Characterizing Perception Module Performance and Robustness in Production-Scale Autonomous Driving System

Alessandro Toschi[1], Mustafa Sanic[1], Jingwen Leng[1] ⋆, Quan Chen[1], Chunlin Wang[2], and Minyi Guo[1]

[1] Department of Computer Science and Engineering, Shanghai Jiao Tong University
[2] Chuxiong Normal University
{aleto_95, leng-jw, chen-quan, guo-my}@sjtu.edu.cn, mtsanic@gmail.com, wcl@cxtc.edu.cn

Abstract. Autonomous driving is a field that gathers many interests in academics and industry and represents one of the most important challenges of next years. Although individual algorithms of autonomous driving have been studied and well understood, there is still a lack of study for those tasks in a production-scale system. In this work, we profile and analyze the perception module of the open-source autonomous driving system Apollo, developed by Baidu, in terms of response time and robustness against sensor errors. The perception module is fundamental to the proper functioning and safety of autonomous driving, which relies on several sensors, such as LIDARs and cameras, for detecting obstacles and perceiving the surrounding environment. We identify the computation characteristics and potential bottlenecks in the perception module. Furthermore, we design multiple noise models for the camera frames and LIDAR cloud points to test the robustness of the whole module in terms of accuracy drop against a noise-free baseline. Our insights are useful for future performance and robustness optimization of autonomous driving system.

Keywords: Autonomous driving · Robustness analysis · Performance profiling · Deep neural networks

1 Introduction

Autonomous driving is becoming one of the most important applications. We must understand the key characteristics of autonomous driving to build proper architectures and systems for it. There are prior efforts in that direction, but they mainly focus on individual tasks/kernels and rely on the usage of an autonomous driving simulator, such as CARLA [1] or OpenPilot [2]. In other words, detailed knowledge about a production-scale autonomous driving software system is still a missing piece to the puzzle of understanding autonomous driving.

Motivated by that challenge, this work studies Apollo (version 3.5) [3], which is an open-sourced production-scale autonomous driving software, developed by

⋆ Jingwen Leng is the corresponding author of this paper.

Baidu. It has many complex and realistic modules, each of which targets a high-level feature of autonomous driving, such as perception, prediction, planning, as shown in Fig. 1. Modules are described by their input/output (I/O) relationship to other modules, modelled as stages of a pipeline, and not to be intended as monolithic pieces of software, so they can be further decomposed as a set of inner components, following the same architecture and design philosophy. The communication, among modules and components, is data-driven and is enabled by a runtime framework, named Cyber [4], that implements the publisher & subscriber architecture. Each component can write and read on multiple channels, and the messages are serialized using Google Protocol Buffer[5].

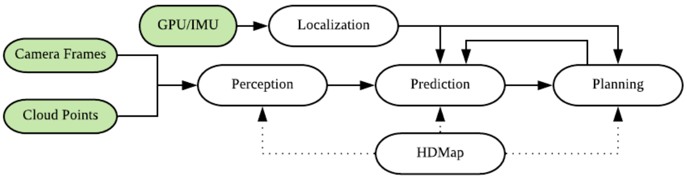


Fig. 1: Apollo Software Architecture

Among the many modules in Apollo, the perception module, which is built to perceive the environment, is the entry point to all the following modules. The module uses multiple sensors, including Full-HD cameras and LIDARs, and is also very computation-intensive as it relies on multiple deep neural networks (DNNs). The whole system depends on the accuracy of such algorithms and detectors to ensure responsiveness and safety. Thus, the perception module needs to be trustable. This paper focuses on the perception module through performance and robustness analysis.

We study the response time of different components in the perception module because it is critical for the predictability and accuracy of the entire system. The response time of each module has been set to 100 ms, which has been adopted as the standard maximum response time since it should ensure a proper and safe reaction to any possible situation. Besides that, response time is also crucial for many dynamic processing routines that use time-deltas to perform online corrections and discard past data. Exceeding these time requirements can lead to a potential loss of useful information that may affect the accuracy of the system. Different from prior efforts on individual sensors [6], our studies focus on the separate and concurrent processing in the presence of multiple sensors, which Apollo uses for safe and reliable output.

