

An illustration of a building under construction. The building is a large, multi-story structure with a grid-like facade. Several workers wearing hard hats are visible on the scaffolding, working on the building's exterior. The background is a solid orange color, and the building's facade is composed of yellow and blue rectangular panels. The title 'Computer Architectures for Autonomous Driving' is overlaid on the illustration in a large, red, sans-serif font.

Computer Architectures for Autonomous Driving

Shaoshan Liu, PerceptIn

Jie Tang, South China University of Technology

Zhe Zhang, PerceptIn

Jean-Luc Gaudiot, University of California, Irvine

To enable autonomous driving, a computing stack must simultaneously ensure high performance, consume minimal power, and have low thermal dissipation—all at an acceptable cost. An architecture that matches workload to computing units and implements task time-sharing can meet these requirements.

Interest in autonomous driving has grown to the point that the US Department of Transportation's National Highway Traffic Safety Administration (NHTSA) has formally defined five autonomous-driving levels.¹ At level 0, the driver has complete control, and at levels 1 through 3 the vehicle gradually increases control until at level 4, the vehicle has complete control. At that level, the vehicle must perform all safety-critical functions for the entire trip, from start to stop, including parking, and the driver is not expected to control the vehicle at any time.

Thus, a fully autonomous vehicle must be able to perceive its environment and safely navigate on the basis of multiple sensors rather than human input. Sensors include laser imaging detection and ranging (LiDAR), a global positioning system (GPS), an inertial

measurement unit (IMU), various cameras, or any combination of these. Each vehicle uses sensor input in *localization*—the process of understanding its environment—and in making real-time decisions about how to navigate within that perceived environment. These tasks involve processing a high volume of sensor data and require a complex computational pipeline. For example, in current designs, an autonomous car must have multiple servers, each with several high-end CPUs and GPUs. Consequently, autonomous vehicles consume an extreme amount of computing and electrical power—often thousands of watts. The resulting prohibitive cost is a formidable barrier to autonomous driving's market penetration.

Researchers are actively exploring computer architectures that will reduce the cost of autonomous driving to

the extent that it is affordable for the general population. To better understand these efforts, we examined the tasks involved in current LiDAR-based autonomous driving and investigated how that approach differs from vision-based autonomous driving—a paradigm that is garnering more attention. We also analyzed how various computing platforms and processing solutions exploit computing resources, such as CPUs, GPUs, field-programmable gate arrays (FPGAs), and digital signal processors (DSPs), and attempted to identify the most suitable computing resource for each autonomous-driving task. On the basis of our analysis, we developed a system architecture that can run tasks on a heterogeneous ARM mobile system on chip (SoC). Our architecture, which is modular, secure, energy-efficient, and capable of delivering high computing performance, has already been implemented in PerceptIn products.

AUTONOMOUS-DRIVING TASKS

As Figure 1 shows, to achieve autonomous operation in urban situations, which typically have unpredictable traffic, several real-time systems must interoperate, including sensor processing, perception, localization, planning, and control.² Most successful implementations of autonomous driving rely heavily on LiDAR for mapping, localization, and obstacle avoidance, using other sensors for peripheral functions.^{3,4}

Sensing

An autonomous vehicle consists of several major sensors, each of which has advantages and drawbacks, which requires combining sensors to increase reliability and safety.

