Impact of Machine Learning on Safety Monitors

Francesco Terrosi[1](✉), Lorenzo Strigini[2], and Andrea Bondavalli[1]

[1] Università degli Studi di Firenze, Firenze, Italy
{francesco.terrosi,andrea.bondavalli}@unifi.it
[2] City, University of London, London, UK
strigini@csr.city.ac.uk

Abstract. Machine Learning components in safety-critical applications can perform some complex tasks that would be unfeasible otherwise. However, they are also a weak point concerning safety assurance. An aspect requiring study is how the interactions between machine-learning components and other non-ML components evolve with training of the former. It is theoretically possible that learning by Neural Networks may reduce the effectiveness of error checkers or safety monitors, creating a major complication for safety assurance. We present an initial exploration of this problem focused on automated driving, where machine learning is heavily used. We simulated operational testing of a standard vehicle architecture, where a machine learning-based Controller is responsible for driving the vehicle and a separate Safety Monitor is provided to detect hazardous situations and trigger emergency action to avoid accidents. Among the results, we observed that indeed improving the Controller could make the Safety Monitor less effective; it is even possible for a training increment to make the Controller's own behaviour safer but the vehicle's less safe. We discuss implications for practice and for research.

Keywords: Safety · Autonomous vehicles · Automotive · Machine-learning

1 Introduction

Machine Learning (ML) is bringing great changes in many embedded computing applications. In many applications, Neural Networks (NNs) generalize well from situations encountered during training to those it will encounter during subsequent testing and, with luck, to those it will encounter during operation. However, neural networks also represent a weak point from the viewpoint of safety assurance. The lack of an explicit design derived from a specification undermines the very basis of established verification activities for critical systems: verifying with confidence that the implementation satisfies its specifications, and the specified safety properties. An additional concern is that established practice requires a safety-critical system to change as little as possible, and changes to be clearly documented, to support verification towards their acceptance. Machine learning, by contrast, encourages a development culture in which frequent change (additional "learning") is accepted and, due to the nature of ML, there is no documentation of the changes that could directly support verification. Manufacturers of

M. Trapp et al. (Eds.): SAFECOMP 2022, LNCS 13414, pp. 129–143, 2022.
https://doi.org/10.1007/978-3-031-14835-4_9

autonomous vehicles are known to collect data from their fleets of vehicles under test, and even in commercial operation, to incrementally train and improve the ML "driver" [1–3]. Last but not least, some self-driving vehicles must satisfy extreme safety requirements (accident rates comparable to, or substantially better than, those of human drivers), such that simple statistical demonstration of their satisfaction through road resting is not feasible [9–12].

Given that we cannot trust these control systems ("Controllers", for brevity) to be safe enough, it is natural to apply independent safety subsystems ("Safety Monitors" SMs, hereafter) that can detect hazardous situations, e.g., approaching collisions, and command remedial actions such as braking, as an additional line of defense [4–7].

Ideally, a safety monitor is much simpler than a Controller, so that, once verified, it gives strong confidence that it will perform to the level of reliability (and hence of vehicle safety) that has been assessed. This may seem to offer a solution for the assurance problem: aim for strong confidence in the safety system even if there will be uncertainties on the safety of the Controller by itself. Although a real safety monitor does not have 100% coverage (probability of detecting and mitigating a hazard situation, conditional on its arising), the coverage could be assessed by extensive simulation testing. The goal is a high enough coverage value that if one multiplies (1-coverage) times the estimate of the rate at which the Controller allows hazardous situations to arise, the result is a low enough rate of accidents. Even if the Controller is frequently changed, this form of reasoning will remain valid. Estimating the two multiplicands separately through testing would require substantially less testing than estimating the rate of accidents directly.

This solution to the assessment difficulties is – however – illusory. The coverage of the Safety Monitor depends on the Controller that it monitors [8]. It is possible that, as a vehicle's Controller improves, and even if this improvement includes its safety (i.e., if without the help of the Safety Monitor each new version would cause fewer accidents than the previous one), the coverage of the Safety Monitor becomes worse, because the fewer hazard situations allowed by the Controller are increasingly of kinds with which the Safety Monitor cannot cope. So, the *whole system* must be tested enough to demonstrate that the rate of accidents would not exceed the required bound. In theory the coverage may decrease so much that *improving* the Controller makes the vehicle as a whole *less safe*. It would be very desirable to have a strong argument that this will not happen [9], since this would support a sound and simple form of safety argument based on operational testing of the vehicle.

A first step to study this possibility is the empirical study that we present here, to answer these research questions:

1. Can one observe in practice these "unwelcome surprises" in which improving a Controller reduces the monitor's coverage, or even increases the vehicle's accident rate?
2. If so, can we derive insights on what factors in the Controller's training, the operating environment or the safety subsystem's design contribute to such surprises?