The robustness of perception module is also crucial for the autonomous driving system as failures in the module would cause disastrous consequences [7]. Meanwhile, DNN models are model-driven, and therefore, not all the possible scenarios are predictable in the training phase of machine learning algorithms, especially [8]. As such, we extend the robustness analysis methodology from recent efforts [9, 10] to create noise models for both the camera and LIDAR sensors. We use them to test the robustness of DNN models deployed in the perception module.

Our performance analysis complies with prior work on that DNN models take the majority of the average response time, and therefore, they are great candidates for architectural specialization. However, prior work fails to recognize the importance of the CPU owing to the pre- and post-processing that only exists in a production-scale system. The robustness analysis tested the accuracy of the perception against camera and LIDAR noise, separately. The loss of accuracy has been evaluated on obstacles, lanes and the outcome of the experiments highlighted that even if detectors deteriorates, the whole module mitigates the noise thanks to the presence of the fusion component, which, combining the data coming from multiple sensors, can alleviate the effects.

The paper is organized as follows: in Section 2, a comprehensive view of algorithms within the perception module are presented. Section 3 discusses each component according to its response time. In Section 4, noise models and robustness experiments are introduced moreover, later in Section 5, conclusions are provided.

2 Perception Module Description

The perception module, as presented in Fig. 2, is composed of several components. The fusion camera detection is in charge of detecting obstacles, lanes and tracking them through past frames. The LIDAR segmentation identifies obstacles from cloud points, which are further classified and tracked by LIDAR recognition. The output of the LIDAR recognition and the fusion camera detection is now homogenous to be fused into a single coherent detection, taking into account all the obstacles, by the fusion component, which represents the last component of the module.

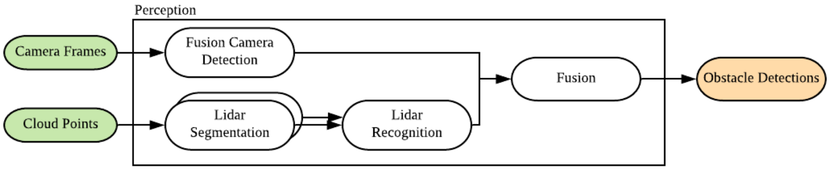


Fig. 2: Perception Software Architecture

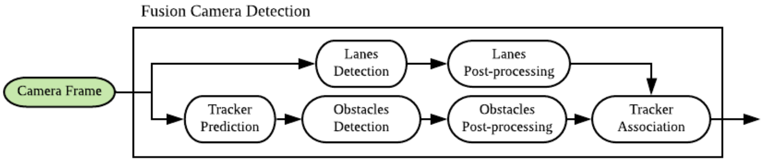


Fig. 3: Obstacle Camera Pipeline

2.1 Camera Sensor and Computation

The objective of the fusion camera detection is to detect obstacles within camera frames, such as vehicles, pedestrians, lanes, and keep the reference of previous obstacles to track them over frames. Detection is achieved by applying camera frames to an obstacle camera pipeline, that is shown in Fig. 3.

Lanes detection is performed using the state-of-art convolutional neural network (CNN) Spatial CNN [11], which outperformed other lanes detector. The main feature of the network is that exploits spatial relationship among pixels, being able to identify straight shaped objects, such as lanes, even if obstructed. Lanes post-processing is necessary to extract, from the raw output of the neural network, the coefficients of the polynomial that approximate each lane together with lane edge points. The points and coefficients belong to the image plane, so another post-processing operation is to project them first upon the car and ground planes and then to combine the two projections, obtaining the three-dimensional representation, which refers to world coordinates.

Obstacles detection is achieved using a CNN based on YOLO [12, 13] that has been enhanced to identify the obstacles bounding boxes in three dimensions. Similar to lanes detection, each bounding box is further projected onto world coordinates by the obstacle post-processor.

Obstacles tracking consists of two main steps: prediction, in which previous obstacles positions and bounding boxes are updated according to time-delta, between the current iteration and the past one, and the car pose through an Adaptive Kalman Filter; association, in which the tracker tries to associate past obstacles to the new obstacles found by the detector, using the frame similarities as a metric, and eventually discard old obstacles.

