
Monitoring of Perception Systems: Deterministic, Probabilistic, and Learning-based Fault Detection and Identification

Pasquale Antonante*, Heath Nilsen, Luca Carlone

Massachusetts Institute of Technology
77 Massachusetts Ave, Cambridge, MA 02139
{antonap, hnilsen, lcarlone}@mit.edu

Abstract

Perception is a critical component of high-integrity applications of robotics and autonomous systems, such as self-driving cars. In these applications, failure of perception systems may put human life at risk, and a broad adoption of these technologies requires the development of methodologies to guarantee and monitor safe operation. Despite the paramount importance of perception, currently there is no formal approach for system-level perception monitoring. This paper investigates runtime monitoring of perception systems. We formalize the problem of runtime fault detection and identification in perception systems and present a framework to model diagnostic information using a *diagnostic graph*. We then provide a set of deterministic, probabilistic, and learning-based algorithms that use diagnostic graphs to perform fault detection and identification. Moreover, we investigate fundamental limits and provide deterministic and probabilistic guarantees on the fault detection and identification results. We conclude the paper with an experimental evaluation, which recreates several realistic failure modes in the LGSVL open-source autonomous driving simulator, and applies the proposed system monitors to a state-of-the-art autonomous driving software stack (Baidu’s Apollo Auto). The results show that the proposed system monitors outperform baselines, have the potential of preventing accidents in realistic autonomous driving scenarios, and incur a negligible computational overhead.

1 Introduction

The number of Autonomous Vehicles (AVs) on our roads is increasing rapidly, with major players in the space already offering autonomous rides to the public [1]. Self-driving cars promise a deep transformation of personal mobility and have the potential to improve safety, efficiency (*e.g.*, commute time, fuel), and induce a paradigm shift in how entire cities are designed [2]. One key factor that drives the adoption of such technology is the capability of ensuring and monitoring safe operation. In a recent survey [3], the American Automobile Association (AAA) reports that vehicles with autonomous driving features consistently failed to avoid crashes with other cars or bicycles. Moreover, an analysis by Business Insider [4] found that the number of accidents involving AVs surged in 2021. This is a clear sign that the industry needs a sound methodology, embedded in the design process, to guarantee safety and build public trust.

Safe operation requires AVs to correctly understand their surroundings, in order to avoid unsafe behaviors. In particular, AVs rely on onboard *perception systems* to provide situation awareness

*This work was partially funded by the NSF CAREER award “Certifiable Perception for Autonomous Cyber-Physical Systems”

and inform the onboard decision-making and control systems. Modern perception systems use both data-driven and classical methods. While classical methods are well-rooted in signal processing and estimation theory and have been extensively studied in robotics and computer vision, they may still have unexpected failure modes in practice, *e.g.*, local convergence in the Iterative Closest Point for 3D object pose estimation [5] or premature termination of robust estimation techniques such as RANSAC [6]. The use of data-driven methods further exacerbates the problem of ensuring correctness of the perception outputs, since current neural network architectures are still prone to creating unexpected and often unpredictable failure modes [7]. Due to the complexity of monitoring the correct operation of the perception system, the self-driving car industry heavily relies on simulation and testing to provide evidence of safety. Although there is an increasing interest in the area of safety certification and runtime monitoring, the literature lacks a system-level framework to organize and reason over the diagnostic information available at runtime for the purpose of detecting and identifying potential perception-system failures. Reliable runtime monitoring would enable the vehicle to have a better understanding of the conditions it operates in, and would give it enough notice to take adequate actions to preserve safety (*e.g.*, switch to fail-safe mode or hand over the control to a human operator) in case of severe failures. In this paper, we use the term “failure” (or “fault”) in a general sense, including failures of the *intended functionality* [8, 9]. For instance, a neural network can execute correctly (*e.g.*, without errors in the implementation or in the hardware running the network) but can still fail to produce a correct prediction for out-of-distribution inputs. In this context, *fault detection* is the problem of detecting the *presence* of a fault in the system, while *fault identification* is the problem of inferring which components of the system are faulty. The latter is particularly important since (i) not every fault has the same severity, hence understanding which component is failing may lead to different responses, (ii) a designer can use fault statistics to decide to focus research and development efforts on certain components, and (iii) a regulator can use information about specific faults to trace the steps or determine responsibilities after an accident.

Most of the existing literature (which we review more extensively in Section 2) has focused on detecting failures of specific modules or specific algorithms, like localization [10, 11], semantic segmentation [12], or obstacle detection [13]. These methodologies often use a white-box approach (the monitor knows how the monitored algorithm works to some extent), and are sometimes computationally expensive to run [12].

Contribution. This paper addresses this gap and provides methodologies for runtime monitoring (in particular, fault detection and identification) of complex perception systems using a system-level approach. Our first contribution is to formalize the problem (Section 3) and to present a probabilistic and deterministic framework (Section 4.1) to organize heterogeneous diagnostic tests of a perception system into a graphical model, the *diagnostic graph*. Our framework adopts a black-box approach, in that it remains agnostic to the inner workings of the perception algorithms, and only focuses on collecting results from diagnostic tests that check the validity of their outputs. Our second contribution is a set of algorithms (Section 4.2) that use diagnostic graphs to perform fault detection and identification. Our third contribution (Section 5) is to investigate fundamental limits and provide deterministic and probabilistic guarantees on the fault detection and identification results. This allows us to establish formal guarantees on the maximum number of faults that can be uniquely identified in a given perception system, leading to the notion of *diagnosability*. Finally, we show that our framework is effective in detecting and identifying faults in a real-world perception pipeline for obstacle detection (Section 6). In particular, we perform experiments using a realistic open-source autonomous driving simulator (the LGSVL Simulator [14]) and a state-of-the-art autonomous driving software stack (Baidu’s Apollo Auto [15]). The open-source version of our code can be found at <https://github.com/MIT-SPARK/PerceptionMonitoring>.

2 Related Work

State of Practice. The automotive industry currently uses four classes of methods to claim the safety of an AV [16], namely: miles driven, simulation, scenario-based testing, and disengagement. The *miles driven* approach is based on the statistical argument that if the probability of crashes per mile is lower for autonomous vehicles than for humans, then AVs are safer; however, such an analysis would require an impractical amount (*i.e.*, billions) of miles to produce statistically-significant results [17, 16]. The same approach can be made more scalable through *simulation* or *scenario-based* testing, however creating a life-like simulator is an open problem and enumerating

all possible corner cases is a daunting task. Finally, the number of *disengagement*, defined as the number of times a safety driver has to intervene to prevent a hazardous situation, does not give enough evidence of the system safety.

An established methodology to ensure safety is to develop a *standard* that every manufacturer has to comply with. In the automotive industry, the standard ISO 26262 [18] is a risk-based safety standard that applies to electronic systems in production vehicles. It mostly focuses on electronic systems rather than algorithmic aspects, hence it does not readily apply to fully autonomous vehicles [19]. The recent ISO 21448 [8], which extends the scope of ISO 26262 to cover AVs functionality, primarily considers mitigating risks due to unexpected operating conditions, and provides high-level considerations on best-practice for the development life-cycle. Koopman and Wagner [20] proposed a standard called UL 4600 [21] specifically designed for high-level autonomy (levels 4 and 5). This standard focuses on ensuring that a comprehensive safety case is created, but it is technology-agnostic, meaning that it requires evidence of system safety without prescribing the use of any specific approach or technology to achieve it.

State of the Art. Related work tries to tackle the problem of safety assurance using different strategies. *Formal methods* [22–30] have been used as a tool to study safety of autonomous systems. These approaches have been successful for decision systems, such as obstacle avoidance [31], road rule compliance [32], high-level decision-making [33], and control [34, 35], where the specifications are usually model-based and have well-defined semantics [36]. However, they are challenging to apply to perception systems, due to the complexity of modeling the physical environment [37, 38]. Current approaches [39–41] consider high-level abstractions of perception [16, 42, 43] or rely on simulation to assert the true state of the world [39, 40, 44]. Other approaches focus on adversarial attacks for neural-network-based object detection [45–47]; these methods derive bounds on the magnitude of the perturbation that induces incorrect detection result, and are typically used off-line [48].

Previous works on **runtime fault detection and identification** focuses on components of the perception system [49]. Miller *et al.* [13] propose a framework for quantifying false negatives in object detection. For semantic segmentation, Besnier *et al.* [12] propose an out-of-distribution detection mechanism, while Rahman *et al.* [50] propose a failure detection framework to identify pixel-level misclassifications. Lambert and Hays [51] propose cross-modality fusion algorithm to detect changes in high-definition map. Liu and Park [52] propose a methodology to analyze the consistency between camera image data and LiDAR data to detect perception errors. Sharma *et al.* [53] propose a framework for equipping any trained deep network with a task-relevant epistemic uncertainty estimate. Another line of works leverages spatio-temporal information to detect failures. You *et al.* [54] use spatio-temporal information from motion prediction to verify 3D object detection results. Balakrishnan *et al.* [42, 55] propose the Timed Quality Temporal Logic (TQTL) to reason about desirable spatio-temporal properties of a perception algorithm. Kang *et al.* [56] use model assertions, which similarly place a logical constraint on the output of a module to detect anomalies.

Fault identification and anomaly detection have been extensively studied in *other areas of engineering*. Bayesian networks, Hidden Markov Models [57, 58], and deep learning [59] have been used to enable fault identification, but mainly in industrial systems instrumented to detect component failures. Graph-neural networks have been used for anomaly detection (see [60] for a comprehensive survey). In this context, “anomaly detection aims to identify the unusual patterns that deviate from the majorities in a dataset” [60]. De Kleer and Williams [61] assumes the availability of a model that predicts the behavior of the system and propose a methodology to detect failures by comparing observations with a predicted output. Preparata, Metze, and Chien [62] study the problem of fault diagnosis in multi-processor systems, introducing the concept of diagnosability; their work is then extended by subsequent works [63–65]. Sampath *et al.* [66] propose the concept of diagnosability for discrete-event systems [67, 68]. The system is modeled as a finite-state machine, and is said to be diagnosable if and only if a fault can be detected after a finite number of events.

The proposed work extends the literature in several directions. First, we take a black-box approach and remain agnostic to the inner workings of the perception system we aim to monitor (relaxing assumptions in related work [61]). Second, we develop a fault identification framework that reasons over the diagnostic information of heterogeneous and potentially asynchronous perception modules (going beyond the homogeneous, synchronous, and deterministic framework in [62] or discrete-event systems [67, 68]).

3 Problem Statement: Fault Detection and Identification

3.1 Perception Systems

A perception system \mathcal{S} comprises a finite set of interconnected *modules* $\mathcal{M} = \{m_1, m_2, \dots, m_{|\mathcal{M}|}\}$; for instance, the perception system of a self-driving car may include modules for lane detection, camera-based object detection, LiDAR-based motion estimation, ego-vehicle localization, etc. Each module $m \in \mathcal{M}$ produces one or more *outputs*. For instance, the lane detection module may produce an estimate of the 3D location of the lane boundaries, while the pedestrian detection module may produce an estimate of the pose and velocity of pedestrians in the surroundings. Some of these outputs provide inputs for other perception modules, while other are the outputs of the perception system and feed into other systems (e.g., to planning and control). The sets of modules' outputs are disjoint (i.e., each output is produced by a single module), and the set of all outputs is denoted by \mathcal{O} . We model the perception system as a graph of modules and outputs.

Definition 1 (Perception System). *A perception system \mathcal{S} is a directed graph $\mathcal{S} = (\mathcal{M} \cup \mathcal{O}, \mathcal{E})$, where the set of nodes $\mathcal{M} \cup \mathcal{O}$ describes modules and outputs in the system, while the set of edges \mathcal{E} describes which module produces or consumes a certain output. In particular, an edge $(m_i, o_j) \in \mathcal{E}$ with $m_i \in \mathcal{M}$ and $o_j \in \mathcal{O}$ models the fact that module m_i produces output o_j . Similarly, and edge $(o_j, m_i) \in \mathcal{E}$ with $o_j \in \mathcal{O}$ and $m_i \in \mathcal{M}$ models the fact that module m_i uses output o_j .*

We treat each module as a *black-box* and remain agnostic to the algorithms they implement. This allows our framework to generalize complex perception systems, possibly including a combination of classical and data-driven methods. For some examples of systems, see Appendix C.

3.2 Fault Detection and Fault Identification

Each module in \mathcal{S} might fail at some point, jeopardizing the system performance or even its safety. In particular, each module $m \in \mathcal{M}$ is assumed to have a set of failure modes. While the list of failures can include any software and hardware failures, in this work we particularly focus on failures of the intended functionality. For example, a neural-network-based camera-based object detection module might experience the failure mode “out-of-distribution sample” when it processes an input image, which indicates that while the module’s code executed successfully, the resulting detection is expected to be incorrect. Similarly, each output $o \in \mathcal{O}$ has an associated set of failure modes. For instance, the output of the camera-based object detector might experience a “mis-detection” failure mode if it fails to detect an object, or a “mis-classification” failure mode if the object is detected but misclassified. A module’s failure mode typically causes a failure in one of its outputs.