Our study applies these questions to a primitive simulated vehicle and its environment. The goal of this paper is to share with the community i) *the methodology*, so that

it can be used and improved, ii) *a proof of existence* of the "unwelcome surprises", and iii) initial insights on what contributes to them.

2 Related Work

Research on machine learning techniques in many diverse applications, some of them safety-critical, has proliferated in recent years [26–30]. Concerns about machine learning in safety-critical systems have led to research to develop techniques for safety and/or explainability of ML components [32–34]. A common approach in safety-critical systems is to pair the main system Controller, which may use ML, with a Safety Monitor which may be a human or, more commonly, a dedicated hardware-software subsystem [13, 20, 31, 33]. Another approach, hardening and verifying the safety properties of neural networks by developing new training algorithms and network architectures, has proved effective in some studies [35, 38]. Unfortunately, improving ML components is not enough by itself to prove valid safety arguments for such systems [19, 36]. Thus, effort is also applied on how to provide sound and reasonable safety arguments of such systems. These research efforts aim at improving the explainability of the decisions of the ML components and at designing and providing guidelines for safety/assurance cases [19, 33, 34, 36–38].

In this rich research corpus, however, we found no studies of our topic, i.e., how improving ML components affects the efficacy of Safety Monitors that monitor them.

3 Problem Statement

Verifying that "ultra-high" dependability requirements are satisfied is known to be a hard problem [9–12] and the use of ML makes it even harder. The challenge of assuring the safety properties of autonomous vehicles is, as of now, one of the main concerns delaying their deployment [13, 14]: because it is hard to collect enough evidence to prove that one system is "safe enough", and because it is difficult to understand the inner process that made a neural network take a specific decision [15]. Simulation proved effective for training a neural network to drive, and it is one of the first steps in the development of automated, unmanned vehicles [16–18]. However, testing an autonomous vehicle is a hard task even with the aid of a simulated environment because i) neural networks cannot generalize their function to every possible event, ii) it is not possible to test every possible event and iii) designing an end-to-end design and deployment process for such complex systems is hard [19].

In this work we are interested in studying the effects of "additional learning" of a Controller on the coverage of the Safety Monitor (probability of detecting a hazard situation, conditional on its arising: true positive rate of the hazard detection – and mitigation – function). Since the probability of the SM preventing an accident depends on the relative frequencies with which the Controller generates various types of demands on the SM (hazardous situations for which coverage is high vs those for which coverage is low) [8], the Controller may significantly change these probabilities as it "learns". So, *every change* in the Controller will invalidate the coverage estimate and thus any safety argument that assumes i) unchanging coverage of the Safety Monitor or even just ii) that more learning by the Controller implies improving system safety.

4 System Model and Terminology

Here we describe the system model and the terminology used and define and discuss the metrics used to measure the performance of the Safety Monitor when applied to a learning Controller. We simulated the architecture depicted in Fig. 1, where an end-to-end learning Controller is paired with a Safety Monitor to make the car move safely.

The *Controller* is the main component of the system. Its task is to drive the car from a starting position to a destination, obeying traffic laws and other internal rules such as ensuring a "smooth" ride or acceptable fuel consumption. A Controller is often built as a set of specialized modules which implement the required functions of perception, planning, etc. This allows run-time monitoring of the operation of each module. We used a simpler, monolithic design: the whole process from perception to motion control is encoded into a single deep learning architecture (end-to-end learning [20]). The Controller is thus a "black box": the Safety Monitor can only react to hazardous actions of the Controller, not to internal errors of the Controller that might lead to such actions.

Our *Safety Monitor* uses data from other sensors (a LiDAR) than those used by the Controller, as recommended by good practice, to sense objects and obstacles near the car. If the action of the Controller would cause a safety hazard (i.e., potential for a crash: e.g., not braking when crossing the minimum safe distance from an obstacle in front), the SM triggers emergency braking.

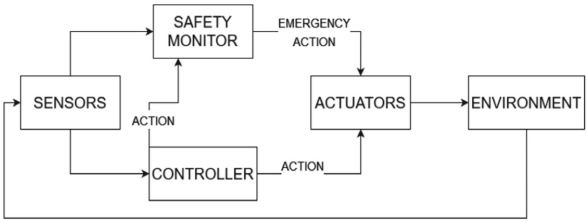

Fig. 1. System architecture

4.1 Terminology

Neural networks can be trained over long periods, using multiple data sets to improve their performance. Their evolution is described by the changes in their internal parameters, i.e., weights of the prediction function. We define a *checkpoint* as the set of weights of the NN's function after a series of training steps. We say: $\text{checkpoint}_i < \text{checkpoint}_j$ if checkpoint j is obtained from checkpoint i after a number of training steps. We define C_i as "the Controller obtained at checkpoint i" and will refer to it just as "Controller" when the level of training is irrelevant. Note that $j > i$ only means that C_j had more training than C_i, not necessarily that it performs better.