2.2 LIDAR Sensor and Computation

The LIDAR is a sensor able to measure distances, using light impulses, generating a three-dimensional representation of the neighboring environment. A LIDAR message is a cluster of cloud points; usually, the size of the cluster is around one hundred thousand points. A cloud point represents a three-dimensional point, expressed in LIDAR coordinates, and a light intensity value, which corresponds to the object reflectance [14].

The computation regarding the LIDAR sensor is divided between two components: LIDAR segmentation and LIDAR recognition, as shown in Fig. 2.

LIDAR Segmentation The LIDAR segmentation [15] is the process that detects obstacles, within the surrounding environment, given the cloud points coming from the LIDAR. The pre-processing filters illegal values and discards points that are outside the Region of Interest (RoI) respect the car position. The segmentation is performed using a CNN, so the filtered cloud points need to be converted into feature maps manageable by the neural network. The conversion into feature maps projects the (x, y) coordinates of cloud points over a quantized two-dimensional grid. The CNN is composed of a custom-design convolutional auto-encoder, which selects only the most semantic information for the segmentation, and then a classifier detects the obstacles attributes such as center position, height and class probability. Finally, a bounding box is built for each obstacle, by finding a 6-edge polygon of cloud points, which completely wraps the obstacle.

LIDAR Recognition The LIDAR recognition [15] is in charge of tracking LIDAR obstacles over time. Similar to, the tracking of camera obstacles, each LIDAR obstacle tries to match to an existing obstacle by constructing a bipartite graph, in which to each obstacle is associated with its distance from existing tracks, and then the assignment problem is solved using the Hungarian algorithm [16]. Later, the class assigned to each obstacle is further assigned, taking into consideration past matched obstacles to reducing the class switch during the whole observation.

2.3 Fusion

The fusion component associates and merges obstacles' bounding boxes, coming from LIDARs and cameras and then updates the motion state of each obstacle. The bounding box [17], at time step k, is represented as a vector $\boldsymbol{x}(k) = [x(k),\, y(k),\, \theta(k),\, v(k),\, \omega(k),\, a(k),\, w(k),\, l(k),\, h(k)]^T$, where $(x,\, y)$ is the center position, θ is the heading angle, v is the linear velocity, ω is the angular velocity, a is the acceleration and w, l, h define the width, length and height of the 3D box. The association among bounding boxes is achieved by minimizing the Euclidean distance of the center positions, using the Hungarian algorithm [16]. The motion state is estimated through an Adaptive Kalman Filter with constant acceleration model using the velocity and position provided by bounding boxes [15]. The fusion algorithm implemented is not sequential; observations from sensors are not treated equally since the component defines a main fusion sensor, which triggers the fusion action. The main fusion sensor is configurable and observations dispatched from it cause the execution, determining the fusion frequency. Measurements from other sensors are cached in an ordered queue, according to timestamps. When a new main fusion sensor's message arrives, the component assesses the reliability of cached measurements by checking the time-deltas, between the new message and their timestamps, discarding those who are above a threshold.

3 Performace Analysis

The objective of the performance analysis is to characterize the perception module, using the response time as a metric, to understand the computational effort required by each task. The study is a useful guide to figure out how a real autonomous driving system is designed and its response to real-world scenarios.

Table 1: Neural Network Parameters - A summary of the neural networks present in the perception module.

Network	Input Size	Layers	Parameters
SCNN [11]	640x480x3	143 convolutional 3 deconvolutional	13.68 M
YOLO [12, 13]	1440x800x3	34 convolutional	6.54 M
Segmentation Section 2.2 Custom Design	864x864x4	15 convolutional 10 deconvolutional	2.97 M

Experimental Setup In our simulation, we configure the perception module to use one camera sensor and one LIDAR sensor. The hardware platform we study includes an Intel i7 8700 CPU and an NVIDIA GTX 1080TI GPU, which aligns with Apollo's officially recommended system Nuvo-6108GC [18]. The recommended system uses an Intel i7 6700 CPU and an NVIDIA GTX 1080 GPU. We compile the code with optimization enabled and GPU support. For the input data to the perception module, we use various scenarios taken from the Kitti dataset [19]. The dataset includes both camera frames and cloud points.