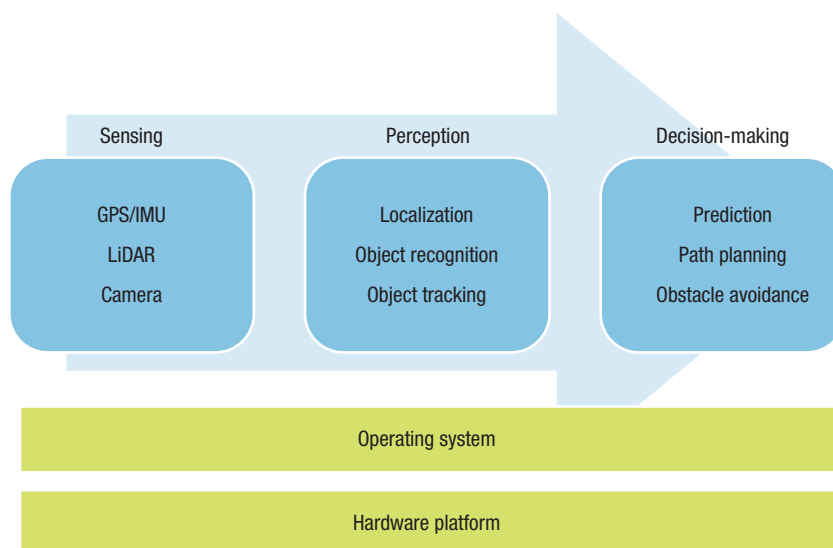


FIGURE 1. Tasks in autonomous driving. Tasks fall roughly into one of three stages: sensing, perception, or decision-making. Driving systems tend to rely heavily on laser imaging detection and ranging (LiDAR) for most sensing, using a global positioning system (GPS) along with an inertial measurement unit (IMU) or a camera for peripheral sensing.

GPS/IMU. The GPS/IMU system reports both a global position estimate and inertial updates at a high rate and is thereby essential in helping the autonomous vehicle localize itself. A GPS provides fairly accurate localization, but its update rate is only 10 Hz, rendering it incapable of providing real-time updates. Conversely, an IMU's accuracy degrades with time, so it cannot provide reliable positioning over long periods. It can, however, furnish more frequent updates at 200 Hz or higher, which satisfies the real-time requirement. For a vehicle traveling 60 mph, the traveled distance is less than 0.2 m between two position updates, which means a worst-case localization error of less than 0.2 m.

Combining GPS and IMU provides both accurate and real-time updates for vehicle localization, but it does not suffice as the only path to localization. There are three drawbacks: accuracy is only about one meter; the GPS signal might bounce off buildings, introducing more noise; and the GPS requires an unobstructed view of the sky, which makes it infeasible for environments such as tunnels.

LiDAR. In LiDAR-based sensing, the sensor bounces a laser beam off surfaces and measures the reflection time to determine the vehicle's distance from various objects. Because LiDAR is highly accurate, an autonomous vehicle can rely on it as the main sensor to produce high-definition maps, localize itself while moving (using the map as a reference), and detect obstacles. Typically, a LiDAR unit, such as a Velodyne 64-beam laser, rotates at 10 Hz and takes about 1.3 million readings per second. Despite these advantages, LiDAR measurements can be extremely noisy, such as when raindrops and dust are in the air, and a 64-beam LiDAR can cost upwards of \$80,000.

Camera. Autonomous vehicles use cameras mostly for object recognition and tracking, for example, to detect lanes, traffic lights, and pedestrians. To enhance safety, existing implementations usually mount eight or more 1,080-pixel cameras around the car such that they can detect, recognize, and track objects in front of, behind, and on both sides of the vehicle. These cameras usually run at 60 Hz and,

when combined, generate around 1.8 Gbytes of raw data per second.

Radar and sonar. The radar and sonar system is used primarily as the last line of defense in obstacle avoidance. The generated data shows the distance to the nearest object in front of the vehicle's path. Once it detects that an object is close and collision is possible, the immediate next step is for the autonomous vehicle to brake or turn. Because of this response urgency, the data generated by radar and sonar is fed directly to the control processor, not through the main computation pipeline. The control processor then implements functions such as swerving, applying the brakes, or prepositioning the seatbelts.

Perception

The sensor data is input to the perception stage, in which the vehicle attempts to understand its environment. The three main perception tasks are localization, object detection, and object tracking.

Localization. Localization requires a fusion of data from sensors such as GPS/IMU and LiDAR systems, which results in a high-resolution infrared reflectance ground map. In some autonomous vehicle implementations, a particle filter is used to correlate the LiDAR measurements and the map.⁵ This method has achieved real-time localization with 10-cm accuracy and has proven effective in urban environments. However, LiDAR's high cost could limit the wide application of a particle filter.