The Controller's task is to drive the car efficiently and safely, while obeying traffic laws. In practice in our simulation, since the car's training was stopped at a comparatively immature stage, we allowed all simulated trips to continue until a crash occurred. Since we are only interested, at this stage of the work, in safety, we define a failure of the Controller as:

"Any action taken by the Controller that would result in a crash", i.e., the output of the Controller will trigger a transition from inside to outside the space of safe states. The safety performance of the Controller can be evaluated as a rate of accidents per km, or per unit of time in operation, or per trip.

Whenever the Controller fails, the SM has to detect this situation and intervene as soon as possible to prevent the imminent crash. The SM may respond correctly, which will in some cases avert the accident and lead the system to a safe state. Obviously, the SM can fail as well, in one of these two ways:

- It does not detect the problem (obstacle).
- It detects the obstacle and takes action, but the car still crashes.

The Safety Monitor can thus be seen as an "extended binary classifier" that classifies the system's state as safe or unsafe, based on the Controller's actions and sensor data, and takes action accordingly. Its performance can be described via a matrix, akin to the Confusion Matrix for a classifier, but related to results of actions (e.g., success or failure of a safety intervention) rather than just classification decisions.

4.2 Description of the State Space

We divide the state space of the system (Controller plus Safety Monitor) into:

- *Safe States*: all the states in which the Controller does not need the intervention of the Safety Monitor, and the Monitor does not intervene.
- *Mitigation States*, in which the Controller behavior would lead to a system failure (accident), but the Monitor correctly prevents the crash.
- *False Alert States*: the states in which the Controller does not need the intervention of the Safety Monitor, but the Monitor wrongly intervenes.
- *Accident States*: all the states in which the Controller's behavior leads to a crash which *are not* solved by the Monitor.

The actions of the Controller cause transitions between the system states. Figure 2 is a Venn diagram representing the events "transitions from the safe state". Areas represent event probabilities, determined by the system and its environment, and which will normally change if the system components change (e.g., through machine learning). For reasonably safe systems, transitions to *safe states* and *mitigation states* will be much more likely than the others. For good availability, performance, comfort, transitions to *false alert states* should also be rare.

Any further training of the Controller will change its behavior and thus the probabilities associated to each transition. If the probabilities of transitions to both *safe states* and *mitigation states* increase, system safety improves. However, it is also possible that, even if the Controller learned to drive very safely (i.e., the probability of transitions to *safe states* gets very large), transitions to *accident states* also become more frequent, at the expenses of transitions to the *mitigation states*. These two possible effects of training *that makes the Controller safer* are shown in Fig. 3. Starting from the diagram in Fig. 2, additional training may produce, among others, either one of these two Venn diagrams.

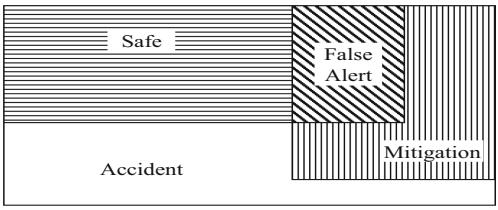

Fig. 2. "Safe" + "False Alerts" indicates probability of the Controller continuing safe operation; the squarish rectangle on the right, "False Alerts" + "Mitigation", represents the SM's interventions. The remaining white area rep resents initiation of accidents.

5 Study Method

To test the Controller at different stages of its training, we generated m checkpoints, resulting in m Controllers $C_1 \ldots C_m$. We tested all these on the same predefined set of scenarios, to observe how well the ML component handles the same task (i.e., reaching a target destination, via specified waypoints, given a starting position, in the same environmental conditions) at different stages of its training. A "scenario" is defined by the initial conditions of the environment in which the system is tested. This includes the starting point, seeds for random number generators, a target destination and intermediate waypoints, and environmental conditions such as weather and traffic density. Scenarios can be made more difficult by manipulating conditions, e.g., by increasing the traffic present in the environment or by simulating adverse weather. We call the difficulty levels h_0, h_1, etc. A higher subscript represents greater difficulty: if $x < y$, h_y is designed to be harder than h_x. We note that a level that is harder for the Controller may not be harder for the SM monitoring *that* Controller in *that* environment.