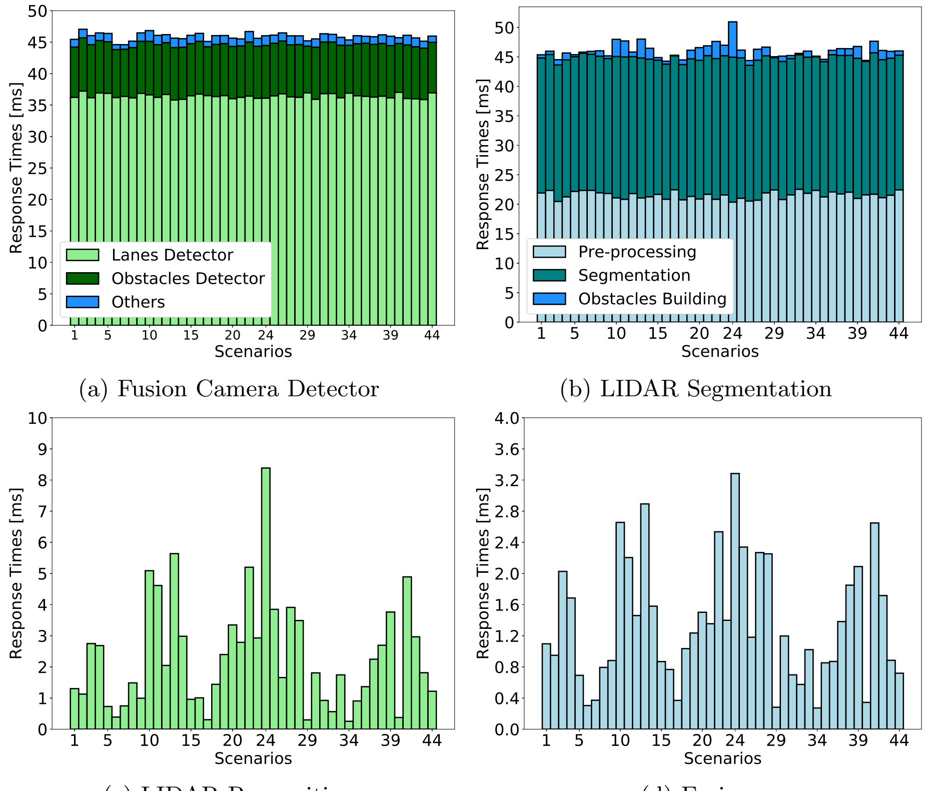
Fig. 4 and Fig. 5 show our performance analysis result. Fig. 4 (a) shows the execution time breakdown of the fusion camera detector component, where the DNN computation (for lane detector and obstacle detector) accounts for the 97.26% of the whole component response time. In contrast, Fig. 4 (b) shows the execution time breakdown of the LIDAR segmentation component, where the DNN computation (for segmentation) only accounts for the 50.55% of the whole component response time. The pre-processing of clout points (one-by-one filtering of cloud points) on the CPU takes about the other half.

Fig. 4 (c) and (d) show the execution time for the LIDAR recognition and fusion component, respectively. Those components run on the CPU and their execution time is much less than the previous two components. However, unlike the constant processing time of the previous two components, the execution time of LIDAR recognition and fusion component highly depend on the number of objects in the frame, which may make their real time processing more challenging. Fig. 5 shows the averaged response time with standard deviation for all computation tasks in the perception module.

In summary, we make the following observations from the results:

O1 The GPU is an crucial architecture for the autonomous driving system since it enables the acceleration of DNNs. It is impossible to use only the CPU to meet the 100 ms because of the heavy computation requirement of DNNs.