Definition 2 (Failure Modes). *At each time instant, the i -th failure mode $f_i \in \{\text{INACTIVE}, \text{ACTIVE}\} \cong \{0, 1\}$ is either **ACTIVE** (also 1) if such failure is occurring, or **INACTIVE** (also 0). If we stack the status (**ACTIVE/INACTIVE**) of all failure modes into a single binary vector, i.e., the fault state vector $\mathbf{f} \in \{0, 1\}^{N_f}$ (where N_f is the number of failure modes), then \mathbf{f} is all zeros if there are no faults, or has entries equal to ones for the active failure modes.*

The goal of this work is then to address the following problems: (i) **Fault Detection**, namely, deciding whether the system is working in nominal conditions or whether a fault has occurred (i.e., infer if there is at least an active failure mode in \mathbf{f}); (ii) **Fault Identification**, namely, identifying the specific failure mode the system is experiencing (i.e., infer which failure mode is active in \mathbf{f}).

Fault detection is the easiest between the two problems, as it only requires specifying the presence of at least a fault, without specifying which modules or outputs are experiencing it. Fault identification goes one step further by explicitly indicating the set of active failure modes. Note that solving fault identification implies a solution for fault detection (i.e., whenever we declare one or more modules to be faulty, we essentially also detected there is a failure), hence in the rest of this paper we focus on the design of a monitoring system for fault identification.

Remark 3 (Modules vs. Outputs). *Our model treats modules and outputs as separate nodes. This is convenient for two reasons. First, fault identification at the modules and outputs may serve different purposes: output fault identification is more useful at runtime to identify unreliable information and prevent accidents; module fault identification is more informative for designers and regulators. Second, in practical applications we can rarely measure if a module is failing. On the other hand, we can directly measure the outputs of the modules and develop diagnostic tests to check if an output is plausible and consistent with other outputs in the system.*

4 Modeling Fault Identification with Diagnostic Graphs

In the previous section we have discussed how the goal is to identify the set of active failure modes. Here we introduce the concept of *diagnostic graphs* (Section 4.1) to study fault identification: diagnostic graph will allow developing fault identification algorithms (Section 4.2) and understanding fundamental limits (Section 5).

4.1 Diagnostic Tests and Diagnostic Graphs

The system \mathcal{S} is equipped with a set of *diagnostic tests* that can (possibly unreliably) provide diagnostic information about the state of a subset of failure modes. For instance, we can compare the outputs of different modules to ensure they are consistent (e.g., compare the obstacles detected by the LiDAR-based obstacle detection against the obstacles detected by the camera-based obstacle detection), or inspect that the output a certain module respects certain requirements (e.g., the vision-based ego-motion module is tracking a sufficient number of features). Each diagnostic test is a function $t : \mathbb{S} \rightarrow \{PASS, FAIL\}$, where $\mathbb{S} \subseteq \{1, \dots, N_f\}$ is a subset of the failure modes that the test is checking, called the *scope* of the test, and the test returns a value $z \in \{PASS, FAIL\}$, called the *outcome* of the test.

In this paper we allow for both *deterministic* and *probabilistic* diagnostic tests. Deterministic diagnostic tests encode the set of possible test outcomes by establishing a deterministic relation between failure modes in the test’s scope and the test outcome. This relation can also capture unreliable outcomes, for example allowing multiple test outcomes for a given set of active failure modes (see examples in Fig. 3 in the appendix). Deterministic tests can leverage formal methods tools or certifiable perception algorithms [69–71]. However, most practical tests are likely to incorrectly detect faults (i.e., produce false positive) or fail to detect faults (i.e., produce false negatives) with some probability. For this reason, we also allow for an arbitrary probabilistic relationship between test outcomes and failure modes in the test scope. For examples of deterministic and probabilistic diagnostic tests, see Appendix B. We remark that a single test does not suffice for fault identification. The collection of the outcomes of multiple diagnostic tests is called a *syndrome*.

To enable fault detection and identification we define the concept of *diagnostic graph*, a structure defined over a perception system that has the goal of describing the diagnostic tests (as well as more general relations among failure modes) and their scope.

Definition 4 (Diagnostic Graph). A diagnostic graph is a bipartite graph $\mathcal{D} = (\mathcal{V}, \mathcal{R}, \mathcal{E})$ where the nodes are split into variable nodes \mathcal{V} , corresponding to the failure modes in the system, and relation nodes \mathcal{R} , where each relation $\phi_k(f) \in \mathcal{R}$, with $k \in \{1, \dots, |\mathcal{R}|\}$, is a function over a subset of failure modes f . Then an edge in \mathcal{E} exists between a failure mode $f_i \in \mathcal{V}$ and a relation $\phi_k \in \mathcal{R}$, if f_i is in the scope of the relation ϕ_k (i.e., if the variable f_i appears in the function ϕ_k).

Relations capture constraints among the variables induced by the test outcomes or from prior knowledge we might have about the failure modes. The relations can be **test-driven relations** if they represent diagnostic tests behaviors or **a priori relations** if they represent a priori knowledge about the system (e.g., input/output relationships between modules or mutual exclusion constraints between failure modes). Both kind of relations can be **deterministic** or **probabilistic**. The diagnostic graph can also model temporal evolution of the failure modes by simply stacking multiple regular diagnostic graphs (one for each time step considered), preserving the failure modes, relations and edges of each sub-graph. We call these graphs, **temporal diagnostic graphs**.

Remark 5 (Temporal Diagnostic Tests). Temporal diagnostic tests are used to monitor the evolution of the system over time (spatio-temporal constraints). For example the Timed Quality Temporal Logic in [43] can be implemented with a temporal diagnostic test that spans multiple $\mathcal{D}^{(t)}$ ’s. Consider the following example (as in [43]): “At every time step, for all the objects in the frame, if the object class is cyclist with probability more than 0.7, then in the next 5 frames the same object should still be classified as a cyclist with probability more than 0.6”. This can be modeled as a diagnostic test that spans 5 diagnostic graphs and that returns FAIL if the predicate is false.

4.2 Algorithms for Detection and Identification

In this section we briefly describe how to perform fault identification over a diagnostic graph. In the deterministic case, our inference algorithm looks for the smallest set of active failure modes that

explains a given syndrome \mathbf{z} , namely, the set of diagnostic test outcomes.

$$\begin{aligned} \mathbf{f}^* = & \arg \min_{\mathbf{f} \in \{0,1\}^{N_f}} \|\mathbf{f}\|_1 \\ & \text{subject to} \quad \text{test-driven relations given the syndrome } \mathbf{z}, \\ & \quad \quad \quad \text{a priori relations,} \end{aligned} \tag{D-FI}$$

Conversely, in the case of probabilistic relations, the fault identification problem reduces to the *maximum a posteriori* (MAP) inference. In MAP inference, given a syndrome \mathbf{z} , we look for the most likely variables \mathbf{f}^* that maximize the posterior distribution μ defined by the diagnostic graph:

$$\mathbf{f}^* = \arg \max_{\mathbf{f} \in \{0,1\}^{N_f}} \mu(\mathbf{f} \mid \mathbf{z}) \tag{P-FI}$$

Deterministic fault identification in Eq. (D-FI) can be solved using mixed-integer programming solvers like Google OR-Tools [72]. The probabilistic fault identification instead can be solved using probabilistic graphical models such as the factor graphs [73] or using learning methods such as factor graph neural networks [74] or graph neural network (GNNs) [75]. It is straightforward to convert a diagnostic graph into a factor graph, which allows us to use the computational tools in the literature (both exact and approximate) to solve Eq. (P-FI). In our experiments we use belief propagation (Sec. 3 in [76]) to solve the MAP inference, which finds the optimal solution for tree-structured factor graphs, and is known to empirically return good approximations for the MAP estimate in general factor graphs. In our experiments we also use GNNs to solve Eq. (P-FI). In particular we use GCN [77], GCNII [78], GIN [79] and GraphSAGE [80]. To do so, we convert the diagnostic graph into an undirected graph (details in Appendix D.3).

While in the deterministic case we know the expression of the relations, in the probabilistic case the probabilistic tests might depend on unknown parameters. There are several paradigms to learn these parameters in factor graphs. In our experiments, we use a method called *maximum margin learning* (Sec. 19.7 in [81]). Instead, in order to train the GNNs, we use a supervised learning approach. In particular, we use a softmax classification function and negative log-likelihood training loss, which is available in standard libraries, such as PyTorch [82].

5 Fundamental Limits

Given a diagnostic graph it is natural to ask, can we guarantee that our fault identification algorithm will be able to correctly identify all active faults? Under which conditions? We answer these questions in this section, where we introduce the concept of *diagnosability*.

5.1 Deterministic Diagnosability

We start with the definition of deterministic diagnosability.

Definition 6 (κ -diagnosability [62, 83]). *A diagnostic graph \mathcal{D} is κ -diagnosable if, given any syndrome, all active failure modes can be correctly identified, provided that the number of active failure modes in the system does not exceed κ .*

The concept of κ -diagnosability helps measure if the system has enough redundancy and if the diagnostic tests are able to capture it for the purpose of fault identification. It basically requires that each syndrome unequivocally encodes the configurations of failure modes that generated it. In the general case, to compute the diagnosability value is necessary to check every subset of failure modes of cardinality up to κ (and their syndromes). However, under technical assumptions, one can directly compute the diagnosability by only looking at the topology of the diagnostic graph [84, 63].

Let us now move our attention to temporal diagnostic graphs. Denote with $\kappa(\mathcal{D})$ the diagnosability of \mathcal{D} . Then the following result characterizes the diagnosability of temporal diagnostic graphs.

Theorem 7 (Diagnosability in Temporal Diagnostic Graphs). *Let $\mathcal{D}^{[K]}$ be a temporal diagnostic graph built by stacking a set of K regular diagnostic graphs $\mathcal{D}^{(1)}, \dots, \mathcal{D}^{(K)}$. Then $\kappa(\mathcal{D}^{[K]}) \geq \min_{i \in \{1, \dots, K\}} \kappa(\mathcal{D}^{(i)})$.*

Proof. Proof in Appendix A.1. □

As an immediate result of Theorem 7, we have that the diagnosability of a temporal diagnostic graph obtained by stacking “identical” regular diagnostic graph over time is monotonically increasing, showing the benefits of temporal diagnostic graphs.

5.2 Probabilistic Diagnosability

The notion of κ -diagnosability introduced in the previous section imposes strong conditions on \mathcal{D} . When the tests are probabilistic, such a condition becomes too stringent: intuitively, κ -diagnosability deals with the worst case over all possible test outcomes, which becomes too conservative when every outcome is possible (with some probability). For this reason, we extend the definition of diagnosability to deal with the case where the diagnostic graph includes probabilistic tests.

Towards defining a probabilistic notion of diagnosability, we introduce the *Hamming distance* $h(\mathbf{f}, \mathbf{f}') = \sum_{i=1}^{N_f} \mathbb{1}[f_i \neq f'_i]$ between two binary vectors \mathbf{f} and \mathbf{f}' where $\mathbb{1}$ is the indicator function. Assuming that \mathbf{f} is the binary vector describing the active failures in the system, and that \mathbf{f}' is an estimated vector of the fault states, the Hamming distance simply counts the number of *mis-identified faults*. We are now ready to introduce the following probabilistic definition of diagnosability.

Definition 8 ((Probably Approximately Correct) PAC-Diagnosability). *Consider a fault identification algorithm $\Psi_{\mathcal{D}}$ applied to a diagnostic graph \mathcal{D} . The diagnostic graph \mathcal{D} is (γ, p) -PAC-diagnosable under $\Psi_{\mathcal{D}}$, if, for some $1 \leq \gamma \leq N_f$*

$$\Pr_{(\mathbf{z}, \mathbf{f}) \sim \mathcal{F}} [h(\Psi_{\mathcal{D}}(\mathbf{z}), \mathbf{f}) \leq \gamma] \geq p$$

where \mathcal{F} is the joint distribution of potential failures and test outcomes.

This definition simply says that a given fault identification algorithm applied to the diagnostic graph \mathcal{D} is (γ, p) -PAC-diagnosable if it is expected to make less than γ mistakes with probability at least p . Since the outcome of the tests is a random variable, so is the Hamming distance $h(\Psi_{\mathcal{D}}(\mathbf{z}), \mathbf{f})$. Therefore, we can define its expected value as $h_{\mathcal{F}}(\Psi_{\mathcal{D}}) = \mathbb{E}_{(\mathbf{z}, \mathbf{f}) \sim \mathcal{F}} [h(\Psi_{\mathcal{D}}(\mathbf{z}), \mathbf{f})]$. This quantity is the number of mistakes that the fault identification algorithm $\Psi_{\mathcal{D}}$ is expected to make. Let us suppose we have a dataset \mathcal{W} of i.i.d. samples of the underlying faults distribution \mathcal{F} . Let $\hat{h}_{\mathcal{W}}(\Psi_{\mathcal{D}}) = 1/|\mathcal{W}| \sum_{(\mathbf{z}, \mathbf{f}) \in \mathcal{W}} h(\Psi_{\mathcal{D}}(\mathbf{z}), \mathbf{f})$ be the empirical number of mistakes the fault identification algorithm $\Psi_{\mathcal{D}}$ makes on \mathcal{W} . For instance, if we are given a dataset \mathcal{W} describing the system execution, with the corresponding ground truth failure modes states \mathbf{f} , we can test our algorithm $\Psi_{\mathcal{D}}$ and calculate the empirical number of mistakes $\hat{h}_{\mathcal{W}}(\Psi_{\mathcal{D}})$ it makes. Then, we can use the following result to bound the expected number of mistakes our algorithm will make over all future scenarios.