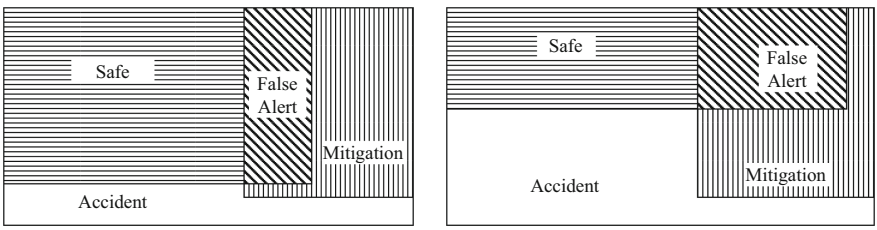

Fig. 3. Examples of training that improves the Controller's safety shown in Fig. 2. In case a) this improvement reduces hazards that the SM could not mitigate; in case b) it reduces hazards that the SM would mitigate, while adding some that the SM cannot mitigate.

5.1 Paired Tests with and Without Safety Monitor

The Controllers were first tested without the Safety Monitor in every scenario, until the car reached the target destination, or crashed. Our setup also allows a test to be stopped earlier, but we did not use this option.

For every such trip, we recorded the initial conditions and the sequence of actions chosen by the Controller; then "replayed" the run exactly, but activating the SM, to observe whether it intervenes correctly to interrupt the specific accident sequences that ended SM-less runs.

This setup allows one not only to observe "how good" the SM is in preventing failures, but also, in some cases, to understand which situations are difficult for the Controller, and which ones are difficult for the Safety Monitor.

In more detail: in the run with the Safety Monitor, we record all the alerts it raises in each simulation step, but with safety braking *disabled* until it becomes necessary to prevent the collision that ended a specific SM-less run. To this end, we computed by what earlier time t the hazard must be detected so that braking may prevent the accident. We assumed that any alert raised by the Safety Monitor before time t is not necessary and thus a *false alarm.* After time t, that is, during the series of simulation frames that directly resulted in a crash, we enable emergency braking by the SM. If the imminent collision is avoided, we terminate the run and log a successful SM intervention.

These precautions are needed because if we simply re-ran each SM-less run, from the same initial conditions but with the SM active, the sequence of events that led to a crash might not happen again: e.g., a false alert by the SM could slow down the car so that it would not encounter the same hazard.

In the present study, we enabled emergency braking 2 seconds before the accident happened; this interval was chosen based on the maximum speed (50 km/h) the car can reach, the reaction time needed to respond to the hazard, and the distance required for braking. In the last 2 seconds of the simulation, an alert raised by the SM will now effectively make the car brake.

We note that with this setup our test of the Safety Monitor will omit some events of potential interest: in reality, false alarms may cause accidents, e.g., if hard braking causes the vehicle to be hit from behind. This risk complicates the task of specifying safety monitors. This potential for the SM to cause accidents is also one way that improving the Controller may make the vehicle less safe, e.g., if the Controller learns "bold" maneuvers that it would complete safely but that prompt a SM to apply potentially risky emergency actions. We left the simulation of these more complex effects to future research; the focus of this study was to demonstrate subtle problems in safety arguments even with a safety monitor whose interventions are always beneficial.

5.2 Evaluation of the Components and the System (Vehicle) Safety

We define the event "a crash would occur without the SM" as "C_crash" (Controller crash). We define classes of correct and wrong actions of the SM as follows:

- *Successful Intervention (SI).* Every crash prevented by the SM: the safety response of the SM triggers a transition from a *safe* state to a *mitigation* state.
- *False Alarm (FA).* Each alert raised by the SM when the system is in a *safe state.*
- *True Negative (TN).* The system is in a *safe state* and the SM does not raise an alert.
- *Crash (CR).* Every crash not prevented by the Safety Monitor, i.e., there is a transition from a *safe* state to an *accident* state.

From the recorded counts of these events, we derived safety measures of interest. First, we computed the Coverage (COV) of the SM, the ratio between the number of crashes avoided by the SM and the number of crashes that the Controller would cause if the SM were not present, that is:

$$\text{COV} = \frac{number\ of\ SIs}{number\ of\ C\_crashes} \tag{1}$$

We also compare the rate of occurrence of accidents per kilometer caused by the Controller without a Safety Monitor:

$$\text{P(C\_crash)} = \frac{number\ of\ C\_crashes}{kilometers\ driven} \tag{2}$$

with the rate when the SM is active:

$$\text{P(crash)} = \frac{number\ of\ Crashes}{kilometers\ driven} \tag{3}$$

We also measure (but did not analyze in detail) the False Alarm Rate:

$$\text{FAR} = \frac{number\ of\ FAs}{number\ of\ FAs\ +\ number\ of\ TNs} \tag{4}$$

The SM may raise a false alarm at any time during a simulation run, while the Coverage is measured on the number of crashes, which happen once per run at most.

These measures are sufficient for answering the immediate questions of this study. This simulation setup allows one also to assess, for instance, Mean Distance Between Accidents, Mean Time Between Accidents and Reliability Functions related to accidents and False Alarms.