Object detection. In recent years, we have seen the rapid development of vision-based deep-learning technology, which enables highly accurate object

detection and tracking.⁶ The Convolution Neural Network (CNN), a deep neural network (DNN) that is widely used in object-recognition tasks, has a four-layer evaluation process:

- › The *convolution* layer contains a variety of filters to extract features from the input image. Because each filter contains a set of learnable parameters that will be derived after the training stage, the convolution layer can be the most computationally intensive layer in the CNN.
- › The *activation* layer has mechanisms for deciding whether to activate the target neuron.
- › The *pooling* layer reduces the representation's spatial size to reduce the number of its parameters and consequently the required computation.
- › The *fully connected* layer contains neurons with full connections to all activations in the pooling layer.

Object tracking. After object recognition identifies the object that must be tracked, object tracking aims to automatically track the moving object's trajectory. The main goal is to ensure that the vehicle does not collide with a moving object, whether a vehicle or a person crossing the road. In recent studies, deep-learning techniques have demonstrated advantages in object tracking relative to computer vision-based approaches.⁷ For example, by using auxiliary natural images, a stacked auto-encoder can be trained offline to learn generic image features that are more robust against variations in viewpoints and vehicle positions. The offline trained model can then be applied in online tracking.

Decision-making

Given an understanding of the vehicle's environment, decision-making tasks aim to generate a safe and efficient action plan in real time using primarily probabilistic processes and Markov chains. The three main tasks are prediction, path planning, and obstacle avoidance.

Prediction. When navigating traffic, one of a human driver's main challenges is how to anticipate and cope with the actions of other drivers. This problem is exacerbated when the road has multiple lanes or when the vehicle is at an intersection.⁸ To ensure that the vehicle travels safely in these environments, decision-making must be able to predict nearby vehicles' actions. One approach is to generate a stochastic model of the reachable position sets of the other traffic participants, and associate these sets with probability distributions.

Path planning. Planning the path of an autonomous, agile vehicle in a dynamic environment is a thorny problem, particularly when the vehicle must use its full maneuvering ability. A brute-force approach—to search all possible paths and use a cost function to identify the best path—would require enormous computational resources and might be unable to deliver navigation plans in real time. To circumvent the computational complexity of deterministic, complete algorithms, one research group proposed using probabilistic planners to provide effective real-time path planning.⁹

Obstacle avoidance. Because safety is the paramount concern in autonomous driving, at least two levels of

obstacle-avoidance mechanisms must be deployed to ensure that the vehicle will not collide with any object. The first level is proactive and is based on traffic predictions.¹⁰ At runtime, the traffic-prediction mechanism generates measures like time to collision or predicted minimum distance, which the obstacle-avoidance mechanism uses to replan local paths. If the proactive mechanism fails, the second level, a reactive mechanism, takes over. This mechanism relies on radar data to detect an obstacle and uses that data to override the current control and avoid the detected obstacles.

LIDAR VS. VISION-BASED SYSTEMS

LiDAR can produce more than a million data points per second with a range up to 200 m, but a high-end LiDAR sensor can cost tens of thousands of dollars. A promising and affordable alternative is vision-based autonomous driving. One advantage relative to LiDAR is that vision-based localization does not rely on a particle filter³ but rather uses visual odometry—a technique that relies more on feature description.¹¹ A vision-based system first triangulates stereo image pairs to obtain a disparity map, which it then uses to derive depth information for each data point. By matching salient features between successive stereo image frames, a vision-based system establishes correlations between feature points in different frames, allowing it to estimate the motion between the past two frames. Finally, by comparing the salient features against those in the known map, it can derive the vehicle's current position.

As this processing pipeline shows, vision-based and LiDAR systems take different approaches to localization.

The point clouds generated by LiDAR provide the environment's shape description, but it is hard to differentiate individual points. Thus, the LiDAR system must use a particle filter to compare a specific observed shape against

platforms, we evaluated the computation hardware for a prototype level 4 autonomous vehicle. We then examined how chipmakers are attempting to meet these challenges. Our investigation of the vehicle hardware is

**OUR SoC PERFORMED SUCCESSFULLY
PRIMARILY BECAUSE WE WERE ABLE
TO MATCH WORKLOAD TYPE WITH THE
MOST SUITABLE COMPUTING UNIT.**

the known map to reduce uncertainty. In vision-based localization, observations are processed through a full pipeline of image processing to detect features and generate feature descriptions. Consequently, vision-based systems can uniquely identify each data point and apply the salient points to directly compute the vehicle's current position.