O2 The CPU tasks (pre-processing in Fig. 4 (b), LIDAR recognition in (c), and fusion in (d)) still take a large portion of the response time. Specific platform solutions, like NVIDIA Jetson, are built to fit the requirements of autonomous driving using ARM CPUs, which do not provide the same power as the high-end consumer CPU, i.e. the Intel i7 8700 [20]. As such, the CPU may become the bottleneck for the system.



(a) Fusion Camera Detector

(b) LIDAR Segmentation

(c) LIDAR Recognition

(d) Fusion

Fig. 4: These plots display the average response time, expressed in milliseconds, over the scenarios, highlighting the significant tasks where necessary.

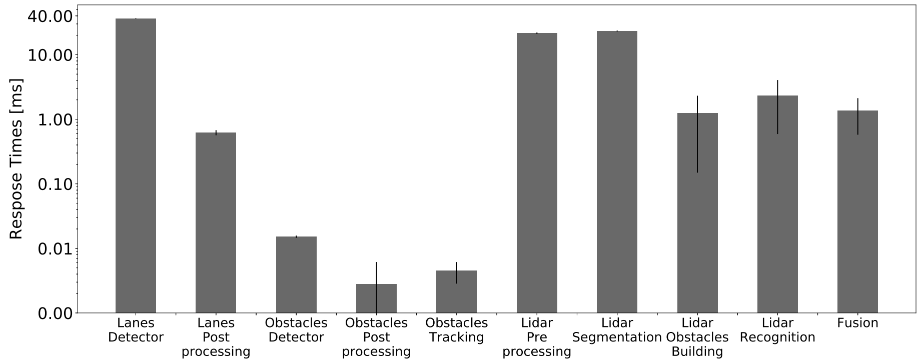


Fig. 5: Averaged response time (logarithmic scale) with standard deviation for different computation tasks in the perception module.

O3 The DNN computation makes the response time highly predictable and can be assumed as constant due to the lower variance exposed by such tasks.

O4 The LIDAR pre-processing can still approximate as a constant task since the number of cloud points varies very little over different frames. But the LIDAR recognition and fusion component highly depend on the number of obstacles, either current and past. As such, it may be desirable to provide computation sprinting mechanism based on the number of obstacles to smooth those modules' response time.

The analysis opens the possibility to explore the relationship among GPU and multiple sensors to identify the proper scheduling policy to adopt in case of race, or the thermal and power impact related to the number of sensors. The study pointed out that neural network computation is statically executed for each frame, without exploiting similarities among them to approximate the outcomes and lighten the workload. Another possibility to diminish the calculation is to share the first convolutional layers among similar neural networks, operating on homogenous input, and then differentiate them according to their functions.

4 Robustness Analysis

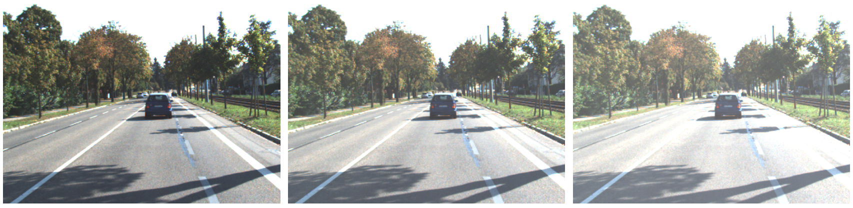
Prior work [21, 22] studies the DNN robustness by inject noise into the GPU architectural states and has shown DNN models are inherently resilient to the architectural transient errors. Unlike those work, we directly inject noise to the different sensors to study the DNN robustness. We set up three experiments to test the accuracy of the obstacles detection, LIDAR segmentation and lanes detection, against camera and LIDAR noise.

We conduct two experiments regard the camera noise, and one experiment regards the LIDAR noise, the trials, for obstacle detection and LIDAR segmentation, using scenarios from the Kitti dataset [19], while the lanes experiment use the TuSimple dataset [23]. The accuracy is evaluated using the F1 metric, which takes into account the presence of false positives and false negatives. A true positive is a matching having an Intersection over Union (IoU) value, with the respects to the noise-free detection, higher than 0.5 for obstacles and 0.3 for the lanes experiment.