Another study of interest would consider the severity of accidents. For example, a crash at 10 km/h against a fence may be flagged as a less serious failure than hitting a group of pedestrians at 50 km/h. These data can be used to observe correlations between failure modes and difficulty levels that may be counterintuitive, such as a Controller that crashes more frequently with *vehicles* when the number of *pedestrians* is increased.

6 Details of the Simulation

6.1 CARLA Simulator

We used CARLA 0.8.4 [21], an open-source simulator, sponsored by Intel and Toyota among others. It provides a realistic urban environment and was developed specifically to train and test autonomous vehicles controlled by ML components. It allows full customization and control over vehicles, pedestrians, weather, and sensors. In this version of CARLA there are four sensors:

- *Scene Final Camera:* provides a view of the scene like that produced by ordinary cameras

- *Depth Map Camera:* provides a depth mapping of the objects in the environment.
- *Semantic Segmentation Camera:* it paints object pertaining to different classes (e.g., vehicles and pedestrians) with different colors.
- *LiDAR sensor:* Light Detection and Ranging creating a 3D map of the surroundings.

The Depth Map Camera and the Semantic Segmentation Camera provide ground truth values for depth mapping and object classification. The ray-cast based LiDAR provided by CARLA was tuned to simulate a slightly modified version of the HDL-64E Velodyne LiDAR. The modifications were necessary because of the computational cost required to simulate a real LiDAR.

6.2 Implementation of the Controller and Safety Monitor

The Controller was implemented using the implementation of the Deep Deterministic Policy Gradient (DDPG) algorithm [22], provided by Coach, a framework for reinforcement learning developed by Intel's AI Labs [23]. The DDPG algorithm was chosen because it is specifically designed for environments with a continuous action space, such as the one we study, and it proved to perform well in driving tasks.

The Safety Monitor, implemented using the Point Cloud Library [24], is based in part on E. Bozkurt's project "Lidar Obstacle Detection", available on GitHub [25]. It implements a safety braking function using non-ML processing of data from the LiDAR sensor to map the environment. Using two consecutive measurements, it can track objects in the environment and estimate the relative speed of objects in front of the car. Thus, it is possible to implement a safety routine based on the braking distance between the car and the object detected, and their relative speed. To test the efficacy of the whole safety routine (not only the ability of the SM to raise an alert) the runs previously recorded without the SM are repeated with it, rather than just replaying the LiDAR data from them to the Safety Monitor to record the alerts raised.

6.3 Structure of the Study

We collected 5 checkpoints from the training activity: Controllers C_1 to C_5. CARLA offers 150 predefined locations in the city. For each one of these, we created a trip specification that started from it and had to travel through a randomly selected sequence of 15 other locations (the latest one being the destination of that trip). Each trip specification was then combined with 4 different traffic conditions, or "difficulty levels", h_0, h_1, h_2, h_3, to vary the difficulty of the Controller's task:

h0) *Default*: the map is generated with 30 pedestrians and 15 vehicles.
h1) *Pedestrians:* the number of pedestrians in the map is doubled.
h2) *Vehicles:* the number of vehicles in the map is twice that in h0.
h3) *Pedestrians and Vehicles:* both pedestrians and vehicles are twice as many as in h0.

From each combination of trip specification and difficulty levels we created 4 scenarios by applying different Random Number Generator seeds in CARLA. We thus had $4 \times 150 \times 4 = 2400$ test scenarios, on which each Controller C_i was tested with and

without the Safety Monitor. A SM-less run ends when a collision happens, or the car reaches its destination (passing by the intermediate waypoints). The paired run with SM is ended at the same point, as explained in Sect. 5.1.

The simulation runs at a fixed time step of 10 Frames Per Second, so the number of simulation steps created per second of simulated time is an invariant. This avoids potential accuracy problems with timing and measurements and gives a reference time-base to compute time-dependent metrics.

7 Results of the Simulation

7.1 Controller

Table 1 shows the rates of occurrence of crashes of Controllers C_1 to C_5, operating, *without* the SM, at the four levels of environment difficulty.

One sees that there is safety improvement from C_1 to C_5: e.g., the rate at difficulty h_0 improved from 0.95 for C_1 to 0.29 for C_5, although the improvement is non-monotonic (e.g., C_3 is less safe than C_2). Moreover, the way we manipulated difficulty from h_0 to h_3 appears effective: it actually makes the environment more difficult for the Controller, as $P(C\_crash)_{hi} < P(C\_crash)_{hj}$ if $j > i$, for all Controllers (except for C_2 performing slightly better in h_1 than in h_0).

7.2 Safety Monitor

The Safety Monitor was tested with the procedure described in Sect. 5.2.