Moreover, unlike LiDAR, a vision-based approach introduces several highly parallel data-processing stages, including feature extraction, disparity map generation, optical flow, feature match, and Gaussian blur. These stages use vector computations extensively, and each task usually has a short processing pipeline, which means that these workloads are best suited for DSPs. In contrast, a LiDAR approach heavily uses the Iterative Closest Point (ICP) algorithm,¹² which is an iterative process that is hard to parallelize and thus more efficiently executed on a sequential CPU.

CURRENT PLATFORMS

To understand the main challenges in autonomous-driving computing

based on interactions with the company designing it, which requested to remain anonymous.

The vehicle's current computing platform consists of two computing boxes, each equipped with an Intel Xeon E5 processor and four to eight Nvidia K80 GPU accelerators, connected with a PCI-E bus. At peak performance, the 12-core CPU can deliver 400 giga-operations per second (GOPS), consuming 400 W of power. Each GPU is capable of 8 tera-operations per second (TOPS), while consuming 300 W of power. Overall, the system can deliver 64.5 TOPS at about 3,000 W. One computing box is connected to 12 high-definition cameras around the vehicle, for object detection and object tracking. A LiDAR unit is mounted on top of the vehicle for vehicle localization as well as for some obstacle-avoidance functions. The second computing box is for reliability, performing exactly the same tasks; if the first box fails, the second can immediately take over. Both boxes running at their peak (worst case) would consume more than 5,000 W of power, which would generate an

extreme amount of heat. Also, each box costs approximately \$20,000 to \$30,000, making the whole solution unaffordable for average consumers.

To address these drawbacks, chip-makers have proposed several platform types and processing solutions, including those based on GPUs, DSPs, FPGAs, and ASICs.

GPUs

The Nvidia PX2 platform is the current leading GPU-based solution for autonomous driving. Each PX2 consists of two Tegra SoCs and two Pascal graphics processors. Each GPU has a dedicated memory, as well as specialized instructions for DNN acceleration. To deliver high throughput, each Tegra connects directly to the Pascal GPU using a PCI-E Gen 2 × 4 bus (for a total bandwidth of 4.0 Gbytes/s). In addition, the dual CPU-GPU cluster is connected over Gigabit Ethernet, delivering 70 Gbits/s. With optimized I/O architecture and DNN acceleration, each PX2 can perform 24 trillion deep-learning calculations per second, which translates to 2,800 images per second when it runs AlexNet deep-learning workloads.

DSPs

Texas Instruments' TDA is a DSP-based autonomous-driving solution. A TDA2x SoC consists of two floating-point C66x DSP cores and four fully programmable vision accelerators, which are designed for vision-processing functions. The accelerators allow the TDA to complete vision tasks eight times faster than is possible with an ARM Cortex-15 CPU while consuming less power.

CEVA's CEVA-XM4, another DSP-based solution, is designed to execute computer-vision tasks on videostreams. Its primary advantage is its energy efficiency; the CEVA-XM4 requires less

than 30 mW for a 1,080-pixel video at 30 frames per second (fps).

FPGAs

Altera's Cyclone V SoC is an FPGA-based autonomous-driving solution that is used in Audi products. Altera's FPGAs are optimized for sensor fusion, combining multiple sensors' data to achieve highly reliable object detection. Another solution is Zynq UltraScale MP SoCs. When running CNN tasks, the UltraScale achieves 14 images per second per watt (images/s/W), which outperforms the Tesla K40 GPU (4 images/s/W). Also, for object tracking, it reaches 60 fps in a live 1,080-pixel videostream.

ASICs

MobilEye EyeQ5 is a leading ASIC-based solution for autonomous driving. EyeQ5 features heterogeneous, fully programmable accelerators, each of which is optimized for its own algorithm family, including computer-vision, signal-processing, and machine-learning tasks. With this architectural diversity, applications can use the most suitable core for each task, saving both computational time and energy. To enable system expansion with multiple devices, EyeQ5 implements two PCI-E ports for inter-processor communication (IPC).

