Characterizing Self-driving Tasks in General-purpose Architectures

Pedro Henrique Exenberger Becker*,1,
Jose Maria Arnau*,1,
Antonio Gonzalez*,1

* *Department of Computer Architecture, Universitat Politècnica de Catalunya*

ABSTRACT

Autonomous Vehicles (AVs) have the potential to radically change the automotive industry. However, computing solutions for AVs have to meet severe performance constraints to guarantee a safe driving experience. Current solutions either exhibit high cost or fail to meet the stringent latency constraints. Therefore, the popularization of AVs requires a low-cost yet effective computing system. Understanding the sources of latency is key in order to improve autonomous driving systems. Here, we present a detailed characterization of Autoware, a modern self-driving car system. We analyze the performance of the different components and leverage hardware counters to identify the main bottlenecks.

KEYWORDS: Autonomous Driving; Characterization

1 Introduction

Autonomous Vehicles (AVs) rely on a complex chain of algorithms to perform the perception of the surrounding world (e.g., object detection and object tracking) that is used to perform the driving decisions (e.g., steering, accelerating, or braking). Notwithstanding, the computing system must complete the execution of these algorithms prior to stringent time deadlines, assuring the vehicle keeps a smooth drive and reacts in time to avoid accidents. These requirements pressure the computing platform for high performance.

The first step to succeed in this challenging scenario and assure the improvement of AVs, is to have a broad yet deep understanding of their software stack and its interaction with the underlying hardware. In this work, we present a thorough characterization of a self-driving architecture, detailing open problems in current software and hardware for future research on AVs. The investigation is performed with a modern and fully open-sourced solution, namely Autoware [aut], which is built upon cutting-edge algorithms for AVs.

Some of the novel findings are the following: i) Autoware cannot guarantee real-time perception on a modern computer with a high-end GPU, as its end-to-end latency frequently exceeds time requirements by more than twofold; ii) LiDAR-related components, that are

[1]E-mail: {pedro,jarnau,antonio}@ac.upc.edu

Table 1: Summary of important Autoware nodes.

Node	Description
voxel_grid_filter	Downsample an input point-cloud, reducing the amount of points to simplify further computations.
ndt_matching	Localize the vehicle by matching LiDAR acquired point-cloud with a region of the HD map point-cloud.
euclidean_cluster	Cluster LiDAR acquired points nearby each other, identifying volumes that can be perceived as objects.
YOLO / SSD	DNN-based nodes used to detect and classify objects (e.g., vehicles, pedestrians) from images.
range_vision_fusion	Combine LiDAR and image-based detected objects into the same coordinates.
ray_ground_filter	Separate an input point-cloud in two: points that compose the ground, and points above the ground level.
imm_ukf_pda_tracker	Track objects by assigning them an identification and keeping it coherent among subsequent frames.
naive_motion_prediction	Extrapolate the current trajectory of different objects to predict where they will be in the future.
costmap_generator	Determine drivable areas in the map, i.e., with no objects at the time or predicted to be in the near feature.
op_planner	Global and local path planning based on the current scene and target location.
pure_pursuit	Calculate the necessary motion (linear and angular acceleration and velocity) to follow the desired path.
twist_filter	A low-pass filter applied over motion control to smooth the vehicle driving.

key to drive the car safely, are important contributors to end-to-end latency, showing execution times in the order of tens of ms; iii) Profiling nodes in isolation leads to a significant underestimation of latency and predictability. A better approach to cover corner cases is to do profiling while stimulating the complete software stack.

2 Autonomous Vehicles Architecture Overview

To introduce the main concepts of AVs we leverage the Autoware.ai project [aut, KTI+15] (here referred to as Autoware) as a representative state-of-the-art AV software stack. The inputs of the system are sensors (e.g., frames from camera and LiDAR), which feed the computing platform with the traffic scene updates, and a High-Definition (HD) map, which contains a point-cloud map enriched with annotations (lanes, crossings, etc.) of the region where the car is driving. The inputs are computed by algorithms for perception (e.g., object detection, object tracking, localization), and actuation (e.g., route planning, steering control). Table 1 summarize some important algorithms featured by Autoware.