Table 2 shows the COV, and FAR of the SM combined with each Controller, for each difficulty level. We observe that as the Controller was trained, the coverage of the SM remained almost unchanged between C_1 and C_2, decreased for C_3, increased again a bit with C_4 and drastically dropped with C_5. Decreased coverage of the SM represents the fact that among the hazardous situations created by the Controller, a larger fraction is harder for the SM to mitigate successfully.

These data confirm that the efficacy of an unchanging SM may depend heavily on the behavior of the Controller, that is, for a ML component, on its training level. With training, the Controller learns to handle by itself some or most of the situations that previously required the SM to intervene; but the fewer hazardous situations it now creates may be too hard for the SM to handle, reducing its effectiveness.

7.3 Whole-Vehicle Evaluation

Table 3 shows the following measures: the rate of occurrence (per km) of crashes if Controller is operating without SM, P(C_crash) (from Table 1), the coverage of the SM (from Table 2), and the rate of occurrence (per km) of crashes with the SM active, P(crash). These three rows are repeated for each difficulty level, $h_0,\ldots,h_3$.

Table 1. Rate of occurrence P(C_crash), per kilometer, of crashes caused by the Controller, in each difficulty level.

	C_1	C_2	C_3	C_4	C_5
h_0	0.95	0.5	0.66	0.54	0.29
h_1	0.95	0.48	0.68	0.64	0.32
h_2	0.96	0.69	0.76	0.79	0.51
h_3	0.97	0.74	0.79	0.8	0.55

Table 2. Coverage and false alarm rate of the SM paired with each Controller

		C_1	C_2	C_3	C_4	C_5
h_0	COV	**0.76**	**0.76**	**0.69**	**0.72**	**0.57**
	FAR	0.005	0.007	0.007	0.008	0.006
h_1	COV	**0.73**	**0.73**	**0.7**	**0.66**	**0.54**
	FAR	0.005	0.007	0.007	0.008	0.005
h_2	COV	**0.71**	**0.75**	**0.71**	**0.73**	**0.6**
	FAR	0.004	0.009	0.009	0.01	0.008
h_3	COV	**0.73**	**0.74**	**0.7**	**0.7**	**0.6**
	FAR	0.004	0.009	0.008	0.01	0.008

Looking at the first two rows, P(C_crash) and COV, for any difficulty level, we see that between the worst and best Controller C_1 and C_5, both decrease: as the Controller learned to cause fewer accidents, it reduced the ability of the SM to *prevent* an accident. Such patterns of contrasting changes appear repeatedly in the table. For example, between the two best Controllers, C_2 and C_5, we observe that for any difficulty level, P(C_crash) improved but COV became worse: $P_{C5}(C\_crash) < P_{C2}(C\_crash)$ but $COV_{C2} > COV_{C5}$. E.g., at difficulty h_0, the additional training that resulted in a 42% improvement of the Controller ($P_{C2}(C\_crash) = 0.5$ but $P_{C5}(C\_crash) = 0.29$) caused a reduction of almost 25% in the coverage of the SM ($COV_{C2} = 0.76 > COV_{C5} = 0.57$). Thus, using the coverage measured on a version of the Controller to estimate the accident rate for a different version may err on the side of optimism.

Table 3. Essential measures of vehicle safety and SM efficacy at different stages of training of the Controller

		C_1	C_2	C_3	C_4	C_5
h_0	P(C_crash)	0.95	0.5	0.66	0.54	0.29
	COV	0.76	0.76	0.69	0.72	0.57
	P(crash)	**0.228**	**0.12**	**0.2046**	**0.1512**	**0.1247**
h_1	P(C_crash)	0.95	0.48	0.68	0.64	0.32
	COV	0.73	0.73	0.7	0.66	0.54
	P(crash)	**0.2565**	**0.1296**	**0.204**	**0.2176**	**0.1472**
h_2	P(C_crash)	0.96	0.69	0.76	0.79	0.51
	COV	0.71	0.75	0.71	0.73	0.6
	P(crash)	**0.2784**	**0.1725**	**0.2204**	**0.2133**	**0.204**
h_3	P(C_crash)	0.97	0.74	0.79	0.8	0.55
	COV	0.73	0.74	0.7	0.7	0.6
	P(crash)	**0.2619**	**0.1924**	**0.237**	**0.24**	**0.22**

Next, we can compare the first and third rows for each difficulty level: the rate of occurrence of crashes without the SM, P(C_crash), against the rate of occurrence of crashes for the complete vehicle (C plus SM), P(crash):

1. adding our SM to *any* version of the Controller reduces the probability of crash if compared to that of the Controller alone.
2. This confirms that our SM is effective. Indeed, this simulation setup is such that it allows the SM to prevent crashes but not cause them, as explained in Sect. 5.1.
3. but making the Controller safer has in certain cases made the vehicle *less* safe.