Theorem 9 (Fault Identification Error Bound). *Consider a dataset \mathcal{W} of i.i.d. samples of the underlying faults distribution \mathcal{F} , and a fault identification algorithm $\Psi_{\mathcal{D}}$ over \mathcal{D} . Then, for any $\delta > 0$, the following inequality holds with probability at least $1 - \delta$:*

$$h_{\mathcal{F}}(\Psi_{\mathcal{D}}) \leq \hat{h}_{\mathcal{W}}(\Psi_{\mathcal{D}}) + N_f \sqrt{\frac{\log(2/\delta)}{2|\mathcal{W}|}}.$$

Proof. Proof in Appendix A.2. □

This result allows us to give performance guarantees by estimating the number of mistakes the fault identification algorithm is expected to make on the provided dataset.

6 Experimental Evaluation

This section shows that diagnostic graphs are an effective model to detect and identify failures in complex perception systems. In particular, we show that the proposed monitors (i) outperform baselines in terms of fault identification accuracy, (ii) allow detecting failures and provide enough notice to prevent accidents in realistic test scenarios, and (iii) add minimal overhead. In the following we mainly focus on temporal diagnostic graph, additional results are reported in Appendix E.

We test our runtime monitors in several scenarios, specifically designed to stress-test the perception system (for more details see Appendix F). The scenarios are simulated using the LGSVL Simulator [14], an open-source autonomous driving simulator. We apply our monitors to a state-of-the-art

Algorithm	Fault Identification Accuracy			Fault Detection Accuracy			Timing
	All	Outputs	Modules	All	Outputs	Modules	Milliseconds
Factor Graph	93.60	96.88	83.74	81.60	91.41	71.78	2.53
Deterministic	89.26	92.33	80.06	93.25	93.25	93.25	3.68
Baseline (w/rel. scores)	90.18	92.69	82.67	85.28	85.28	85.28	0.27
Baseline	83.90	87.73	72.39	85.28	85.28	85.28	0.26
GCN	91.79	96.06	78.99	80.06	90.18	69.94	0.68
GCNII	92.60	96.01	82.36	78.83	85.89	71.78	24.56
GIN	93.21	96.47	83.44	83.13	92.64	73.62	0.50
GraphSage	92.71	96.42	81.60	79.14	89.57	68.71	0.85

Table 1: Results of the proposed algorithms for detection and identification on temporal diagnostic graphs, averaged over all scenarios. Best values are highlighted in green, second-best in yellow.

perception system. In our experiments we focus on the object-detection pipeline of Baidu’s Apollo Auto [15] version 7 [85]. Baidu’s Apollo is an open-source, state-of-the-art, autonomous driving stack that includes all the relevant functionalities for level 4 autonomous driving. For the details of the diagnostic graph and diagnostic tests used in our experiments, see Appendix D.

Baselines. We compared the proposed monitors against two simple baselines. In the first baseline (label: “Baseline”), whenever a diagnostic test returns FAIL, all failure modes in its scope are considered active. In the second baseline (label: “Baseline (w/rel. scores)”), modules are ordered by a reliability score defined by the system designer. In our experiments we considered the radar to be more reliable than the sensor fusion, which is more reliable than the LiDAR, which in turn is more reliable than the camera. When a diagnostic test fails, this second baseline labels all the failure modes in the test scope associated to the least reliable module (and its outputs) as ACTIVE.

6.1 Fault Detection and Identification Results

We used three metrics to evaluate the performance for both the fault detection and identification problems: (i) accuracy, the percentage of correctly detected (resp. identified) failures over the total number of samples; (ii) precision, the percentage of correct detections (resp. identifications) over the number of failures reported (a monitor achieves high precision if it has a low rate of false alarms); and, (iii) recall, the percentage of correct detections (resp. identifications) over the number of failures the system experienced (a monitor has high recall if it is able to catch a large fraction of failures occurring in the perception system).

Fault Identification Results. Table 1 reports the fault identification accuracy of all compared techniques, averaged across all test scenarios (described in Appendix F). The overall accuracy results suggest that factor-graph-based probabilistic fault identification outperforms all other algorithms and achieves 96.88% accuracy with temporal diagnostic graphs. GIN architectures achieve the second-best performance. Looking at the results we notice that output fault identification has higher accuracy than module fault identification; this is expected, since most of our tests directly involve outputs, while we can only indirectly infer module failures via the a priori relations. Note that the two statistics (output fault identification vs. module fault identification) are typically used for different purposes, as discussed in Remark 3. Fig. 1 shows precision-recall trade-offs when using temporal diagnostic graphs. Also in this case, the factor graph achieves the best performance overall, with fault identification precision and recall being better in the output space. The figure shows that the baseline makes conservative decisions (high recall and low precision).

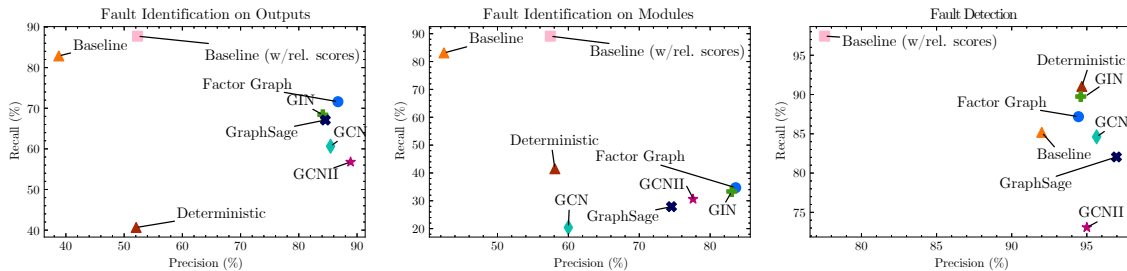


Figure 1: Precision/Recall for fault detection and identification on temporal diagnostic graphs.

Fault Detection Results. Recall that fault detection is the problem of deciding whether the system is working in normal conditions or whether at least a fault has occurred. Table 1 and Fig. 1 show detection accuracy, precision, and recall. Table 1 shows that the deterministic approach and the baselines do particularly well for fault detection: they both detect failure as soon as a single test fails, which makes their accuracy high. On the other hand, the factor graph approach may prefer explaining a failed test as a false alarm. Therefore, while factor graphs would be the go-to approach for fault identification, a simpler baseline approach suffices for fault detection.

Timing. All algorithms perform inference in less than 4 ms, except for GCNII which averages at around 24 ms. This is likely due to the fact that GCNII uses a deep architecture, which incurs an increased computational cost. The best performing algorithm, *i.e.*, the factor graph, can be executed in real-time as its runtime averages around 2.5 ms for temporal graphs. The fastest algorithm is the baseline, with a runtime below 0.3 ms for temporal graphs.

Diagnosability. Let us now discuss the deterministic diagnosability of the perception system considered in our experiments (the diagnostic graph is described in appendix Appendix D.2). If the tests are reliable (never fail to detect a failure) the diagnostic graph used in our experiments is 5-diagnosable. This means that if there are up to 5 active failure modes the deterministic fault identification will be able to correctly identify them. If we instead assume the tests are unreliable, for example we assume tests might fail when all the failure modes in their scope are active, the diagnostic graph is 3-diagnosable. It’s worth noticing that this does not mean that if there are more than 3 (or 5) active failure modes the fault identification will surely fail, but rather that we do not have the guarantee that it will not make any mistake. Fig. 2 show the PAC-Diagnosability bound defined in Definition 8 for each of the compared techniques. The bound represents the number of fault identification mistakes each algorithm is expected to make with a given confidence (δ in Theorem 9). The plots show that with high probability, most of the algorithms are expected to make less than 1 mistake in the fault identification (*i.e.*, false alarms or false negatives). The factor graph has the lowest bound of all methods.

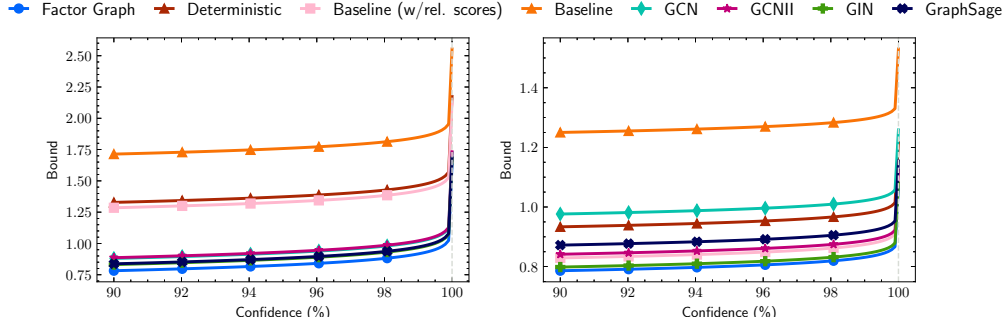


Figure 2: PAC-diagnosability bounds. (Left) Modules, (Right) Outputs. Lower is better.

7 Conclusions

This paper investigated runtime monitoring of complex perception systems. Toward this goal, we formalized the concept of *diagnostic tests*, a generalization of runtime monitors, and introduced the concept of *diagnostic graph*, as a structure to organize diagnostic information. We then provided a set of deterministic, probabilistic, and learning-based algorithms that use diagnostic graphs to perform fault detection and identification. Moreover, we investigated fundamental limits and provided deterministic and probabilistic guarantees on the fault detection and identification results. We conclude the paper with the experimental evaluation, showing that the proposed system monitors outperform baselines in terms of fault identification accuracy, and have the potential of preventing accidents in realistic scenarios, while incurring in a negligible computational overhead.

This work opens a number of avenues for future work. First, we plan to test our monitors on real-world datasets and to provide more examples of the proposed approach applied to other perception subsystems (*e.g.*, localization, lane segmentation). Second, we plan to add a risk metric to the fault identification process that could help the decision layer to make more informed decisions. Finally, in this paper, we used simple diagnostic tests. Moving forward, it would be desirable to use more advanced diagnostic tests available in the literature.

References

- [1] Google’s self-driving startup Waymo is introducing fully driverless rides to San Francisco, <https://www.businessinsider.com/waymo-testing-fully-automated-cars-san-francisco-2022-3>, accessed: 2022-05-14.
- [2] G. Silberg, R. Wallace, G. Matuszak, J. Plessers, C. Brower, D. Subramanian, Self-driving cars: The next revolution, White paper, KPMG LLP & Center of Automotive Research 9 (2) (2012) 132–146.
- [3] American Automobile Association, Active driving assistance system performance, <https://newsroom.aaa.com/asset/active-driving-assistance-system-performance-may-2022/> (2022).
- [4] Waymo and Cruise self-driving cars took over San Francisco streets at record levels in 2021 — so did collisions with other cars, scooters, and bikes, <https://www.businessinsider.com/self-driving-car-accidents-waymo-cruise-tesla-zoox-san-francisco-2022-1>, accessed: 2022-05-14.
- [5] H. Yang, J. Shi, L. Carlone, TEASER: Fast and Certifiable Point Cloud Registration, arXiv preprint arXiv: 2001.07715(pdf) (2020).
- [6] M. Fischler, R. Bolles, Random sample consensus: a paradigm for model fitting with application to image analysis and automated cartography, *Commun. ACM* 24 (1981) 381–395.
- [7] R. Salay, R. Queiroz, K. Czarnecki, An analysis of iso 26262: Using machine learning safely in automotive software, arXiv preprint arXiv:1709.02435 (2017).
- [8] ISO Standard, Road vehicles — safety of the intended functionality, iSO/PAS 21448:2019(en) (2019).
- [9] Aptiv, Audi, B. Apollo, BMW, Continental, Daimler, F. Group, Here, Infineon, Intel, Volkswagen, Safety First for Automated Driving (2019).
URL <https://www.daimler.com/innovation/case/autonomous/safety-first-for-automated-driving-2.html>
- [10] H. Jing, Y. Gao, S. Shahbeigi, M. Dianati, Integrity monitoring of gnss/ins based positioning systems for autonomous vehicles: State-of-the-art and open challenges, *IEEE Transactions on Intelligent Transportation Systems* (2022).
- [11] O. A. Hafez, G. D. Arana, M. Joerger, M. Spenko, Quantifying robot localization safety: A new integrity monitoring method for fixed-lag smoothing, *IEEE Robotics and Automation Letters* 5 (2) (2020) 3182–3189.
- [12] V. Besnier, A. Bursuc, D. Picard, A. Briot, Triggering failures: Out-of-distribution detection by learning from local adversarial attacks in semantic segmentation, in: *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 15701–15710.
- [13] D. Miller, P. Moghadam, M. Cox, M. Wildie, R. Jurdak, What’s in the black box? the false negative mechanisms inside object detectors, arXiv preprint arXiv:2203.07662 (2022).
- [14] LG, LGSVL Simulator.
URL <https://www.lgsvlsimulator.com>
- [15] Baidu, Apollo Auto.
URL <https://apollo.auto/>
- [16] S. Shalev-Shwartz, S. Shammah, A. Shashua, On a formal model of safe and scalable self-driving cars, *ArXiv abs/1708.06374* (2017).
- [17] N. Kalra, S. M. Paddock, Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability?, *Transportation Research Part A: Policy and Practice* 94 (2016) 182–193.