The software stack is divided into multiple *nodes*, each implementing an algorithm to solve a task (e.g., detect an object) while globally collaborating towards a major goal (e.g., successfully self-drive a vehicle). The *nodes* communicate among each other through a publish-subscribe arrangement: *nodes* publish their outputs into a shared memory space (*topic*) which other *nodes* can subscribe to. When new messages are published, all *subscriber nodes* are notified, being able to read and process data. This simple solution allows *node* collaboration. More detail on the algorithms and how they collaborate can be found at [KTI+15, BAG20].

3 Characterization Analysis

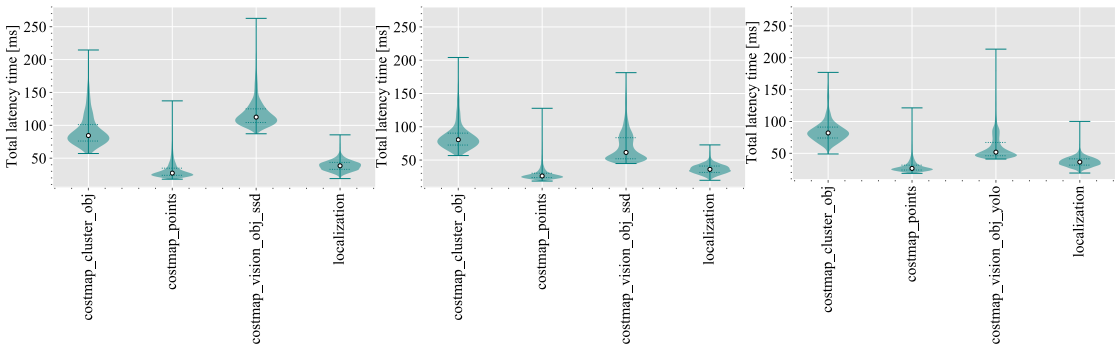
Figure 1 depicts the end-to-end latency distribution of different computation paths (a pipeline of algorithms that transform an input into relevant perception information) performed by Autoware, with data from an eight-minute real-life drive. Results were performed with an

Intel i7-7700K CPU and an NVIDIA GeForce GTX 1080 GPU. The end-to-end latency accounts for the time an input is processed along with a set of consecutive nodes until its final contribution to perception. For example, *costmap_vision_obj_yolo* (in Figure 1c) indicates the latency distribution of performing image-based object detection, fusion with recent-most LiDAR data, object tracking, motion prediction, and generating a costmap of what are safe spaces to drive by. Other measurements in Figure 1c are similar, but with different combinations of algorithms, and using LiDAR data instead of camera images. Figures 1a and 1b change the vision detection node (SSD instead of Yolo-v3). More information can be found at [BAG20].

Some interesting observations can be taken from Figure 1. First, looking at the average values of the distributions (small white circle), we can identify that computation paths which do object detection are more demanding than others (for instance, localization). Thus improving these computations should be prioritized. Also, we see that the tail latency (end limits of the distribution range) frequently crosses the 100 ms barrier, which is commonly accepted as a target time deadline for AVs [LZH+18]. This happens regardless of the vision detection node, even though, on average, Yolo-v3 and SSD300 are faster than SSD500. This means that current high-end systems are not sufficient to execute cutting-edge self-driving algorithms under time deadlines. Here it is also important to highlight that computation paths that handle LiDAR data also surpass the time constraint. This indicates that better hardware support for speeding up LiDAR-based algorithms would be fruitful. Nonetheless, there is still a large research gap in optimizing computer architecture for LiDAR-based algorithms compared to image-based neural networks.

