CTMC Analysis of Level-3 Autonomous Driving System with PRISM

Christopher M. Champion
1719247
Department of Computer Science and Software Engineering
Xi'an Jiaotong Liverpool University
Suzhou, Jiangsu, China
C.Champion17@student.xjtlu.edu.cn

Jun Woo Kim
1614645
Department of Computer Science and Software Engineering
Xi'an Jiaotong Liverpool University
Suzhou, Jiangsu, P.R. China
Junwoo.Kim16@student.xjtlu.edu.cn

Yifu Qian
1613161
Department of Computer Science and Software Engineering
Xi'an Jiaotong Liverpool University
Suzhou, Jiangsu, P.R. China
Yifu.Qian16@student.xjtlu.edu.cn

***Abstract*—The recent increase in available autonomous driving assistants in non-commercial passenger vehicles has raised safety concerns for automotive manufacturers and customers. Manufacturers face increasing pressure to bring such systems to the consumer market and have a limited amount of time to test them under real world conditions. Modeling a self-driving embedded system as a continuous-time Markov chain (CTMC) and applying probabilistic model checking can help determine system faults using a broader range of conditions when compared to real world testing. This analysis of a Level-3 autonomous driving assistant uses PRISM modeling software to check the probability of system failure before and after the inclusion of fault-tolerant sensors.**

Keywords—autonomous, self-driving, audi, ctmc, markov, prism, sensor, probability

I. Introduction

Autonomous driving systems are categorized into six levels of autonomy, with Level-0 offering no automation and Level-5 reaching total autonomy with no human intervention required [1]. Self-driving systems are currently being developed at a rapid pace, not only by automotive manufacturers like Tesla, but also by companies in the IT services industry, such as Google and Uber. Most of the currently available systems are classified as Level-2 automation, or "partial automation". Our CTMC is modeled after Audi AG's "Traffic Jam Pilot" assistant, the world's first Level-3, or "conditional automation" [1] system to be made available to consumers. With safety being a major concern, a model checking system can be used in addition to real world testing to determine faults in proprietary self-driving systems.

In order to reach the level of accuracy employed in autonomous vehicles, a complex system that controls dozens of sensors, cameras, and CPUs is needed. This is commonly known as an embedded system, or a "microprocessor-based system that is built to control a function or a range of functions. [2]" It is embedded as part of a complete device often including hardware and mechanical parts. Modern embedded systems are often based on microcontrollers and multiple sensors as inputs (i.e. CPUs with integrated memory or peripheral interfaces). Thus, it is reasonable to consider the system in the self-driving vehicle as an embedded system.

Continuous-time Markov chains (CTMC) refers to the sequence of random variables that a process moves through defined by the Markov property serial dependence only between adjacent periods (as in a "chain") [3]. CTMC is suitable for representing labeled transition systems augmented with rates such as embedded systems and test the reliability of those systems.

This paper introduces a PRISM model based on CTMC and the embedded system of Audi's self-driving vehicle and tests the model's failure rate.

II. Audi AI Traffic Jam Pilot

Audi's traffic jam pilot is the world's first Level-3 automated driving system to be made available to consumers [1]. It relies on a multitude of software and hardware systems to provide "conditional automation", which means the car can drive autonomously without input from the driver when specific conditions are met.

A. Autonomous Driving Conditions

- The system can only be activated on highways with oncoming traffic lanes separated by a barrier and a guardrail on one side.
- As its name suggests, the system is intended only for use in heavy "bumper-to-bumper" traffic situations.
- Speed cannot exceed 60 km/h.
- The system does not operate in the presence of traffic lights or pedestrians.

B. Hardware Implementation

- 12 ultrasonic sensors surrounding the vehicle.
- 4 360-degree cameras on all sides of the vehicle.
- 1 camera at the front of the vehicle.
- 4 mid-range radar sensors at each corner of the vehicle.
- 1 long-range radar sensor at the front of the vehicle.

- 1 laser scanner (LIDAR) sensor at the front of the vehicle.
- 1 interior camera for observing the driver.
- 2 "zFAS" controller units for creating a localization model of the car's surroundings based on sensor input.

III. PRISM Model

To translate Audis' traffic jam pilot automated system into a PRISM model, we created eight modules: seven for each of the sensor component types and a module for the redundant central driver assistance controller (zFAS) that uses input data from the sensors to generate a model of the car's surrounding environment [1]. Our initial model was based on non-fault tolerant (NFT) sensor nodes, with sensor modules only containing two states: state 1 being "OK" and state 0 being "failed" [4].