E.g., Controller C_5 without SM is safer than C_2, but with the SM, the vehicle with Controller C_5 crashes more often that with Controller C_2. The worst case is for difficulty h2: C_5 by itself would cause 26% fewer crashes than C_2, but C_5 with the SM causes 18% crashes more than C_2 with SM. The system was safer with C_2 thanks to the greater efficacy of SM with that Controller, that is, thanks to C_2's flaws "favouring" those accidents that the SM can prevent. Point 2 above indeed proves that, in certain situations, the decreased coverage of the SM may outstrip the Controller improvement and reduce overall vehicle safety. Table 4 highlights this by showing accident rates obtained for the vehicle with C_2, with C_5, and in a hypothetical calculation for C_5 under the wrong assumption of unchanging coverage, i.e., multiplying the SM coverage measured with C_2 by P_{C5}(C_crash). The wrong assumption would lead to an underestimation of the accident rate by 39%.

Table 4. Accident rates (per km, averaged over difficulty levels) of the system for different configurations: $C_2 + SM$, $C_5 + SM$ (observed values), and for the system under the wrong assumption of unchanging coverage of the SM.

	C_2 + SM	C_5 + SM	C_5 with COV_2
P(crash)=P(C_crash)(1-COV)	0.154	0.174	0.1066

8 Concluding Remarks

We have shown an empirical example of how an error checker's (our SM's) efficacy may change when the system that it monitors changes ("learns"). The essential conclusions are that:

1. In this study the safety monitor made *safer every* version of the monitored system, yet it may be *less effective* on an improved version of the monitored system (one that is *safer* than the previous version, *without* the safety monitor).
2. this reduction of coverage may be so large that the new, improved version of the monitored system may be *less safe*, when paired with the safety monitor, than the earlier, worse version was, when paired with the same safety monitor.

With the frequent, hard-to-analyze changes typical in the development of machine learning systems, the implication is that architectures that pair ML components with safety monitors need joint quantitative assessment of the entire architecture at every change of the ML component, a much more onerous process than separate assessment of the ML-based part alone and of the safety monitor, as is often advocated.

Our very basic experiment does not prove that such "unwelcome surprises" will be common in real-life systems, or in autonomous cars in particular; nor that they will be rare. It proves instead that safety arguments cannot assume them to be rare or impossible. We ran the simulations on an "immature" simulated car, allowing us to count large numbers of events that in a real, mature products would very rare. Thus, the car was unsafe from the start, improved very quickly and yet was still unrealistically unsafe at the point where we took the final set of measurements. We do not propose the numbers we report as generalizable to any real-world situation, but rather as a *proof of existence* of the phenomena of concern, lest they be thought possible "only in theory".

These early observations suggest directions for future work including: applying this methodology to more thoroughly trained Controllers, with repeated training, to study the likelihoods of the various possible trends in how improvements to the Controller affect SM coverage; studying how variations in training strategy affect these likelihoods (e.g., would using SM alerts as input in the training, to make the Controller safer, exacerbate the reduction in SM coverage?); a more complete simulation design that allows for SM-caused accidents; more detailed measurement to study various trade-offs involving severity of accident, ride comfort, energy efficiency.

Acknowledgements. Strigini's work was supported in part by ICRI-SAVe, the Intel Collaborative Research Institute on Safe Automated Vehicles. The authors are grateful to Peter Bishop for his insightful comments on the results.

References

1. WAYMO: Technology. https://waymo.com/tech/. Accessed 23 Dec 2021
2. NVIDIA: Training AI for Self-Driving Vehicles: the challenge of scale. https://developer.nvidia.com. Accessed 23 Dec 2021
3. Drago Anguelov (Waymo): MIT Self-Driving Cars (2019)
4. WAYMO: Waymo Safety Report (2021)
5. UNECE: UN Regulation on Advanced Emergency Braking Systems for cars to significantly reduce crashes (2019)
6. EU: Road safety: commission welcomes agreement on new EU rules to help save lives (2019)
7. American Safety Council – Should Autonomous Emergency Braking be Mandatory?
8. Popov, P., Strigini, L.: Assessing asymmetric fault-tolerant software. ISSRE, IEEE (2010)
9. Zhao, X. et al.: Assessing safety-critical systems from operational testing: a study on autonomous vehicles. Inform. Software Technol. **128**, 106393 (2020)
10. Littlewood, B., Strigini, L.: Validation of ultrahigh dependability for software-based systems. Commun. ACM **36**, 69–80 (1993)
11. Butler, R.W., Finelli, G.B.: The infeasibility of quantifying the reliability of life-critical real-time software. IEEE Trans. Software Eng. **19**(1), 3–12 (1993)