4.1 Noise Models

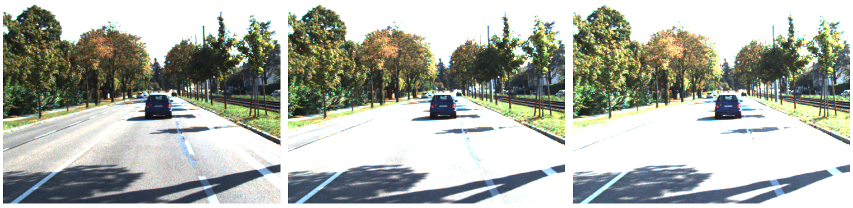
Our noise models between camera and LIDAR sensors are independent. In the real world, external factors like weather can impact the camera and LIDRA sensors simultaneously. Although camera noise models that simulate weather conditions [24] exist, LIDAR still lacks rigorous noise model definitions. Our work can be easily extended to use those joint models in the future.

The camera noise has been modelled using two classical image filters: brightness and contrast [25]. The filters can be expressed using the following mathematical function: $f(\boldsymbol{X}) = \alpha\boldsymbol{X} + \beta$, where α and β represent the contrast and the brightness factor, respectively.



$\beta = 30$ $\beta = 60$ $\beta = 90$

Fig. 6: Brightness



$\alpha = 1.5$ $\alpha = 2.0$ $\alpha = 2.5$

Fig. 7: Contrast

Table 2: Camera Noise Model Parameters

Filter	Values					
Brightness	30.0	60.0	90.0	0.0	0.0	0.0
Contrast	1.0	1.0	1.0	1.5	2.0	2.5

The parameters, used in the camera noise experiment, are listed in Table 2, and an application example of such filters is shown in Fig. 6 and 7.

The LIDAR noise model is composed of two filters: drop filter and Gaussian noise. The drop filter simulates a cloud point loss by randomly discarding a fixed percentage of cloud points, 25% and 50% in this study. The Gaussian noise is applied to cloud points coordinates to simulate a measurement error, having zero mean and a standard deviation of 0.1, which represents a deviation of 10 cm.

$$\boldsymbol{p} = [x, y, z] \qquad \boldsymbol{p}' = \boldsymbol{p} + \boldsymbol{\varepsilon} \qquad \boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, 0.1^2 \boldsymbol{I}_3) \tag{1}$$

4.2 Experiments

The first experiment involves obstacles detection; the camera noise is injected to camera frames to evaluate the accuracy of camera detection and fusion component. The main fusion sensor is set to be the camera, so, LIDAR cloud points are provided without alteration. Similar to the previous, the second experiment modifies cloud points according to LIDAR noise model. The accuracy evaluation accounts LIDAR segmentation together with the fusion component, having set LIDAR as main fusion sensor. The last experiment focuses on lane detection and concerns only the camera detection evaluation since the fusion component is not implicated in this task.

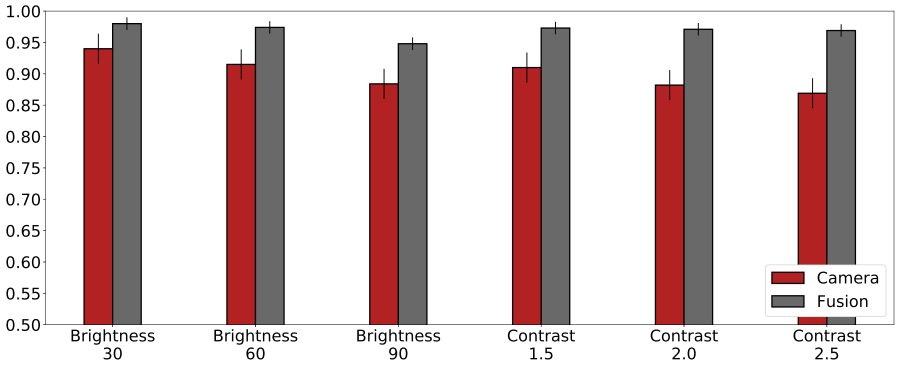


Fig. 8: Accuracies of camera and fusion components based on camera filters.