- [18] ISO Standard, Road vehicles – functional safety, iSO 26262-1:2011 (2011).
- [19] P. Koopman, M. Wagner, Challenges in autonomous vehicle testing and validation, *SAE Int. J. Trans. Safety* 4 (1) (2016).
- [20] P. Koopman, U. Ferrell, F. Fratrick, M. Wagner, A safety standard approach for fully autonomous vehicles, in: *International Conference on Computer Safety, Reliability, and Security*, Springer, 2019, pp. 326–332.
- [21] Underwriters Laboratories, ANSI/UL 4600 Standard for Safety for the Evaluation of Autonomous Products.
URL <https://ul.org/UL4600>
- [22] F. Ingrand, Recent trends in formal validation and verification of autonomous robots software, in: *2019 Third IEEE International Conference on Robotic Computing (IRC)*, 2019, pp. 321–328.
- [23] A. Desai, T. Dreossi, S. Seshia, Combining model checking and runtime verification for safe robotics, in: *International Conference on Runtime Verification*, Springer, 2017, pp. 172–189.
- [24] B. Hoxha, G. Fainekos, Planning in dynamic environments through temporal logic monitoring, in: *AAAI Workshop: Planning for Hybrid Systems*, Vol. 16, 2016, p. 12.
- [25] C.-I. Vasile, J. Tumova, S. Karaman, C. Belta, D. Rus, Minimum-violation scLTL motion planning for mobility-on-demand, in: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2017, pp. 1481–1488.
- [26] S. Dathathri, R. Murray, Decomposing GR(1) games with singleton liveness guarantees for efficient synthesis, *arXiv abs/1709.07094* (2017).
- [27] S. Ghosh, D. Sadigh, P. Nuzzo, V. Raman, A. Donzé, A. L. Sangiovanni-Vincentelli, S. S. Sastry, S. A. Seshia, Diagnosis and repair for synthesis from signal temporal logic specifications, in: *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control, HSCC '16*, ACM, 2016, pp. 31–40.
- [28] W. Li, L. Dworkin, S. A. Seshia, Mining assumptions for synthesis, in: *Ninth ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMPCODE2011)*, 2011, pp. 43–50.
- [29] W. Li, D. Sadigh, S. Sastry, S. Seshia, Synthesis for human-in-the-loop control systems, in: *Intl. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2014.
- [30] M. Kloetzer, C. Belta, A fully automated framework for control of linear systems from temporal logic specifications, *IEEE Trans. on Automatic Control* 53 (1) (2008) 287–297.
- [31] S. Mitsch, K. Ghorbal, D. Vogelbacher, A. Platzer, Formal verification of obstacle avoidance and navigation of ground robots, *The International Journal of Robotics Research* 36 (12) (2017) 1312–1340.
- [32] N. Roohi, R. Kaur, J. Weimer, O. Sokolsky, I. Lee, Self-driving vehicle verification towards a benchmark, *arXiv preprint arXiv:1806.08810* (2018).
- [33] R. C. Cardoso, M. Farrell, M. Luckcuck, A. Ferrando, M. Fisher, Heterogeneous verification of an autonomous curiosity rover (2020) 353–360.
- [34] S. Jha, V. Raman, D. Sadigh, S. Seshia, Safe autonomy under perception uncertainty using chance-constrained temporal logic, *Journal of Automated Reasoning* 60 (2017) 43–62.
- [35] F. Pasqualetti, F. Dörfler, F. Bullo, Attack detection and identification in cyber-physical systems, *IEEE Transactions on Automatic Control* 58 (11) (2013) 2715–2729.

- [36] M. Foughali, B. Berthomieu, S. Dal Zilio, P.-E. Hladik, F. Ingrand, A. Mallet, Formal verification of complex robotic systems on resource-constrained platforms, in: 2018 IEEE/ACM 6th International FME Workshop on Formal Methods in Software Engineering (FormaliSE), 2018, pp. 2–9.
- [37] S. Seshia, D. Sadigh, Towards verified artificial intelligence, ArXiv abs/1606.08514 (2016).
- [38] M. Luckcuck, M. Farrell, L. A. Dennis, C. Dixon, M. Fisher, Formal specification and verification of autonomous robotic systems: A survey, ACM Computing Surveys (CSUR) 52 (5) (2019) 1–41.
- [39] T. Dreossi, D. Fremont, S. Ghosh, E. Kim, H. Ravanbakhsh, M. Vazquez-Chanlatte, S. Seshia, VERIFAI: A toolkit for the design and analysis of artificial intelligence-based systems, ArXiv:1902.04245 (2019).
- [40] D. J. Fremont, T. Dreossi, S. Ghosh, X. Yue, A. L. Sangiovanni-Vincentelli, S. A. Seshia, Scenic: a language for scenario specification and scene generation, in: Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, 2019, pp. 63–78.
- [41] K. Leahy, E. Cristofalo, C. Vasile, A. Jones, E. Montijano, M. Schwager, C. Belta, Control in belief space with temporal logic specifications using vision-based localization, Intl. J. of Robotics Research 38 (04 2019).
- [42] A. Balakrishnan, A. G. Puranic, X. Qin, A. Dokhanchi, J. V. Deshmukh, H. Ben Amor, G. Fainekos, Specifying and evaluating quality metrics for vision-based perception systems, in: Design, Automation Test in Europe Conference Exhibition (DATE), 2019, pp. 1433–1438.
- [43] A. Dokhanchi, H. B. Amor, J. Deshmukh, G. Fainekos, Evaluating perception systems for autonomous vehicles using quality temporal logic, in: Intl. Conf. on Runtime Verification (RV), 2018.
- [44] T. Dreossi, S. Ghosh, A. Sangiovanni-Vincentelli, S. Seshia, Systematic testing of convolutional neural networks for autonomous driving, ArXiv abs/1708.03309 (2017).
- [45] Y. Cao, C. Xiao, B. Cyr, Y. Zhou, W. Park, S. Rampazzi, Q. A. Chen, K. Fu, Z. M. Mao, Adversarial sensor attack on lidar-based perception in autonomous driving, in: Proceedings of the 2019 ACM SIGSAC conference on computer and communications security, 2019, pp. 2267–2281.
- [46] A. Boloor, K. Garimella, X. He, C. Gill, Y. Vorobeychik, X. Zhang, Attacking vision-based perception in end-to-end autonomous driving models, Journal of Systems Architecture 110 (2020) 101766.
- [47] H. Delecki, M. Itkina, B. Lange, R. Senanayake, M. J. Kochenderfer, How do we fail? stress testing perception in autonomous vehicles, arXiv preprint arXiv:2203.14155 (2022).
- [48] N. Akhtar, A. Mian, Threat of adversarial attacks on deep learning in computer vision: A survey, Ieee Access 6 (2018) 14410–14430.
- [49] Q. M. Rahman, P. Corke, F. Dayoub, Run-time monitoring of machine learning for robotic perception: A survey of emerging trends, IEEE Access 9 (2021) 20067–20075.
- [50] Q. M. Rahman, N. Sünderhauf, P. Corke, F. Dayoub, Fsnet: A failure detection framework for semantic segmentation, Vol. 7, 2022, pp. 3030–3037. doi:10.1109/LRA.2022.3143219.
- [51] J. Lambert, J. Hays, Trust, but verify: Cross-modality fusion for hd map change detection, in: Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2), 2021.
- [52] J. Liu, J.-M. Park, “seeing is not always believing”: Detecting perception error attacks against autonomous vehicles, IEEE Transactions on Dependable and Secure Computing 18 (5) (2021) 2209–2223.

- [53] A. Sharma, N. Azizan, M. Pavone, Sketching curvature for efficient out-of-distribution detection for deep neural networks, in: *Uncertainty in Artificial Intelligence*, PMLR, 2021, pp. 1958–1967.
- [54] C. You, Z. Hau, S. Demetriou, Temporal consistency checks to detect lidar spoofing attacks on autonomous vehicle perception, in: *Proceedings of the 1st Workshop on Security and Privacy for Mobile AI*, 2021, pp. 13–18.
- [55] A. Balakrishnan, J. Deshmukh, B. Hoxha, T. Yamaguchi, G. Fainekos, Percemon: Online monitoring for perception systems, in: *International Conference on Runtime Verification*, Springer, 2021, pp. 297–308.
- [56] D. Kang, D. Raghavan, P. Bailis, M. Zaharia, Model assertions for debugging machine learning, in: *NIPS*, 2018.
- [57] B. Cai, L. Huang, M. Xie, Bayesian networks in fault diagnosis, *IEEE Transactions on industrial informatics* 13 (5) (2017) 2227–2240.
- [58] A. Abdollahi, K. R. Pattipati, A. Kodali, S. Singh, S. Zhang, P. B. Luh, Probabilistic graphical models for fault diagnosis in complex systems, in: *Principles of Performance and Reliability Modeling and Evaluation*, Springer, 2016, pp. 109–139.
- [59] Y. Lei, B. Yang, X. Jiang, F. Jia, N. Li, A. K. Nandi, Applications of machine learning to machine fault diagnosis: A review and roadmap, *Mechanical Systems and Signal Processing* 138 (2020) 106587.
- [60] X. Ma, J. Wu, S. Xue, J. Yang, C. Zhou, Q. Z. Sheng, H. Xiong, L. Akoglu, A comprehensive survey on graph anomaly detection with deep learning, *IEEE Transactions on Knowledge and Data Engineering* (2021).
- [61] J. De Kleer, B. C. Williams, Diagnosing multiple faults, *Artificial intelligence* 32 (1) (1987) 97–130.
- [62] F. P. Preparata, G. Metze, R. T. Chien, On the connection assignment problem of diagnosable systems, *IEEE Transactions on Electronic Computers* (6) (1967) 848–854.
- [63] S. L. Hakimi, A. T. Amin, Characterization of connection assignment of diagnosable systems, *IEEE Transactions on Computers* 100 (1) (1974) 86–88.
- [64] K. Bhat, Algorithms for finding diagnosability level and t-diagnosis in a network of processors, in: *Proceedings of the ACM’82 conference*, 1982, pp. 164–168.
- [65] A. T. Dahbura, System-level diagnosis: A perspective for the third decade, in: *Concurrent Computations*, Springer, 1988, pp. 411–434.
- [66] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, D. Teneketzis, Diagnosability of discrete-event systems, *IEEE Transactions on automatic control* 40 (9) (1995) 1555–1575.
- [67] J. Zaytoon, S. Lafortune, Overview of fault diagnosis methods for discrete event systems, *Annual Reviews in Control* 37 (2) (2013) 308–320.
- [68] T. M. Tuxi, L. K. Carvalho, E. V. Nunes, A. E. da Cunha, Diagnosability verification using ltl model checking, *Discrete Event Dynamic Systems* (2022) 1–35.
- [69] H. Yang, L. Carlone, One ring to rule them all: Certifiably robust geometric perception with outliers, in: *Conf. on Neural Information Processing Systems (NeurIPS)*, Vol. 33, 2020, pp. 18846–18859, ([pdf](https://proceedings.neurips.cc/paper/2020/file/da6ea77475918a3d83c7e49223d453cc-Paper.pdf)).
URL <https://proceedings.neurips.cc/paper/2020/file/da6ea77475918a3d83c7e49223d453cc-Paper.pdf>
- [70] H. Yang, L. Carlone, In perfect shape: Certifiably optimal 3D shape reconstruction from 2D landmarks, in: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020, arxiv version: 1911.11924, ([pdf](#)).

