develops a prototype autonomous car with real hardware to demonstrate how we can verify a safety-critical system using runtime monitoring and force the system to satisfy our policies at runtime. We also propose, in future work, that we can use these dynamic verification and enforcement approaches on NN based controllers too.

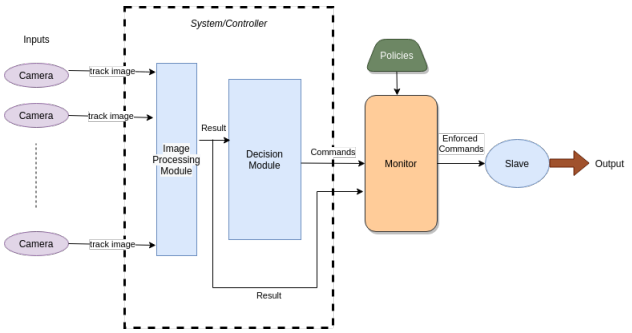

Figure 1: Proposed Model.

As illustrated in Figure 1 using formal runtime monitoring approaches and tools, verification (enforcement) monitors can be synthesized from the formal specification of desired policies, which can be integrated with the system. The monitor is fed with the input/output of the system (current execution), and it verifies the current execution w.r.t the desired policies, enforces (corrects) the current execution when dealing with enforcement of the desired safety policies at runtime.

2 Runtime Verification and Enforcement

Formal runtime verification and enforcement approaches deal with automatic synthesis of monitors from high-level specification of the policies (which the monitor should verify (or enforce)) [1, 24, 17, 22, 10, 7, 16, 19, 20, 13], that are correct-by-construction. For generation of monitors, knowledge of the system being monitored is not necessary (i.e., the system being monitored can be considered as a black-box). Monitors

are automatically synthesized from policies expressed using high-level formalisms such as automata [17, 22, 7, 16, 19, 20, 13] or some variant of Temporal Logic [1, 24].

A RV monitor does not modify the system execution, and is used to check the current execution of a system [1, 24, 17]. It takes a sequence of events from the system being monitored as input and produces verdicts that provide information whether the current execution satisfies the desired policy or not.

A monitor that deals with enforcement can be considered as a safety wrapper for the system being monitored. An Enforcement Monitor (EM) observes (and safe-guards/filters) the execution of a system to ensure that a set of desired policies are fulfilled. There are several EM synthesis frameworks proposed that vary in the supported policy specification language, and (or) the power of the enforcement mechanism. In [16, 8, 18] policies are expressed as automata and the EM is allowed to buffer input events until a future time when it could be forwarded. In [16] supports EM synthesis for real-time properties expressed as Timed Automata (TA). Other approaches such as [10] allow EM to modify input sequence by suppressing and (or) inserting events.

However, the above mentioned Runtime Enforcement (RE) approaches are not suitable for reactive systems such as AV. As pointed in [13], mechanisms such as edit automata [10], which focus on buffering or deleting events, are only suitable for transformational systems and not reactive systems as the latter one need to continually capture and emit events. Some of the recent works deal with synthesis of RE monitors for reactive and cyber-physical systems (CPS), which are more suitable for AV [19, 20].

In our work, we use the RE monitor synthesis tools that are based on the approaches proposed in [19, 20, 14] that are suitable for CPSs. The framework in [20] supports RE monitor synthesis for untimed polices, and [19, 14] extends it by supporting timed policies and parameters. In [14], Valued Discrete Timed Automata (VDTA) (where valued signals, internal variables, and complex guard conditions are supported, ensuring compatibility with real-world CPS and industrial systems) is used for formal specification of policies, and EMs are synthesized from these VDTA to enforce a set of desired policies.

3 Design of the Proposed System

Hardware Setup: For prototyping the AV, a Robot Car Chassis is needed consisting of 4 DC gear motors. A motor driver is used to control speed and direction of the DC motors. Raspberry pi and a microcontroller are used as master and slave device. Raspicam camera is used for sensing. The whole system is given power by a power bank with at least two output ports, and the connections are made using jumper wires. We do not go into further details of the setup due to space limitation. Figure 2 illustrates the setup/hardware model developed.

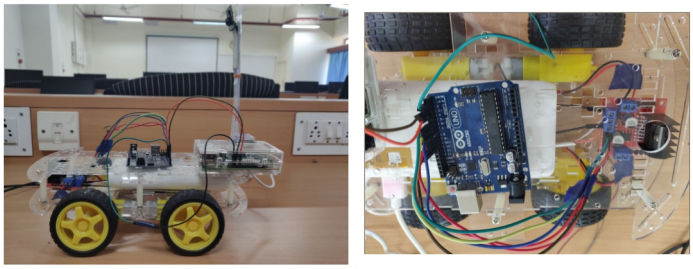

Figure 2: Hardware Setup.

Software Architecture: Various modules are needed for precise steering of the AV. The input module consists of Raspicam, which takes images of the track and forward it to image processing module.

The output of the AV module is sent to the decision module of the system. The resultant steering commands from the controller is sent to the output module, if no enforcement monitor is deployed into the system.