12. Kalra, N., Paddock, S.M.: Driving to safety: how many miles of driving would it take to demonstrate autonomous vehicle reliability? Transportation Research Part A: Policy and Practice (2016)
13. Koopman, P., Wagner, M.: Autonomous vehicle safety: an interdisciplinary challenge. IEEE Intell. Transp. Syst. Magaz. **9**, 90–96 (2017)
14. Varshney, K.R.: Engineering safety in machine learning. IEEE ITA (2016)
15. Nguyen, A., Yosinski, J., Clune, J.: Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. CVPR, IEEE, pp. 427–436 (2015)
16. Zhao, D., et al.: Autonomous driving simulation for unmanned vehicles. In: IEEE Winter Conference on Applications of Computer Vision, pp. 185–190 (2015)
17. Baltodano, S., et al.: The RRADS platform: a real road autonomous driving simulator. In: Proceedings of AUTOUI, pp. 281–288 (2015)
18. Osiński, B., et al.: Simulation-based reinforcement learning for real-world autonomous driving. IEEE ICRA, pp. 6411–6418 (2020)
19. Koopman, P., Wagner, M.: Challenges in autonomous vehicle testing and validation. SAE Int. J. Transp. Saf. **4**(1), 15–24 (2016)
20. Grigorescu, S., et al.: A survey of deep learning techniques for autonomous driving. J. Field Robot. **37**(3), 362–386 (2020)
21. Dosovitskiy, A., et al.: CARLA: an open urban driving simulator. CoRL, pp. 1–16 (2017)
22. Lillicrap, T.P., et al.: Continuous Control with Reinforcement Learning. arXiv:150902971 (2015)
23. Caspi, I., Leibovich, G., Novik, G., Endrawis, S.: Reinforcement Learning Coach (2017)
24. Rusu, R.B., Cousins, S.: 3d is here: Point cloud library (pcl). IEEE ICRA, pp. 1–4 (2011)
25. Bozkurt, E.: LidarObstacleDetection (2019). https://github.com/enginBozkurt/
26. Greengard, S.: Gaming machine learning. Commun. ACM **60.12** (2017)
27. Singh, N., et al.: Facial recognition using deep learning. In: Jain, V., Chaudhary, G., Taplamacioglu, M., Agarwal, M. (eds.) Advances in Data Sciences, Security and Applications, LNEE, vol. 612, pp. 375–382. Springer, Singapore (2020). https://doi.org/10.1007/978-981-15-0372-6_30
28. Rao, Q., Jelena F.: Deep learning for self-driving cars: Chances and challenges. SEFAIS 2018
29. Ravi, M., Kantheti, S.C.: Application of artificial intelligence in healthcare: chances and challenges. Curr. J. Appl. Sci. Technol. (2021)
30. Hoang, D.-T., Kang, H.-J.: A survey on deep learning based bearing fault diagnosis. Neurocomputing **335**, 327–335 (2019)
31. Jesse, L., et al.: Towards fully autonomous driving: Systems and algorithms. In: 2011 IEEE Intelligent Vehicles Symposium (IV). IEEE (2011)
32. Koorosh, A., et al.: SafeML: Safety monitoring of machine learning classifiers through statistical difference measures. In: Zeller, M., Höfig, K. (eds.) IMBSA, LNPSE, vol. 12297. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58920-2_13
33. Kurd, Z., Kelly, T., Austin, J.: Developing artificial neural networks for safety critical systems. Neural Comput. Appl. **16**(1), 11–19 (2007)
34. Randy, G., et al.: Explainable AI: the new 42? In: Holzinger, A., Kieseberg, P., Tjoa, A., Weippl, E. (eds.) CD-MAKE, LNISA, vol. 11015. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-99740-7_21
35. Cheng, C.-H.: Safety-aware hardening of 3D object detection neural network systems. In: Casimiro, A., Ortmeier, F., Bitsch, F., Ferreira, P. (eds.) SAFECOMP, LNPSE, vol. 12234. Springer, Cham, 2020. https://doi.org/10.1007/978-3-030-54549-9_14
36. Koopman, P., et al.: Credible autonomy safety argumentation. SCSC, UK (2019)

37. Gauerhof, L., Munk, P., Burton, S.: Structuring validation targets of a machine learning function applied to automated driving. In: Gallina, B., Skavhaug, A., Bitsch, F. (eds.) SAFECOMP 2018. LNCS, vol. 11093, pp. 45–58. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-99130-6_4
38. Huang, X., et al.: A survey of safety and trustworthiness of deep neural networks: verification, testing, adversarial attack and defence, and interpretability. Comput. Sci. Rev. **37**, 100270 (2020)