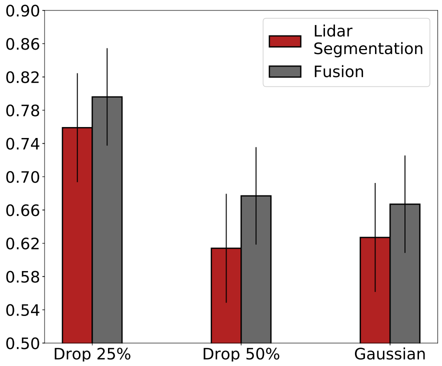


Fig. 9: Accuracies of LIDAR and fusion components based on LIDAR filters.

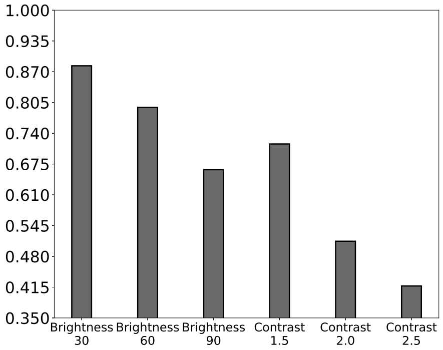


Fig. 10: Accuracies of lane detection based on camera filters.

The experiments show similar behavior; however, there are several points they differ.

O6 The number of obstacles detected by LIDAR is higher than those of the camera, due to the narrower perspective and range of the camera view. The different perspective explains why LIDAR noise compromises more the fusion accuracy than the camera noise, proving that Apollo perception heavily relies on LIDAR.

O7 Fig. 8 and Fig. 9 suggest that obstacles fusion makes the overall component more noise resistant. Moreover, this feature causes further modules to experience less noise by limiting propagation.

O8 Fig. 6 and Fig. 7 show that as brightness and contrast values increase, it gets harder to detect lanes, especially under the sunlight, due to the tendency of the image to become whiter. Fig. 10 presents that contrast filter has a more significant impact on the accuracy of lane detection. The accuracy drops below 45% when the contrast value reaches 2.5.

The results show that the system is susceptible to vision disruptions. Further exploration should involve more complex simulation environments, populated by multiple sensors and validated using rigorous fault injection tools, like the recently proposed model [26], capable of generating faults covering different granularity, such as GPU bit flipping, obstacle obscuration, output alteration within ranges. Efforts should be spent to develop an enhanced version of the fusion component that dynamically changes the main fusion sensor according to the encountered scenario. Finally, it should be advantageous to study the effects of weather conditions on cloud points, to provide reproducible noise models to help the training phase; besides, data augmentation techniques should be adopted to counteract the effects of noise.

5 Conclusion

This work presents the performance and robustness analysis of the perception module belonging to a production-ready autonomous driving system. The performance analysis, focuses on the average response time, composed by neural networks and multiple sensors, which require acceleration via GPU to meet temporal constraints. This analysis demonstrated that neural networks take most of the calculation time and are fixed system, which cannot be dynamically adapted over similar inputs to reduce the inference time. The robustness analysis determines how accurate the camera and LIDAR are when they encounter challenging situations. The models proposed in this paper are similar to real events, such as brightness, contrast and measurement error. The analysis illustrated that the camera component is majorly affected by contrast, which causes detection of lanes and obstacles to be compromised. The robustness analysis also highlighted a new role for the fusion component within the module, which reduces the noise propagation to the following modules.

Acknowledgement This work is supported by National Key R&D Program of China (2018YFB1305900); the National Natural Science Foundation of China under Grant (61702328 and 61602301); Microsoft Research Asia Research Grant.