- [71] H. Yang, L. Carlone, A polynomial-time solution for robust registration with extreme outlier rates, in: Robotics: Science and Systems (RSS), 2019, ([pdf](#)), ([video](#)), ([media](#)), ([media](#)), ([media](#)).
- [72] Google, Google OR-Tools.
URL <https://developers.google.com/optimization>
- [73] F. Dellaert, M. Kaess, Factor graphs for robot perception, Foundations and Trends in Robotics 6 (1-2) (2017) 1–139.
- [74] Z. Zhang, F. Wu, W. S. Lee, Factor graph neural networks, Advances in Neural Information Processing Systems 33 (2020) 8577–8587.
- [75] W. L. Hamilton, R. Ying, J. Leskovec, Representation learning on graphs: Methods and applications, IEEE Bulletin of the Tech. Committee on Data Engineering 40 (3) (2017) 52–74.
- [76] S. Nowozin, C. H. Lampert, Structured learning and prediction in computer vision, Vol. 6, Now publishers Inc, 2011.
- [77] T. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, in: Intl. Conf. on Learning Representations (ICLR), 2017.
- [78] M. Chen, Z. Wei, Z. Huang, B. Ding, Y. Li, Simple and deep graph convolutional networks, in: International Conference on Machine Learning, PMLR, 2020, pp. 1725–1735.
- [79] K. Xu, W. Hu, J. Leskovec, S. Jegelka, How powerful are graph neural networks?, in: Intl. Conf. on Learning Representations (ICLR), 2019.
- [80] W. H. L., R. Ying, J. Leskovec, Inductive representation learning on large graphs, in: Advances in Neural Information Processing Systems (NIPS), 2017, p. 1025–1035.
- [81] K. P. Murphy, Machine learning: a probabilistic perspective, MIT press, 2012.
- [82] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al., Pytorch: An imperative style, high-performance deep learning library, Advances in neural information processing systems 32 (2019).
- [83] P. Antonante, D. Spivak, L. Carlone, Monitoring and diagnosability of perception systems, arXiv preprint arXiv: 2011.07010([pdf](#)) (2020).
- [84] P. Antonante, H. Nilsen, L. Carlone, Monitoring of perception systems: Deterministic, probabilistic, and learning-based fault detection and identification, arXiv preprint arXiv: 2205.10906([pdf](#)) (2022).
- [85] Baidu, Apollo Auto.
URL <https://github.com/ApolloAuto/apollo>
- [86] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, J. Leonard, Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age, IEEE Trans. Robotics 32 (6) (2016) 1309–1332, arxiv preprint: 1606.05830, ([pdf](#)). doi: 10.1109/TR0.2016.2624754.
- [87] H. Yang, J. Shi, L. Carlone, TEASER: Fast and Certifiable Point Cloud Registration, IEEE Trans. Robotics 37 (2) (2020) 314–333, extended arXiv version 2001.07715 ([pdf](#)).
- [88] H. Yang, L. Carlone, Certifiably optimal outlier-robust geometric perception: Semidefinite relaxations and scalable global optimization, IEEE Trans. Pattern Anal. Machine Intell. ([pdf](#)) (2021).
- [89] P. Antonante, D. Spivak, L. Carlone, Monitoring and diagnosability of perception systems, in: IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS), 2021, ([pdf](#)).
- [90] Z. Liu, Z. Wu, R. Tóth, SMOKE: Single-stage monocular 3d object detection via keypoint estimation, arXiv preprint arXiv:2002.10111 (2020).

- [91] P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine, et al., Scalability in perception for autonomous driving: Waymo open dataset, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 2446–2454.
- [92] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, O. Beijbom, Pointpillars: Fast encoders for object detection from point clouds, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 12697–12705.
- [93] Google’s self-driving startup Waymo is introducing fully driverless rides to San Francisco, https://www.continental-automotive.com/getattachment/5430d956-1ed7-464b-afa3-cd9cdc98ad63/ARS408-21_datasheet_en_170707_V07.pdf.pdf, accessed: 2022-05-15.
- [94] D. F. Crouse, On implementing 2d rectangular assignment algorithms, IEEE Transactions on Aerospace and Electronic Systems 52 (4) (2016) 1679–1696.
- [95] W. Falcon, et al., Pytorch lightning, <https://github.com/PytorchLightning/pytorch-lightning> (2019).
- [96] L. N. Smith, Cyclical learning rates for training neural networks, in: 2017 IEEE winter conference on applications of computer vision (WACV), IEEE, 2017, pp. 464–472.
- [97] The Guardian, Tesla driver dies in first fatal crash while using autopilot mode, www.theguardian.com/technology/2016/jun/30/tesla-autopilot-death-self-driving-car-elon-musk, accessed: 2022-05-15 (2016).

A Proofs

A.1 Proof for Theorem 7

Proof. Let \mathbf{z} be a syndrome for the temporal diagnostic graph $\mathcal{D}^{[K]}$, generated by a set of active failure mode \mathcal{A} , such that $|\mathcal{A}| = m \leq \min_{i \in \{1, \dots, K\}} \kappa(\mathcal{D}^{(i)})$. Clearly, each element of \mathcal{A} is a variable node of one of the regular diagnostic graphs $\mathcal{D}^{(1)}, \dots, \mathcal{D}^{(K)}$ that compose $\mathcal{D}^{[K]}$, therefore we can split \mathcal{A} into the variables nodes of each regular diagnostic graph, obtaining $\mathcal{A}^{(1)}, \dots, \mathcal{A}^{(K)}$ (these sets are non-overlapping and are such that $\bigcup_{i=1}^K \mathcal{A}^{(i)} = \mathcal{A}$). Similarly, we can project the syndrome \mathbf{z} into K sub-syndromes $\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(K)}$ each containing only the test outcomes of the corresponding regular diagnostic graphs (notice that doing the projection we lose the temporal tests, if any). By construction $|\mathcal{A}^{(i)}| \leq m$ for each $i = 1, \dots, K$. From the assumption, we know that each sub-graph $\mathcal{D}^{(i)}$ is m -diagnosable. Therefore, each sub-graph $\mathcal{D}^{(i)}$ will be able to correctly identify the set of active failure modes $\mathcal{A}^{(i)}$ from the syndrome $\mathbf{z}^{(i)}$. This means that $\mathcal{D}^{[K]}$ is at least m -diagnosable, concluding the proof. \square

A.2 Proof for Theorem 9

Proof. For each sample $(\mathbf{z}^{(i)}, \mathbf{f}^{(i)})$ in \mathcal{W} , the result of each Hamming distance will be less or equal than N_f . From the Hoeffding's inequality we have that

$$\Pr \left[|h_{\mathcal{F}}(\Psi_{\mathcal{D}}) - \hat{h}_{\mathcal{W}}(\Psi_{\mathcal{D}})| \geq \epsilon \right] \leq 2 \exp \left(-\frac{2\epsilon^2 |\mathcal{W}|}{N_f^2} \right) \quad (1)$$

Setting the right-hand side of Eq. (1) to be equal to δ and solving for ϵ yields:

$$\epsilon = N_f \sqrt{\frac{\log(2/\delta)}{2|\mathcal{W}|}} \quad (2)$$

After setting the right-hand side to δ , Eq. (1) can be rewritten as:

$$\Pr \left[|h_{\mathcal{F}}(\Psi_{\mathcal{D}}) - \hat{h}_{\mathcal{W}}(\Psi_{\mathcal{D}})| \leq \epsilon \right] \geq \delta \quad (3)$$

Combining (2) and (3) and removing the absolute value we get:

$$\Pr \left[h_{\mathcal{F}}(\Psi_{\mathcal{D}}) - \hat{h}_{\mathcal{W}}(\Psi_{\mathcal{D}}) \leq N_f \sqrt{\frac{\log(2/\delta)}{2|\mathcal{W}|}} \right] \geq \delta \quad (4)$$

from which the result easily follows. \square

B Examples of Diagnostic Tests

In the following, we describe how to mathematically model the relation between the failure modes and the test outcomes; this will be instrumental in solving the inverse problem of identifying the failure mode from a given syndrome. We provide a deterministic and a probabilistic model for the tests below.

Deterministic Tests. Deterministic diagnostic tests encode the set of possible test outcomes, by establishing a deterministic relation between failure modes in the test's scope and the test outcome. We discuss potential models for deterministic diagnostic test below.

Ideally we would like the test be *reliable*, that is, to return FAIL if and only if at least one of the failure modes in its scope is active. This leads to the definition of a "Deterministic OR" test.

Definition 10 (Deterministic OR). *A diagnostic test $t(\mathbf{f}_{\text{scope}(t)})$ is a deterministic OR if its test outcome z is*

$$z = \begin{cases} \text{PASS} & \text{if } \|\mathbf{f}_{\text{scope}(t)}\|_1 = 0 \\ \text{FAIL} & \text{otherwise} \end{cases} \quad (5)$$

This kind of tests can be hard to implement in practice. For example, imagine a diagnostic test that compares the output of two object classifiers: if one of them produces a wrong label, it is easy to detect there is a failure; however, if both classifiers are trained on similar data and both report the incorrect label there is no way to detect the failure. In this case, the test outcome is unreliable. The following definition introduces a type of unreliable test.

Definition 11 (Deterministic Weak-OR). *A test $t(\mathbf{f}_{\text{scope}(t)})$ is a deterministic Weak-OR if its test outcome z is*

$$z = \begin{cases} \text{FAIL} & \text{if } 0 < \|\mathbf{f}_{\text{scope}(t)}\|_1 < |\text{scope}(t)| \\ \text{PASS or FAIL} & \text{if } \|\mathbf{f}_{\text{scope}(t)}\|_1 = |\text{scope}(t)| \\ \text{PASS} & \text{otherwise} \end{cases} \quad (6)$$

Intuitively, a “Deterministic Weak-OR” may return PASS even if all failure modes are active, since the test might fail to detect an inconsistency if all faults are consistent with each others (again, think about two object classifiers failing in the same way). Even though the Weak-OR test may pass or fail when all failure modes are active, its outcome remains deterministic.

Probabilistic Tests. Deterministic tests might not capture the complexity of real world diagnostic tests. Most practical tests are likely to incorrectly detect faults (*i.e.*, produce false positive) or fail to detect faults (*i.e.*, produce false negatives) with some probability. For this reason, in this paper, we also allow for an arbitrary probabilistic relationship between test outcomes and failure modes in the test scope.

A simple-yet-expressive way to formalize a probabilistic test is to use what we call a “Noisy-OR” model. In particular, the Noisy-OR model represents the probability of a diagnostic test outcome as a conditional probability distribution over the failure modes in its scope $\Pr(z \mid \mathbf{f}_{\text{scope}(t)})$ as defined below.

Definition 12 (Noisy-OR). *A diagnostic test $t(\mathbf{f}_{\text{scope}(t)})$ is a probabilistic Noisy-OR if its test outcome z follows*

$$\Pr(z = \text{PASS} \mid \mathbf{f}_{\text{scope}(t)}) = \prod_{i \in \text{scope}(t)} \Pr(z = \text{PASS} \mid f_i) \quad (7)$$

where $\Pr(z \mid f_i)$ denotes the conditional probability of the test outcome (PASS/FAIL) conditioned on the status (ACTIVE/INACTIVE) of the failure mode f_i . Clearly, $\Pr(z = \text{FAIL} \mid \mathbf{f}_{\text{scope}(t)}) = 1 - \Pr(z = \text{PASS} \mid \mathbf{f}_{\text{scope}(t)})$.

Now suppose each test has a probability $p_{d,i}$ of correctly identifying failure f_i (detection probability), and a probability $p_{a,i}$ of false alarm for f_i . Exploiting the fact that $f_i \in \{0, 1\}$, we can write Eq. (7) as:

$$\Pr(z = \text{PASS} \mid \mathbf{f}_{\text{scope}(t)}) = \prod_{i \in \text{scope}(t)} (1 - p_{d,i})^{f_i} (1 - p_{a,i})^{1-f_i} \quad (8)$$

An example of probabilistic test outcome is given in Table 2.

C Examples of Diagnostic Graphs

C.1 Example 1: Multi-sensor Obstacle Detection

Consider the perception system in Fig. 4. We can associate a diagnostic graph to the system where the variable nodes of the diagnostic graph are the failure modes of modules and outputs in the system. The diagnostic graph, shown in Fig. 5, also includes two diagnostic tests and a priori relations encoding input/output relationship between modules and outputs.

Each diagnostic test compares a pair of outputted obstacles, namely LiDAR obstacles and camera obstacles (with failures f_4 and f_5), and camera obstacles and fused obstacles (with failures f_4 and f_6).

Scope		Test outcome z		
f_1	f_2	OR	Weak-OR	Noisy-OR
0	0	0	0	$\begin{cases} 0 \text{ with prob. } (1 - p_{a,1})(1 - p_{a,2}) \\ 1 \text{ with prob. } p_{a,1} + p_{a,2} - p_{a,1}p_{a,2} \end{cases}$
0	1	1	1	$\begin{cases} 0 \text{ with prob. } (1 - p_{a,1})(1 - p_{d,2}) \\ 1 \text{ with prob. } p_{a,1} + p_{d,2} - p_{a,1}p_{d,2} \end{cases}$
1	0	1	1	$\begin{cases} 0 \text{ with prob. } (1 - p_{d,1})(1 - p_{a,2}) \\ 1 \text{ with prob. } p_{d,1} + p_{a,2} - p_{d,1}p_{a,2} \end{cases}$
1	1	1	0 or 1	$\begin{cases} 0 \text{ with prob. } (1 - p_{d,1})(1 - p_{d,2}) \\ 1 \text{ with prob. } p_{d,1} + p_{d,2} - p_{d,1}p_{d,2} \end{cases}$

Figure 3: A test comparing two outputs, LiDAR Obstacles and Camera Obstacles

Table 2: Table of possible outcomes for the Deterministic OR and the probabilistic Noisy-OR version of a test with scope f_1 and f_2 .