A. *Initial Model*

The system utilizes multiple sensors for some of the sensor types, e.g. twelve ultrasonic sensors are positioned around the vehicle. For these modules, we defined a minimum number of sensors that must be in state 1 "OK" for the system to continue to operate. Due to the zFAS controller's redundant design, we require at least one of the two available controllers to be in state 1 for the system to operate. The sensor modules were tested with a failure rate of lambda_s (=1/(30*24*60*60)) and the zFAS controller with a failure rate of lambda_a (=1/(2*30*24*60*60)), assuming the sensors have a higher failure rate than the controller. Upon failure, the PRISM model reduces the number of working sensors (or zFAS controllers) by one.

Out of the eight types of components used by the system, four are not redundant, meaning the whole system will shut down if any one those components fail. These modules were tested with a failure rate of delta_f (=1/(24*60*60)).

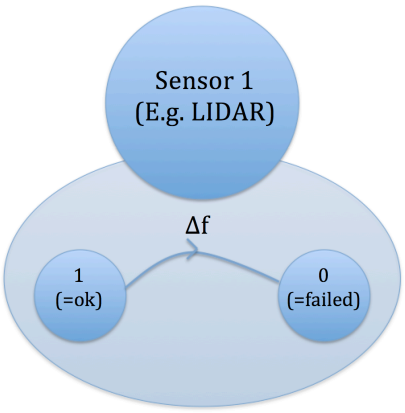

Fig. 1. NFT sensor node Markov model with delfa_f probability of transitioning to failed state

B. *Initial Model Testing*

To test the model, we defined a constant time variable "T" hours and created a conditional property for each of the modules. The modules with multiple sensors are checked against a constant "MIN" variable, e.g. the ultrasonic sensor system is considered "down" if less than 10 (=MIN_US_SENSOR) ultrasonic sensors are working. Modules with individual components are considered "down" if their state is 0. If any one module is considered "down", the whole system is down.

C. *Initial Model Test Results*

Testing system components with a given T value of 48 hours revealed that NFT sensor nodes, especially for modules with only one component, have a significantly high failure rate.

Property:
P=? [!"down" U<=T*3600 "fc_down"]

Defined constants:
T=48

Method:
Verification

Result (probability):
0.6831388616162891 (value in the initial state)

1 warning (see log)

Fig. 2. Probability of NFT front camera module failure after 48 hours

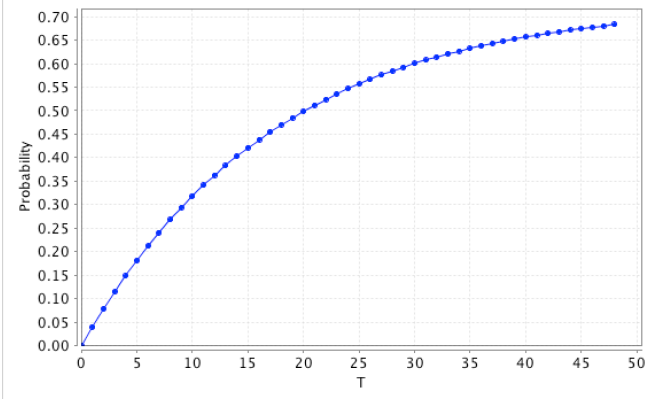

Fig. 3. Graph representation of front camera module failure after 48 hours

This discovery led us to reevaluate the NFT-basis after which these modules were modeled.

D. Revised Model

We decided to revise the modules consisting of only one system component by introducing a fault-tolerant (FT) duplex sensor node model. This model includes a third "transient fault" state and an inactive sensor that becomes active when a transient fault occurs [4], i.e. state is 1. Since most faults that occur in a system are transient and non-persistent [5], if a transient fault occurs, the module is restarted and given a high probability of returning to its normal working state.

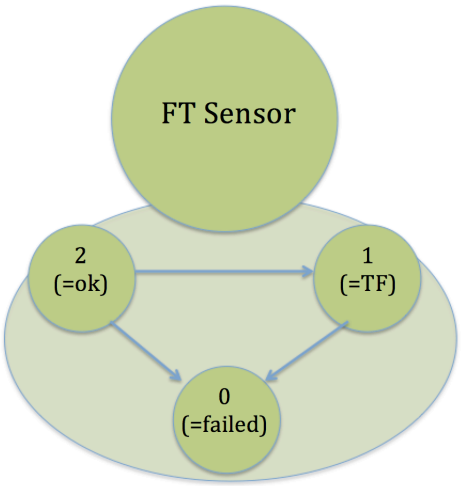

Fig. 4. Fault-tolerant sensor node Markov model

E. Revised Model Testing

By introducing fault-tolerance to the model with an additional transient fault state, the probability of failure significantly dropped when compared to the original NFT model. The front camera module was tested with an 85% probability of returning to its normal state of 2 (=OK) if a fault occurs. The other three modules were given a probability of 90%, 80%, and 85% for the long-range radar, LIDAR, and observation camera modules respectively. The LIDAR module was given the highest probability of 20% of transitioning to a failed state (=0) due to the affects of weather conditions on LIDAR-based systems [6].