References

1. A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, “Carla: An open urban driving simulator,” *arXiv preprint arXiv:1711.03938*, 2017.
2. CommaAI, “Openpilot: Open source driving agent,” 2019. `https://github.com/commaai/openPilot`.
3. Baidu, “Apollo open platform,” 2019. `https://github.com/ApolloAuto/apollo`.
4. Baidu, “Apollo cyber,” 2019. `https://github.com/ApolloAuto/apollo/tree/master/docs/cyber`.
5. Google, “Protocol buffers,” 2008. `https://developers.google.com/protocol-buffers/`.
6. S.-C. Lin, Y. Zhang, C.-H. Hsu, M. Skach, M. E. Haque, L. Tang, and J. Mars, “The architectural implications of autonomous driving: Constraints and acceleration,” in *ACM SIGPLAN Notices*, vol. 53, pp. 751–766, ACM, 2018.
7. W. Jack Stewart, “People keep confusing their teslas for self-driving cars,” 2018. `https://www.wired.com/story/tesla-autopilot-crash-dui/`.
8. T. K. S. Lab, “Experimental security research of tesla autopilot,” 2019. `https://keenlab.tencent.com/en/whitepapers/Experimental_Security_Research_of_Tesla_Autopilot.pdf`.

9. D. Hendrycks and T. Dietterich, “Benchmarking neural network robustness to common corruptions and perturbations,” *arXiv preprint arXiv:1903.12261*, 2019.
10. K. Pei, Y. Cao, J. Yang, and S. Jana, “Deepxplore: Automated whitebox testing of deep learning systems,” in *proceedings of the 26th Symposium on Operating Systems Principles*, pp. 1–18, ACM, 2017.
11. X. Pan, J. Shi, P. Luo, X. Wang, and X. Tang, “Spatial as deep: Spatial cnn for traffic scene understanding,” in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
12. J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.
13. J. Redmon and A. Farhadi, “Yolo9000: better, faster, stronger,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7263–7271, 2017.
14. N. O. Service, “What is lidar?,” 2018. `https://oceanservice.noaa.gov/facts/lidar.html`.
15. Baidu, “Apollo 3d obstacles perception,” 2019. `https://github.com/ApolloAuto/apollo/blob/master/docs/specs/3d_obstacle_perception.md`.
16. H. W. Kuhn, “The hungarian method for the assignment problem,” *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.
17. H. Cho, Y.-W. Seo, B. V. Kumar, and R. R. Rajkumar, “A multi-sensor fusion system for moving object detection and tracking in urban driving environments,” in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1836–1843, IEEE, 2014.
18. Neousys, “Nuvo-6108gc series,” 2019. `https://www.neousys-tech.com/en/product/application/rugged-embedded/nuvo-6108gc-gpu-computing`.
19. A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, “The kitti vision benchmark suite,” *URL http://www. cvlibs. net/datasets/kitti*, 2015.
20. P. Michael Larabel, “Nvidia’s jetson agx xavier carmel performance vs. low-power x86 processors,” 2019. `https://www.phoronix.com/scan.php?page=article&item=nvidia-xavier-carmel&num=1`.
21. S. Jha, T. Tsai, S. Hari, M. Sullivan, Z. Kalbarczyk, S. W. Keckler, and R. K. Iyer, “Kayotee: A fault injection-based system to assess the safety and reliability of autonomous vehicles to faults and errors,” in *3rd IEEE International Workshop on Automotive Reliability & Test*, 2018.
22. S. Jha, S. S. Banerjee, J. Cyriac, Z. T. Kalbarczyk, and R. K. Iyer, “Avfi: Fault injection for autonomous vehicles,” in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pp. 55–56, IEEE, 2018.
23. TuSimple, “Tusimple dataset,” 2019. `https://github.com/TuSimple/tusimple-benchmark/issues/3`.
24. K. Garg and S. K. Nayar, “Photorealistic rendering of rain streaks,” in *ACM Transactions on Graphics (TOG)*, vol. 25, pp. 996–1002, ACM, 2006.
25. A. H. M. Rubaiyat, Y. Qin, and H. Alemzadeh, “Experimental resilience assessment of an open-source driving agent,” in *2018 IEEE 23rd Pacific Rim International Symposium on Dependable Computing (PRDC)*, pp. 54–63, IEEE, 2018.
26. S. Jha, T. Tsai, S. Banerjee, S. Kumar Sastry Hari, M. Sullivan, S. Keckler, Z. Kalbarczyk, and R. Iyer, “Ml-based fault injection for autonomous vehicles: A case for bayesian fault injection,” 06 2019.