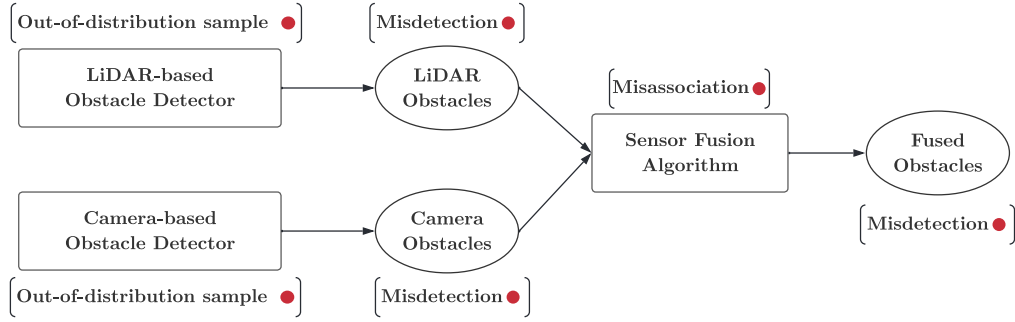


Figure 4: Perception system including 3 modules (rectangles) and 3 outputs (circles). Modules are connected by edges describing which module produces or consumes a given output. The failure modes of each module (resp. output) are represented by red dots.

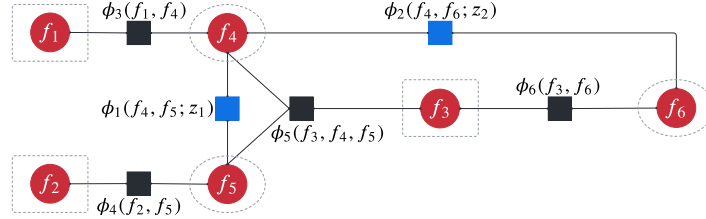


Figure 5: A diagnostic graph for the perception system example in Fig. 4. Red circles represent variable nodes (failure modes) while squares represent relations. Test-driven Relations are shown in blue, while a priori relations are shown in black.

C.2 Example 2: LiDAR-based Ego-motion Estimation

We provide a second example that also includes singleton diagnostic tests (having a single failure mode in their scope) and includes explicit tests over modules. The example consists of a LiDAR-based odometry system that computes the relative motion between consecutive LiDAR scans using feature-based registration, see *e.g.*, [86, 87]. The system \mathcal{S} comprises two modules, a *feature extraction* module and a *point-cloud registration* module, as depicted in Fig. 6(left). The feature extraction module extracts 3D point features from input LiDAR data, while the point-cloud registration module uses the features to estimate the relative pose between two consecutive LiDAR scans. Suppose that the feature extraction module is based on a deep neural network and that it can experience an “out-of-distribution sample” failure, which causes the corresponding output to potentially experience “too-many outliers” or “few features” failures. Similarly, the module *point-cloud registration* can experience the failure “suboptimal solution”, which leads its outputs, the relative pose, to experience

a “wrong relative pose” failure. Fig. 6(right) shows a diagnostic graph for the system. The system is equipped with three diagnostic tests. A diagnostic test detects if the failure mode “few features” is active by checking the cardinality of the feature set. If the point-cloud registration module is a certifiable algorithm [88], we can attach a diagnostic test to the point-cloud registration module that uses the module’s certificate to detect if the module is experiencing a “suboptimal solution” failure. Another diagnostic test detects if the relative pose is wrong by checking that the relative pose does not exceed some meaningful threshold given the vehicle dynamics. Finally, another test checks if under the computed relative pose, the feature extractor has “too many outliers”. This can be achieved by counting the number of features that are correctly aligned after applying the estimated relative pose. The diagnostic graph also contains a priori relations encoding constraints on the input/output relationships.

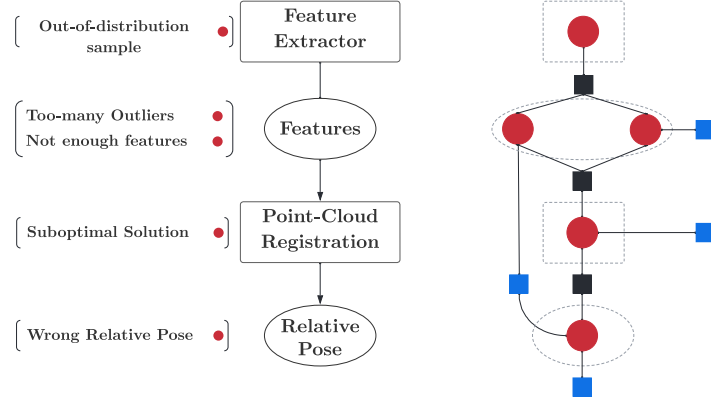


Figure 6: (Left) Example of the LiDAR-based ego-motion estimation system S . The system is composed by two modules (rectangles), each producing one output (circles). (Right) The corresponding diagnostic graph, where red circles represent variable nodes (failure modes) while squares represent relations (test-driven Relations in blue, a priori relations in black).

D Implementation Details

D.1 Apollo Auto

Baidu’s Apollo Auto [15] uses a flexible and modularized architecture for the autonomy stack based on the sense-plan-act framework. The stack includes seven subsystems: (i) the *localization subsystem* provides the pose of the ego vehicle; (ii) the *high-definition map* provides a high-resolution map of the environment, including lanes, stop signs, and traffic signs; (iii) the *perception subsystem* processes sensory information (together with the localization data) and creates a world model; (iv) the *prediction subsystem* predicts future evolution of the world state; (v) the *motion planning subsystems* and (vi) the *routing subsystem* generate a feasible trajectory for the ego vehicle, and finally, (vii) the *control subsystem* generates low-level control signals to move the vehicle. In our experiments, we focus on the *perception subsystem*, to which we apply our runtime monitors. In the following, we briefly review the key aspects of the Apollo Auto perception system.

D.1.1 Apollo Auto Perception System

Apollo Auto’s perception system is tasked with the detection and classification of obstacles and traffic lights.² The perception module is capable of using multiple cameras, radars, and LiDARs to recognize obstacles. There is a submodule for each sensor modality, that independently detects, classifies, and tracks obstacles. The results from each sub-module are then fused using a probabilistic sensor fusion algorithm.

Obstacle Detection. Obstacles such as cars, trucks, bicycles, are detected using an array of radars, LiDARs, and cameras. Each obstacle is represented by a 3D bounding-box in the world frame,

²Note that our monitors can be also applied to other perception-related subsystems, such as the localization and high-definition map subsystem, see [89] for an example.

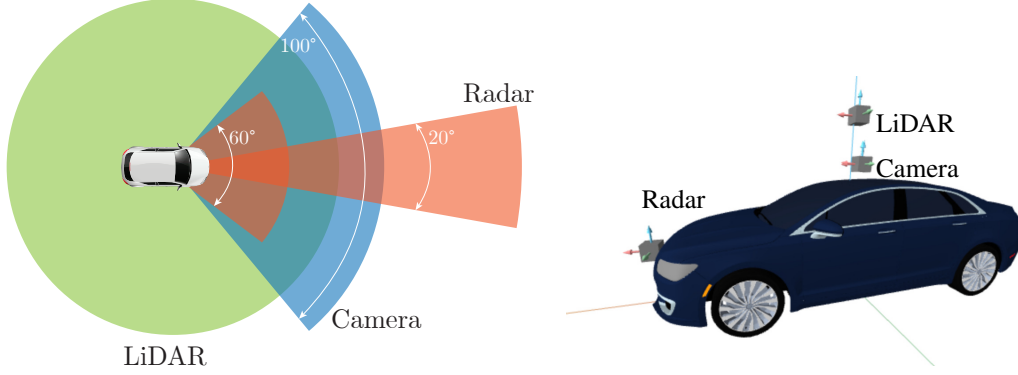


Figure 7: Vehicle configuration and sensor field-of-view (FOV). LiDAR FOV is shown in green, the camera FOV in blue and the radar FOV in orange.

the class of the object, a confidence score, together with other sensor-specific information (*e.g.*, the velocity of the obstacle). Each sensor is processed as follows:

Camera: The camera-based obstacle detection network is based on the monocular object detection SMOKE [90] and trained on the Waymo Open Dataset [91]. The network predicts 2D and 3D information about each obstacle, and then a post-processing step predicts the 3D bounding box of each obstacle by minimizing the reprojection error of available templates for the predicted obstacle class;

LiDAR: The LiDAR-based obstacle detection network, called Mask-Pillars [92], but enhanced with a residual attention module to improve detection in case of occlusion;

Radar: Apollo Auto uses directly the obstacles detections reported by the radar (assumed to have an embedded detector [93]), that are post-processed to be transformed to the world frame.

D.1.2 Vehicle Configuration

The simulated vehicle is a Lincoln MKZ with one Velodyne VLS-128 LiDAR, one front-facing camera with a field-of-view of 50° , one front-facing telephoto camera (pointed 4° upwards) for traffic light detection and recognition, one Continental ARS 408-21 front-facing radar, GPS, and IMU.

We ran the Baidu’s Apollo AV stack on a computer with an Intel i9-9820X (4.1 GHz) processor, 64 GB of memory and two NVIDIA GeForce RTX 2080Ti. The simulator ran on a computer with 11th Generation Intel i7-11700F (4.8 GHz) processor, 16 GB of memory, and an NVIDIA GeForce RTX 3060. The two computers were connected using a Gigabit Ethernet cable.

D.2 Diagnostic Graph

We focused our attention on the obstacle detection pipeline. The system we aim to monitor, together with the failure modes considered, is shown in Fig. 8 The system is composed of four modules:

- Lidar-based Obstacle detector, based on a deep learning algorithm, subject to *out-of-distribution sample* failure mode;
- Camera-based Obstacle detector, based on a deep learning algorithm, subject to *out-of-distribution sample* failure mode;
- Radar-based Obstacle detector subject to *misdetetection* failure mode;
- Sensor Fusion subject to *misassociation* failure mode.

Each module produces a set of detected obstacles. We identified three failure modes for each set of detected obstacles:

- *misdetetection*: the module detected a ghost obstacle or is missing an obstacle in the scene;

- *misposition*: the module detected the obstacle correctly, but its position is incorrect (*i.e.*, more than 2.5m error in our tests);
- *misclassification*: the module detected the obstacle correctly but the obstacle’s semantic class is incorrect.

We equipped the obstacle detection system with 18 diagnostic tests. For each pair of modules’ outputs, namely (Lidar, Camera), (Radar, Camera), (Lidar, Sensor Fusion), (Radar, Sensor Fusion), (Lidar, Radar), and (Camera, Sensor Fusion), there is a test that compares the outputs to diagnose each of the output’s failure modes (*i.e.*, misdetection, misposition, and misclassification). Intuitively, each test compares the two sets of obstacles coming from the corresponding modules, and if they are different, it reports if the inconsistency was due to a misdetection, misposition, or misclassification. Moreover, we included a priori relation between every module and its output. In particular, the modules are assumed to fail if their outputs have at least one active failure mode. In the probabilistic diagnostic graph we also added an a priori relation for each module’s failure mode, indicating the prior probability of that failure mode being active.

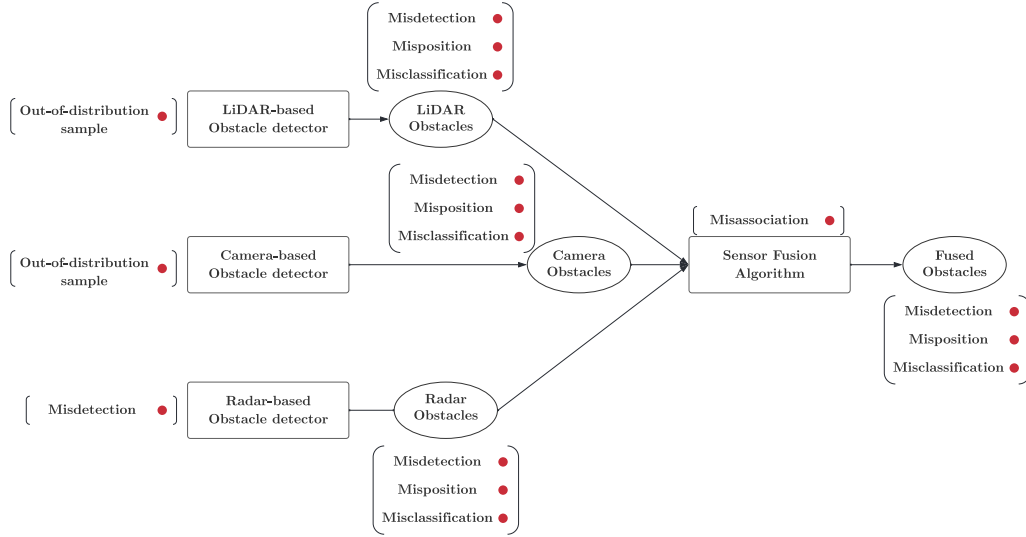


Figure 8: Perception system considered in our experiments. Modules are shown as rectangular blocks, outputs are shown as rounded boxes, while failure modes are denoted with red dots.