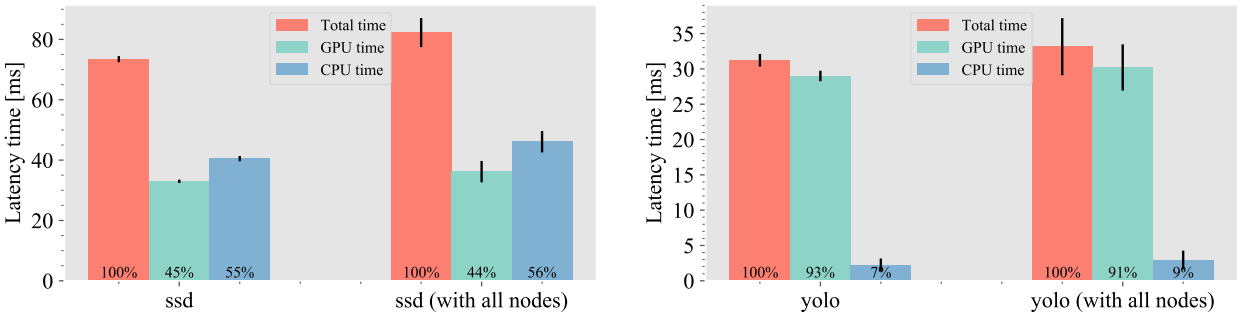
Finally, we present Figure 2. In this case, we study the latency of an individual algorithm: image-based object detection with Deep Neural Networks (DNNs). We experiment with two different DNNs, namely SSD500 and Yolo-v3, accounting for the CPU and GPU share of time for their executions. In each case, we provide inputs (frames) to the node while running *alone* and while running along with *the complete Autoware stack*. With this, we assess the impacts of resource sharing on the performance of a given node. Bars indicate the mean value, and the vertical line on the top indicates the standard deviation of the data.

Two main points arise from these experiments. First, when nodes share the computing platform, their mean latency increase. For instance, SSD500 mean latency increases from 73.45 ms (isolation) to 82.26 ms (all software stack); an increase of 12%. As for the Yolo-v3



(a) Autoware with SSD512 image detector. (b) Autoware with SSD300 image detector. (c) Autoware with Yolo-v3 image detector.

Figure 1: Autoware's end-to-end perception latency considering different image detectors.



(a) SSD512 image detector.

(b) Yolo image detector.

Figure 2: The CPU and GPU time share for SSD512 and Yolo DNNs. Experiments consider nodes running in isolation and together with the complete software stack.

node, the mean latency rises from 31.23 ms (isolation) to 33.14 ms (all software stack); a 6% increase in the mean latency. Thus, using single node measurements to assess latency bottlenecks can be a pitfall, since threats may be under-considered as nodes perform better, on average, when running in isolation. The second finding is that the standard deviation of an individual node latency increases when other nodes are executing. For SSD500 the standard deviation goes from 1.01 ms (isolation) to 4.81 ms (all software stack). Similarly, the standard deviation for Yolo-v3 node latency goes from 0.88 ms (isolation) up to 4.05 ms (all software stack). Thus, when all nodes are executing, the predictability of the latency for each node is weakened. Since AVs should cope with worst-case scenarios, such as the ones that cause tail latency, full system experimentation is best suited for profiling purposes. In this case, the node's variability increases, causing unpredictable behavior that cannot be found when nodes run alone.

References

[aut] Autoware.AI · GitHub. https://github.com/Autoware-AI.

[BAG20] Pedro H E Becker, Jose Maria Arnau, and Antonio Gonzalez. Demystifying Power and Performance Bottlenecks in Autonomous Driving Systems. In *2020 IEEE International Symposium on Workload Characterization (IISWC)*, pages 205–215. IEEE, oct 2020.

[KTI+15] Shinpei Kato, Eijiro Takeuchi, Yoshio Ishiguro, Yoshiki Ninomiya, Kazuya Takeda, and Tsuyoshi Hamada. An open approach to autonomous vehicles. *IEEE Micro*, 35(6):60–68, 2015.

[LZH+18] Shih Chieh Lin, Yunqi Zhang, Chang Hong Hsu, Matt Skach, Md E. Haque, Lingjia Tang, and Jason Mars. The architectural implications of autonomous driving: Constraints and acceleration. *ACM SIGPLAN Notices*, 53(2):751–766, 2018.

This work has been supported by the the CoCoUnit ERC Advanced Grant of the EU's Horizon 2020 program (grant No 833057), the Spanish State Research Agency under grant PID2020-113172RB-I00 (AEI/FEDER, EU), the ICREA Academia program, and the grant 2020 FPI-UPC_033.