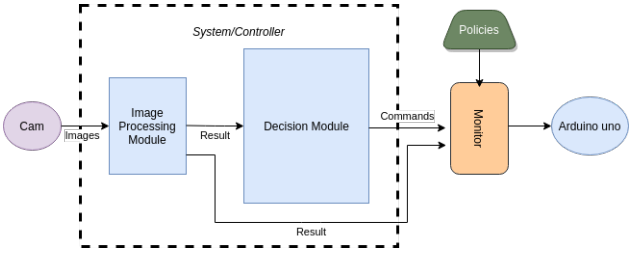

Figure 3: Design of AV with Monitor.

For applying runtime monitoring (to verify and enforce crucial properties) during execution, it is not necessary to have knowledge/ formal model of the system (i.e., the system/controller may be treated as a black-box). Figure 3 illustrates wrapping/safe-guarding the system/controller with monitors for desired safety policies. If we want to monitor and verify the input and output of the decision module of the controller at runtime, then a

monitor synthesized from the security policies (which the user specifies for the CPS) is deployed, which takes the input and output of the decision module and checks if these policies are obeyed by the current execution of the system or not (if not then enforce them).

The system is deployed on Raspberry pi and the output module is the micro-controller which when receives the output from the system, commands the motor driver to steer the AV.

4 Experimentation

For experimentation, Arduino Uno microcontroller is used for steering and L298N dual H-Bridge motor driver is used for controlling the DC motors. For steering the AV, track images are collected from the Raspicam (mounted on chassis) and is processed to extract the frame centre. Then the resultant difference (*Result*) between the frame centre and the lane centre is relayed to the decision module of the system (controller), which issues commands to the Arduino unit, to steer the vehicle.

We present an example policy related to the output of the decision module. We sampled the *Result* here, to define the safe behaviour of the AV. Let $V(Result)$ be the value of *Result*. We have:
- $-10 < V(Result) < 0$: the sensed input (value of result) is between -10 and 0, denoted as event W.
- $V(Result) == 0$: the sensed input is equal to 0, denoted as event X.
- $0 < V(Result) < 10$: the sensed input is between 0 and 10, denoted as event Y.
- others: the sensed input is anything other than above, denoted as event Z.

For synthesis of EM from policies, using approaches [20, 14], for example, consider the desired safety policy (on the output of the decision module of the controller) to be "*If the input is Z, then the vehicle should not move (forward/left/right).*" This policy is defined as automata over alphabet $\Sigma = \{W, X, Y, Z\}$ illustrated in Figure 4.

In the automaton in Figure 4, *stop* is the only non-accepting state. From any accepting state, upon receiving event W, the AV goes to left state; upon receiving event X, the AV goes to forward state; upon receiving event Y, the AV goes to right state; and upon receiving event Z, the AV goes to stop state. From all the states ($forward/left/right$), when the input received is Z, the vehicle moves to the *stop* state. When a situation arises where the policy can not be retained in an accepting state, the enforcer issues command to stop the vehicle.

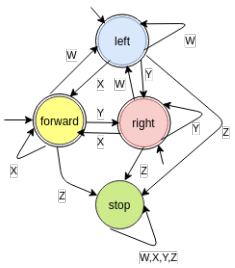

Figure 4: Automata for the desired policy.

The image processing module is a complex one which can be replaced with a NN based solution. So, the input fed to the decision module also has to be verified at runtime. We can define policies on input of the decision module; for example, *"For input to change from A ($A \in \Sigma$) to B ($B \in \Sigma$), there should be at least say n control steps"*.

We implemented a basic controller in C language. Regarding EM, the desired policies such as the policy described above are modelled as Finite Automata (FA) (alternatively as VDTA which supports valued channels and expressing timed constraints that will allow to model the desired properties more realistically without abstractions). From the policies defined as FA (VDTA), using the tools proposed in [20, 14], the monitor is synthesized automatically. The monitor (C code synthesized from high-level policies) integrated with the system takes the *Result* and *commands* from controller and checks if the policies are obeyed by the system or not (if not then enforce them).

5 Conclusion and Future Work

Autonomous vehicle is one of many trends likely to affect future transport demands. We built a working hardware prototype and demonstrated that formal runtime monitoring approaches can be used to wrap the controller of the AV to guarantee a safe behaviour. The prototype is built with Raspberry pi and Arduino Uno, in which the monitor is created out of the desired security policies, which will enforce the policies on the output of the controller, before relaying it to the slave Arduino micro-controller to steer the vehicle, thus, guaranteeing safe behaviour of the vehicle.

In future work, we plan to extend the set-up with multiple input sources, use more complex NN based controllers (e.g., in place of the current image processing module), and illustrate the practical applicability of dynamic formal monitoring approaches in more complex settings.

References

[1] A. Bauer, M. Leucker, and C. Schallhart. Runtime verification for LTL and TLTL. *ACM Trans. Softw. Eng. Methodol.*, 20(4):14:1–14:64, Sept. 2011.

[2] B. Beckert, T. Hoare, R. Hahnle, D. R. Smith, C. Green, S. Ranise, C. Tinelli, T. Ball, and S. K. Rajamani. Intelligent systems and formal methods in software engineering. *IEEE Intelligent Systems*, 21(6):71–81, 2006.