D.2.1 Diagnostic Tests

We now describe the logic for the diagnostic tests we implemented. Consider two sets of synchronized detected obstacles³, say \mathcal{A} and \mathcal{B} , produced by two modules, using some sensor data. Let Ω be the region defined by the intersection of both sensor fields of view and a region of interest (*e.g.*, a region close to a drivable area⁴). Denote by \mathcal{A}_Ω and \mathcal{B}_Ω the set of obstacles restricted to the region Ω , namely $\mathcal{A}_\Omega \subseteq \mathcal{A}$ such that for each obstacle o in \mathcal{A} , o is in \mathcal{A}_Ω if and only if o is inside the region defined by Ω . The same relation holds for \mathcal{B}_Ω . Then the diagnostic test checking for misdetections is defined as follows:

$$t_{\text{misdetection}} = \begin{cases} \text{FAIL} & \text{if } |\mathcal{A}_\Omega| \neq |\mathcal{B}_\Omega| \\ \text{PASS} & \text{otherwise} \end{cases}$$

Note that if the two sets of obstacles have a different cardinality —when restricted to the area co-visible by both sensors— it means that one of the two sets contains a ghost obstacle or one of the two sets is missing an obstacle. From a single test, we are not able to say which of the two sets is experiencing the misdetection, but we know at least one output did.

³By synchronized we mean that the two outputs are produced at the same time instant.

⁴In our experiments, the region of interest is the area within 5 meters from a drivable lane.

Let us now move our attention to the misposition failure mode. Let \mathcal{C} be the set of *matched* obstacles, that is, a pair of obstacles (l, r) —with $l \in \mathcal{A}_\Omega$ and $r \in \mathcal{B}_\Omega$ — is in \mathcal{C} , if l and r represent the same obstacles. A common approach for finding the set of matches is to select all the pairs that are closest to each other (*i.e.*, solving an assignment problem)⁵. The diagnostic test checking for mispositioned obstacles is defined as follows:

$$t_{\text{misposition}} = \begin{cases} \text{FAIL} & \exists (l, r) \in \mathcal{C} \text{ such that } |\text{pos}(l) - \text{pos}(r)| \geq \theta \\ \text{PASS} & \text{otherwise} \end{cases}$$

where $\text{pos}(\cdot)$ is the position of an obstacle and θ is an error threshold, chosen as $\theta = 2.5$ m in our experiments.

Finally, the test checking for misclassified obstacles is defined as follows:

$$t_{\text{misclassification}} = \begin{cases} \text{FAIL} & \exists (l, r) \in \mathcal{C} \text{ such that } \text{cls}(l) \neq \text{cls}(r) \\ \text{PASS} & \text{otherwise} \end{cases}$$

where $\text{cls}(\cdot)$ is the class of the obstacle, *i.e.*, the test fails if associated obstacles are assigned different semantic classes.

D.2.2 Temporal Diagnostic Graph

To build a temporal diagnostic graph we stack 2 regular diagnostic graphs into a temporal diagnostic graph. In the probabilistic case, each module failure mode is connected to its successive (in time) via a priori relationships, which represent the transition probability between states in consecutive time steps. No temporal a priori relations are added in the deterministic case. We also added temporal tests. The logic of the tests presented in the previous section is applicable to temporal tests with small changes. In temporal tests, the sets \mathcal{A} and \mathcal{B} are not time-synchronized anymore (*e.g.*, they are obstacles detected by the same sensor at consecutive time stamps), therefore the position of each obstacle in each set must be adjusted for the distance the obstacle traveled between consecutive detections. To use the tests described earlier in the temporal domain we used the following approach. If the obstacle is equipped with an estimated velocity vector, since the time difference between detections is usually below 30 ms, we assume constant speed and integrate the speed over the time interval to find an approximate position of each obstacle. When the velocity is not available, we use the average speed of an obstacle (for a given obstacle's class) and adapt the misposition threshold θ to account for the uncertainty.

D.3 Graph-Neural-Network-based Fault Identification

Diagnostic Graphs to Undirected Graphs. In order to apply GNNs to our diagnostic graph \mathcal{D} , we need to convert $\mathcal{D} = (\mathcal{V}_\mathcal{D}, \mathcal{R}_\mathcal{D}, \mathcal{E}_\mathcal{D})$ into an undirected graph $G = (\mathcal{V}_G, \mathcal{E}_G)$. Towards this goal, we take the set of nodes \mathcal{V}_G to be *both* the set of failure modes and diagnostic test outcomes. Note that we add the diagnostic test outcomes as nodes in the graph since this allows attaching the test outcomes as features to these nodes. For each test t_k we form a clique⁶ involving the set of nodes in the test's scope and the variable corresponding to the test z_k , namely the set $\text{scope}(t_k) \cup \{z_k\}$. We then form another clique for each a priori relation $\phi_j \in \mathcal{R}_{\text{prior}}$ using the set of failure modes $\mathcal{N}(\phi_k)$ connected to ϕ_j . For example if we have a factor $\phi(f_1, f_2; z_2)$ we add the following (undirected) edges to \mathcal{E}_G : $(f_1, f_2), (f_1, z_2), (f_2, z_2)$. We attach a feature vector to each node in the graph. For the test nodes, we use a one-hot encoding describing the test outcome as node feature. For the module nodes, we use the failure probability (computed from the training data) as node features. We provide more details on the node features in Section 6.

We now discuss how we set the feature vector for each node in the graph. The feature $\mathbf{x}_{t_k} \in \mathbb{R}^2$ for a test t_k is set as the one-hot encoding of the test outcome (*i.e.*, $[1 \ 0]$ if the test passed, $[0 \ 1]$ if it failed). For the failure mode nodes we do not have any measurable quantity at runtime; we therefore use the training dataset to compute the feature vectors. In particular the feature vector $\mathbf{x}_{f_i} \in \mathbb{R}^2$ for a failure mode f_i is computed as follow: let ρ_i be the empirical probability that f_i is ACTIVE, *i.e.*,

⁵We matched obstacles using a generalization of the Hungarian algorithm [94], with the cost of each match being the Euclidean distance between obstacles.

⁶A clique is a subset of vertices of an undirected graph such that every two distinct vertices in the clique share an edge.

$\rho_i = \frac{1}{|\mathcal{W}|} \sum_{(z, f) \in \mathcal{W}} \mathbb{1}[f_i = \text{ACTIVE}]$; then the feature vector is chosen as $\mathbf{x}_{f_i} = [1 - \rho_i, \rho_i]^\top$. Intuitively, the feature describes the prior probability of the failure mode f_i 's state.

Network Architectures. We now discuss the architecture of the GNN. Our GNN is composed by a linear layer that embeds the feature vectors in \mathbb{R}^{16} , followed by a ReLU function. The output is then passed to a stack of graph convolution layers interleaved with ReLU activation functions. We tested four different graph convolution layers

- in the case of GCN, we stack 3 layers with 16 hidden channels each;
- in the case of GCNII, we stack 64 layers with 16 hidden channels each with $\alpha = 0.1$, $\beta = 0.4$;
- in the case of GIN, we stack 3 layers with 16 hidden channels each with the update function $\zeta^{(k)}(\cdot)$ being a 2-layer perceptron for $k = 1, \dots, 3$;
- in the case of GraphSAGE, we stack 3 (and 6 for temporal diagnostic graphs) layers with mean aggregator and 16 hidden channels each.

Finally, the readout function that converts the graph embedding to node labels is a linear layers followed by a softmax pooling.

We implemented the GNNs in PyTorch [82] and trained them on the training dataset for 100 epochs using the Adam optimizer. To reduce the amount of guesswork in choosing an initial learning rate, we used the learning rate finder available in the PyTorch Lightning library [95]. The procedure is based on [96]: the learning rate finder does a small training run where the learning rate is increased after each processed batch and the corresponding loss is logged. Then, the learning rate is chosen to be the point with the steepest negative gradient.

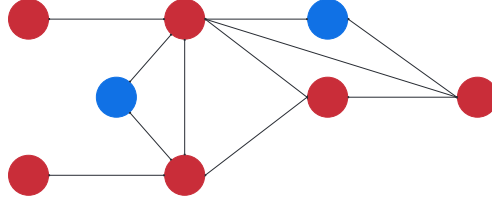


Figure 9: Example of conversion of the diagnostic graph in Fig. 5 into an undirected graph.

Learning to Identify Active Faults. In order to train the GNN to identify active faults, we use a supervised learning approach. In particular, we use a softmax classification function and negative log-likelihood training loss, which is available in standard libraries, such as PyTorch [82].

E Additional Results

E.1 Regular diagnostic graphs

Here we report the results of fault detection, identification and PAC-diagnosability bounds for the proposed methods on regular diagnostic graph. Table 3 shows the accuracy of fault detection and identification, together with the average runtime in milliseconds.

Comparing Fig. 1 to Fig. 10, we see steep decrease in the fault identification precision in the output space in regular diagnostic graphs compared to the temporal diagnostic graph. The best-performing model goes from around 97 % precision of the temporal diagnostic graph to 90 % of the regular diagnostic graph.

E.2 Example Scenario: Using Monitoring to Prevent Accidents

We conclude the experimental section by showing how fault detection and identification can be effectively used to prevent dangerous situations. To this aim, we developed an additional scenario (not included in Table 4) where a deer crosses the road while the ego vehicle cruises on a straight road (Fig. 12).

Algorithm	Fault Identification Accuracy			Fault Detection Accuracy			Timing Milliseconds
	All	Outputs	Modules	All	Outputs	Modules	
Factor Graph	93.30	96.72	83.03	76.67	88.48	64.85	0.79
Deterministic	91.06	93.69	83.18	89.09	89.09	89.09	3.25
Baseline (w/rel. scores)	92.39	94.65	85.61	89.09	89.09	89.09	0.1
Baseline	84.85	89.09	72.12	89.09	89.09	89.09	0.1
GCN	92.27	96.01	81.06	71.82	86.06	57.58	0.63
GCNII	87.61	93.94	68.64	68.48	87.88	49.09	19.88
GIN	91.89	96.06	79.39	83.94	86.06	81.82	0.48
GraphSage	92.84	96.46	81.97	76.67	89.09	64.24	0.59

Table 3: Results of the proposed algorithms for detection and identification on regular diagnostic graphs, averaged over all scenarios. Best values are highlighted in green, second-best in yellow.

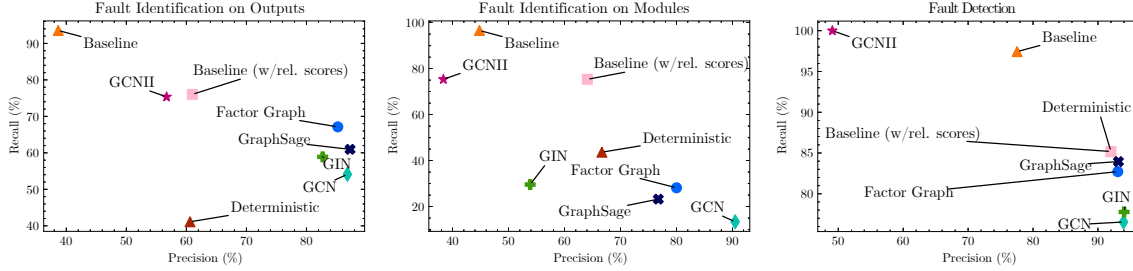


Figure 10: Precision/Recall for fault detection and identification on regular diagnostic graphs.

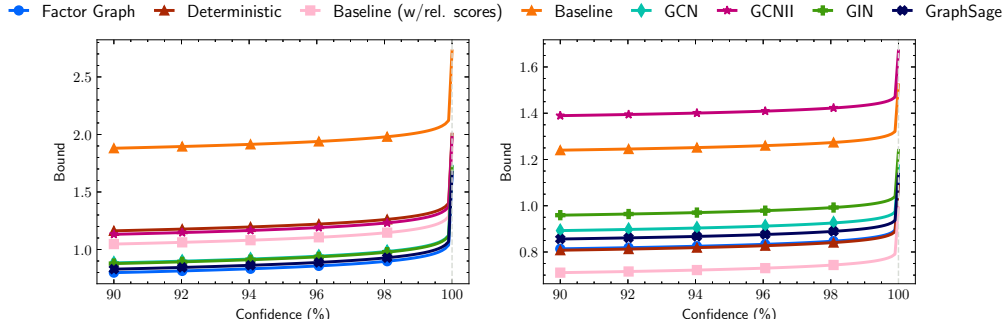


Figure 11: PAC-diagnosability bounds for regular diagnostic graphs. (Left) Modules, (Right) Outputs. Lower is better.