Property:
P=? [!"down" U<=T*3600 "fc_down"]

Defined constants:
T=48

Method:
Verification

Result (probability):
0.18588620868623823 (value in the initial state)

Fig. 5. Probability of failure-tolerant front camera module after 48 hours

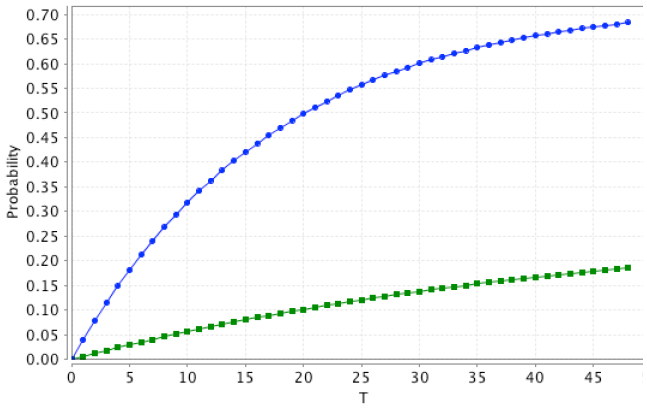

Fig. 6. Graph representation of front camera module before and after introducing transient fault

F. Revised Model Test Results

Probability testing of the four single component modules with and without fault-tolerance showed a significant reduction in failure probability rates within 48 hours after the inclusion of fault-tolerant sensors.

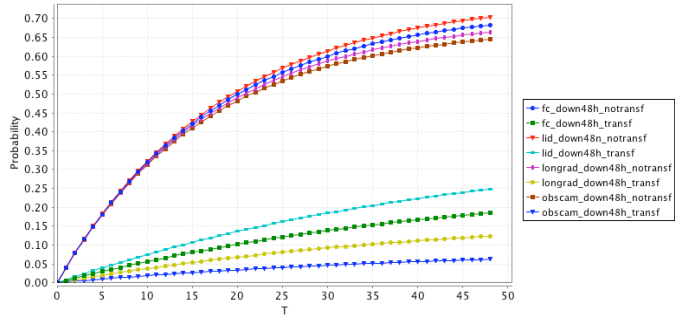

Fig. 7. Graph representation of non-FT and FT sensor modules

IV. Conclusion

In this paper, we have introduced two PRISM models, modeled after Audi's Level-3 autonomous driving system. One model includes transient faults for the front camera, long-range radar, LIDAR, and observation camera as variables within the system. When those sensors are facing transient faults, the system reboots. The original model does not include transient faults. The experiment has been conducted to evaluate the reliability of the two models, including the influence of sensors and controllers and the comparison between components with transient faults and those without transient faults.

Based on the experiment's results, it is shown that system reliability has a deep correlation to the inclusion of fault-tolerant sensors. Because handling a transient fault offers the module a chance to reboot, the failure rates of fault-tolerant modules rise more gently than modules without transient faults. In general, although incorporating fault tolerance into an embedded system increases programming resources and total system costs, the improvement in

component reliability translates to a system that provides much higher levels of safety to passengers and pedestrians.

REFERENCES

[1] "TechDay piloted driving – The traffic jam pilot in the new Audi A8," Audi MediaCenter. August 08, 2017, pp. 3–8.

[2] S. Heath, "Embedded systems design. EDN series for design engineers (2 ed.)". Newnes., 2003, p. 2. ISBN 978-0-7506-5546-0.

[3] P. A. Gagniuc, "Markov Chains: From Theory to Implementation and Experimentation", USA, NJ: John Wiley & Sons., 2017, pp. 1–256 ISBN 978-1-119-38755-8.

[4] A. Munir, A. Gordon-Ross, S. Ranka, "Modeling and Optimization of Parallel and Distributed Embedded Systems", John Wiley & Sons Ltd, 2016, pp. 79–80.

[5] P. Koopman, "System Resets; Robustness; Power Management," Carnegie Mellon University, April 18, 2016, Slide 27 https://www.ece.cmu.edu/~ece348/lectures/25_booting_robustness_handouts.pdf.

[6] R. H. Rasshofer, M. Spies, H. Spies, "Influences of weather phenomena on automotive laser radar systems," Advances in Radio Science, vol. 9, Copernicus Publications, 2011, pp. 49–60.