[3] J. Bengtsson, K. Larsen, F. Larsson, P. Pettersson, and W. Yi. Uppaal—a tool suite for automatic verification of real-time systems. In *International hybrid systems workshop*, pages 232–243. Springer, 1995.

[4] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016.

[5] E. M. Clarke Jr, O. Grumberg, D. Kroening, D. Peled, and H. Veith. *Model checking*. MIT press, 2018.

[6] C. Colombo, G. J. Pace, and G. Schneider. LARVA — safer monitoring of real-time Java programs (tool paper). In D. V. Hung and P. Krishnan, editors, *Proceedings of SEFM 2009*, pages 33–37, 2009.

[7] Y. Falcone. You should better enforce than verify. In *Runtime Verification - First International Conference, RV 2010, St. Julians, Malta, November 1-4, 2010. Proceedings*, volume 6418 of *Lecture Notes in Computer Science*, pages 89–105. Springer, 2010.

[8] Y. Falcone, L. Mounier, J.-C. Fernandez, and J.-L. Richier. Runtime enforcement monitors: composition, synthesis, and enforcement abilities. *Formal Methods in System Design*, 38(3):223–262, 2011.

[9] G. J. Holzmann. The model checker spin. *IEEE Trans. Softw. Eng.*, 23(5):279–295, May 1997.

[10] J. Ligatti, L. Bauer, and D. Walker. Edit automata: enforcement mechanisms for run-time security policies. *Int. J. Inf. Sec.*, 4(1-2):2–16, 2005.

[11] M. Maurer, J. C. Gerdes, B. Lenz, H. Winner, et al. Autonomous driving. *Berlin, Germany: Springer Berlin Heidelberg*, 10:978–3, 2016.

[12] D. Nickovic and O. Maler. AMT: a property-based monitoring tool for analog systems. In *Proceedings of the 5th International Conference on Formal modeling and analysis of timed systems (FORMATS 2007)*, volume 4763 of *Lecture Notes in Computer Science*, pages 304–319, 2007.

[13] H. Pearce, S. Pinisetty, P. S. Roop, M. M. Kuo, and A. Ukil. Smart i/o modules for mitigating cyber-physical attacks on industrial control systems. *IEEE Transactions on Industrial Informatics*, 2019.

[14] H. A. Pearce, S. Pinisetty, P. S. Roop, M. M. Y. Kuo, and A. Ukil. Smart I/O modules for mitigating cyber-physical attacks on industrial control systems. *IEEE Trans. Ind. Informatics*, 16(7):4659–4669, 2020.

[15] S. Pinisetty, Y. Falcone, T. Jéron, and H. Marchand. Tipex: A tool chain for timed property enforcement during execution. In *Runtime Verification - 6th International Conference, RV 2015 Vienna, Austria, September 22-25, 2015. Proceedings*, volume 9333 of *LNCS*, pages 306–320. Springer, 2015.

[16] S. Pinisetty, Y. Falcone, T. Jéron, H. Marchand, A. Rollet, and O. Nguena-Timo. Runtime enforcement of timed properties revisited. *Formal Methods Syst. Des.*, 45(3):381–422, 2014.

[17] S. Pinisetty, T. Jéron, S. Tripakis, Y. Falcone, H. Marchand, and V. Preoteasa. Predictive runtime verification of timed properties. *J. Syst. Softw.*, 132:353–365, 2017.

[18] S. Pinisetty, V. Preoteasa, S. Tripakis, T. Jéron, Y. Falcone, and H. Marchand. Predictive runtime enforcement. *Formal Methods Syst. Des.*, 51(1):154–199, 2017.

[19] S. Pinisetty, P. S. Roop, S. Smyth, N. Allen, S. Tripakis, and R. von Hanxleden. Runtime enforcement of cyber-physical systems. *ACM Trans. Embedded Comput. Syst.*, 16(5s):178:1–178:25, 2017.

[20] S. Pinisetty, P. S. Roop, S. Smyth, S. Tripakis, and R. von Hanxleden. Runtime enforcement of reactive systems using synchronous enforcers. In *Proceedings of the 24th ACM SIGSOFT International SPIN Symposium, Santa Barbara, CA, USA, July 10-14, 2017*, pages 80–89. ACM, 2017.

[21] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, SFCS '77, pages 46–57, Washington, DC, USA, 1977. IEEE Computer Society.

[22] F. B. Schneider. Enforceable security policies. *ACM Trans. Inf. Syst. Secur.*, 3(1):30–50, Feb. 2000.

[23] S. A. Seshia and D. Sadigh. Towards verified artificial intelligence. *CoRR*, abs/1606.08514, 2016.

[24] P. Thati and G. Rosu. Monitoring algorithms for metric temporal logic specifications. *Electronic Notes in Theoretical Computer Science*, 113:145–162, 2005.

[25] J. Woodcock, P. G. Larsen, J. Bicarregui, and J. Fitzgerald. Formal methods: Practice and experience. *ACM Comput. Surv.*, 41(4), Oct. 2009.