The scenario is novel to the identification algorithm, *i.e.*, not used for training, test, or validation. The results of the failure identification are shown in Fig. 13, where we used the probabilistic fault identification. Initially, the monitor detects no failure (rightmost green section). As the ego vehicle gets closer to the undetected obstacle, the radar detects the obstacle but the camera does not. The inconsistency between the two sets of obstacles causes the test between camera and radar to return FAIL. Given the test’s outcomes, the factor graph correctly detects and identifies the failure, triggering an alarm (rightmost red section). As the ego vehicle gets even closer, the deer goes out of the field-of-view of the radar while entering the LiDAR field-of-view. For a few meters, both camera and LiDAR fail to detect the deer Fig. 14, but since it is out of the field-of-view of the radar, the diagnostic test fails to report the failure⁷. As the obstacle re-enters the field-of-view of the radar, the diagnostic test again returns FAIL, signaling the presence of a failure.

The first alarm is raised 7.19s before the collision, flagging the camera misdetection as an active failure mode. Before the collision, the AV has a speed of 8.43 m/s. The car can reach a maximum deceleration of 6 m/s^2 . As result, the car would need 1.4s to come to a complete stop. We note that after detecting the fault, for a short interval of time the monitor detects no failure: this is due to

⁷This could be solved by improving the logic of the diagnostic test; for instance, it could predict that —while the obstacle moved outside the field-of-view— it is unlikely it disappeared.

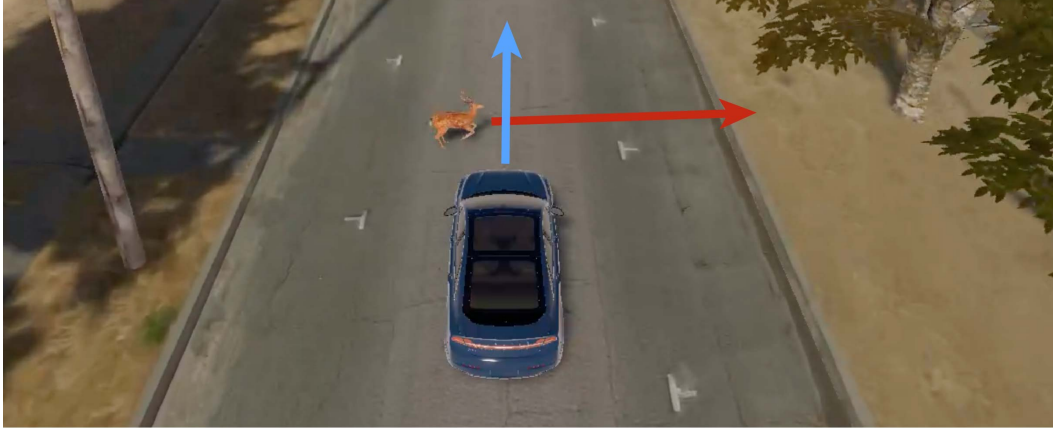


Figure 12: Example scenario involving a deer crossing the road in front of the ego vehicle.

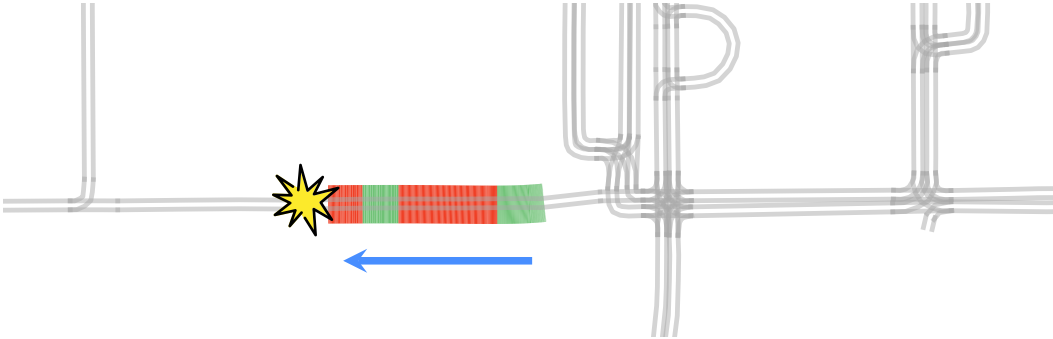


Figure 13: Fault identification results for the example scenario in Fig. 12. The car travels from right to left. Initially, the monitor detects no failure (rightmost, green section). As the ego vehicle gets closer to the obstacle, the LiDAR-based and camera-based obstacle detectors fail to detect the deer while the radar-based obstacle detector correctly locates the obstacle; as a result the fault identification/detection triggers an alarm (red sections).

the fact that the deer goes out of the radar field-of-view, and no other obstacle detector is capable of detecting it, thus lacking redundancy to diagnose the failure; see the visualization and explanation in Fig. 15.

To gather statistical evidence of the effectiveness of the fault detection, we run the same scenario 10 times at different times of the day (sun, twilight, and night) and different weather conditions (including fog and rain). The probabilistic fault detection approach never raised false alarms in these tests, and the average time between the alarm and the collision was 7.54 s. The car traveled at an average speed of 6.16 m/s, requiring 1.03 s to come to a complete stop. The fault identification exhibited an average accuracy of 93.75 %.



Figure 14: Camera Image for the scenario in Fig. 12. Blue bounding box is the ground truth detection. The camera fails to detect the deer crossing the road (misdetection failure).

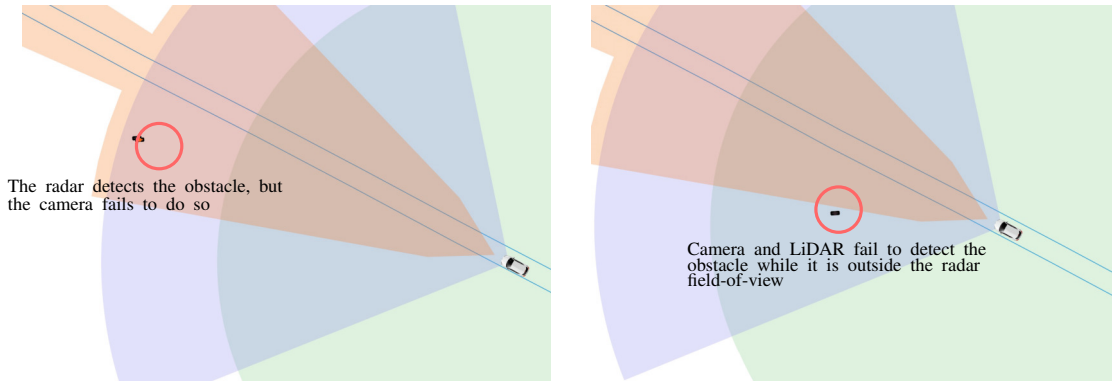

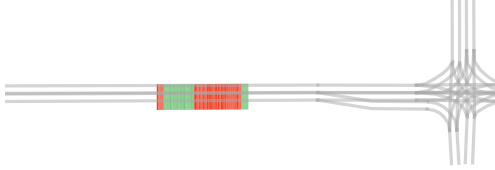

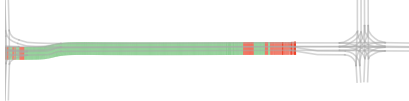

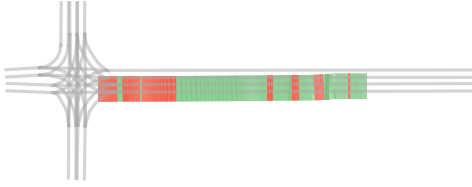




Figure 15: Two snapshots from the example scenario of Fig. 12. Shaded areas represent the sensor field-of-view (FOV): green, blue, and orange represent the LiDAR, camera, and radar FOVs, respectively. On the left, the deer is outside the LiDAR FOV (so the LiDAR obstacle detector is not supposed to detect the obstacle); the radar detects the obstacle, while the camera fails to detect it even if it is inside its FOV. Since the corresponding diagnostic test fails, our monitors can detect the failure. On the right, the deer is outside the radar FOV; in this case, both the camera and the LiDAR fail to detect the obstacles (even though it is within their FOVs), hence no diagnostic test fails and our monitor fails to detect the fault.

F Scenarios

We designed a set of challenging scenarios to stress-test the Apollo Auto perception system. These scenarios were created using the LGSVL Simulator Visual Scenario Editor, which allows the user to create scenarios using a drag-and-drop interface. The vehicle behavior is tested on each scenario in a multitude of situations including different time of day (noon, 6 PM, 9 PM) or weather condition (rain and fog). The scenarios are described in Table 4.

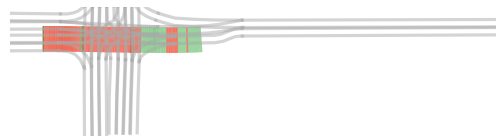
Table 4: Scenarios. ■ Fault-free. ■ Fault-detected.

Scene	Fault Detection Results
 <p>Hidden Pedestrian. A pedestrian, initially occluded by a truck parked on the right-hand side of the street, steps in front of the ego vehicle.</p>	
 <p>Overturned Truck. The ego vehicle encounters an overturned truck occupying the lane it is driving in. The scenario recreates an accident occurred in Taiwan where a Tesla hit an overturned truck on a highway [97].</p>	
 <p>Stopped Vehicle. While driving, the car in front of the ego vehicle makes a lane change to avoid the stationary car that is in their lane. This leaves the ego vehicle with little to no time to react to the stationary car.</p>	
	

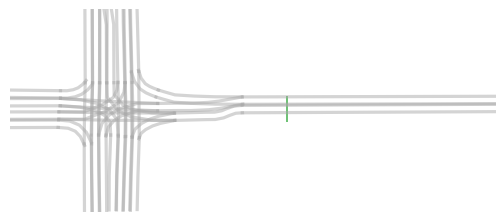
Cut Off Left. While driving in the right lane on a three-lane road, a vehicle from the left lane cuts the ego vehicle off.



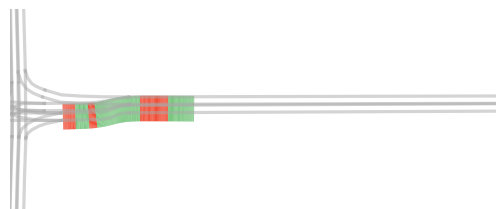
Cut Off Right. While driving in the left lane on a two-lane road, a vehicle from the right lane cuts the ego vehicle off while turning into a parking lot.



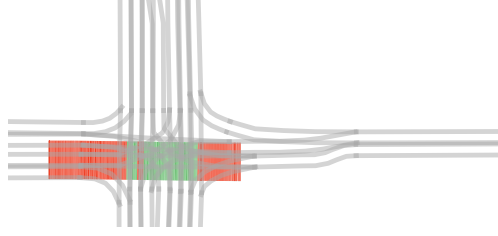
School Bus Intersection. The ego vehicle drives through an intersection. A school bus crosses the intersection coming from the left-hand side. As the ego vehicle crosses the intersection, a pedestrian steps into the intersection from the left-hand side.



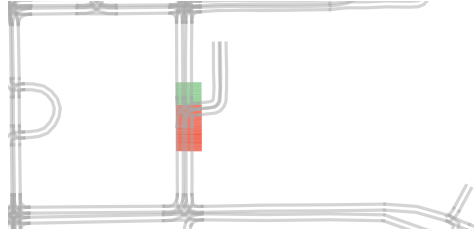
Car in Front. A car is still in front of the ego vehicle preventing it to move forward.



Cones in the Lane. The ego vehicle is driving on a lane partially delimited by traffic cones, while another vehicle is driving in the opposite lane. After passing traffic cones, another vehicle exits a parking lot and merges right in front of the ego vehicle.



Cyclist. The ego vehicle is stopped at an intersection and as it starts driving through the intersection, a cyclist enters the field of view from the left-hand side of the intersection and rides right in front of the ego vehicle.



Turkeys. While driving on a straight road, the ego vehicle must avoid a collision with two turkeys that suddenly walk in front of the ego vehicle.

F.1 Dataset generation

We executed the diagnostic tests described in Appendix D.2 every 0.3 seconds, and used the corresponding test outcomes to perform fault identification. Time synchronization of the modules' output is achieved by pairing outputs that are closest in time to each other. Ground-truth labels for the outputs' failure modes are generated using the ground-truth detections provided by the simulator. In particular, to generate the label for each failure mode of an output, we used the three diagnostic tests described in Appendix D.2.1 comparing the set of obstacles to the ground-truth detections. For a module m instead, since all modules have only one failure mode, the associated failure mode f_m is labeled as ACTIVE if and only if any failure mode if its output is ACTIVE. We collected 1650 regular diagnostic graphs, and split them into 1320 (80%) training samples, 165 (10%) testing samples, and 165 validation samples. We also collected 1590 temporal diagnostic graphs, and split them into 1272 (80%) training samples, 159 (10%) testing samples, and 159 validation samples.