FINDING AN OPTIMAL PLATFORM

Each prospective solution has its strengths, but no one platform appears to be optimal. As part of analyzing the current state of computer architecture for autonomous driving, we attempted to answer three questions:

- › Which computing units are best suited for what workloads?

- › Would a mobile processor be sufficient to perform autonomous-driving tasks?
- › How can we design the most efficient computing platform for autonomous driving?

Matching workload to computing unit

To answer the first question, we had to identify which computing units best fit with convolution and feature-extraction workloads, which are the most computationally intensive. We conducted experiments on an off-the-shelf ARM mobile SoC consisting of a four-core CPU, a GPU, and a Qualcomm Snapdragon 820 DSP (qualcomm.com/products/snapdragon/processors/820). To study the performance and energy consumption of this heterogeneous platform, we implemented and optimized convolution and feature-extraction tasks and measured chip-level energy consumption.

Figure 2 shows the results. Figure 2a, the results for convolution tasks, confirms that the GPU is the most efficient computing unit. On the CPU, each convolution takes about 8 ms to complete, consuming 20 mJ; on the DSP, each convolution takes 5 ms to complete, consuming 7.5 mJ; and when running on a GPU, each convolution takes only 2 ms to complete, consuming only 4.5 mJ.

Figure 2b shows the results for feature extraction, which generates feature points for localization and is the most computationally intense task in the localization pipeline. Each feature-extraction task running on a CPU takes about 20 ms to complete and consumes 50 mJ; when running on a GPU, each task takes 10 ms to complete, consuming 22.5 mJ; and when running on a DSP, each task takes only 4 ms to

complete and consumes 6 mJ. Thus, the DSP is the most efficient unit for feature-extraction tasks, both in performance and in energy consumption.

Performance on a mobile processor

To determine how well an autonomous-driving system would perform on the ARM mobile SoC, we implemented the vision-based system shown in Figure 3.

When we ran the navigation system on the mobile SoC, the localization pipeline was able to process 25 images/s, and the deep-learning pipeline performed three object-recognition tasks per second. We designed the planning and control pipeline to plan a path within 6 ms. The entire SoC consumed only 11 W on average, and we were able to drive the vehicle about 5 mph without losing any localization. These results are remarkable considering that the vehicle ran on a mobile SoC. With additional computing resources, the system should be able to process much more data, allowing the vehicle to go faster and eventually satisfying the needs of a production-level autonomous-driving system.

Designing an efficient platform

Our performance on the ARM mobile SoC was possible because we fully used the system's heterogeneous computing resources and applied the most suitable computing unit for each task. However, we could not fit all the tasks into this system and had to exclude object tracking, lane-change prediction, cross-road traffic prediction, and so on. Moreover, the autonomous-driving system should be able to upload raw sensor data and process data to the cloud, which would take all the available network bandwidth.

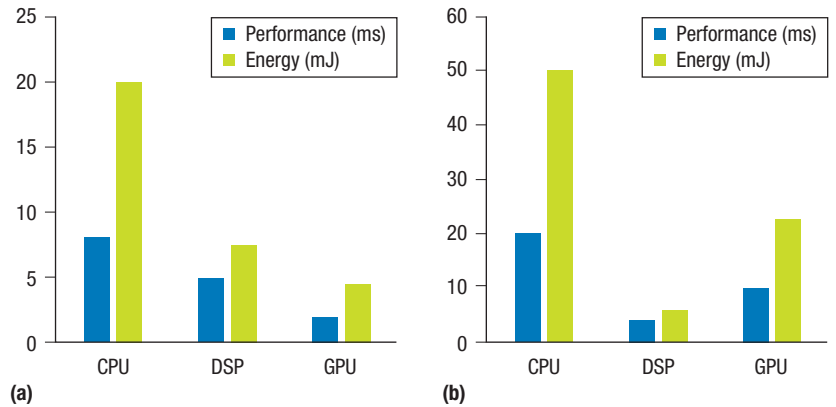


FIGURE 2. Performance and energy in (a) convolution and (b) feature-extraction tasks. In (a), the GPU takes only 2 ms and uses only 4.5 mJ to complete convolution tasks. In (b), the digital signal processor (DSP) is the most efficient unit for feature extraction, taking 4 ms and consuming only 6 mJ to complete a task.

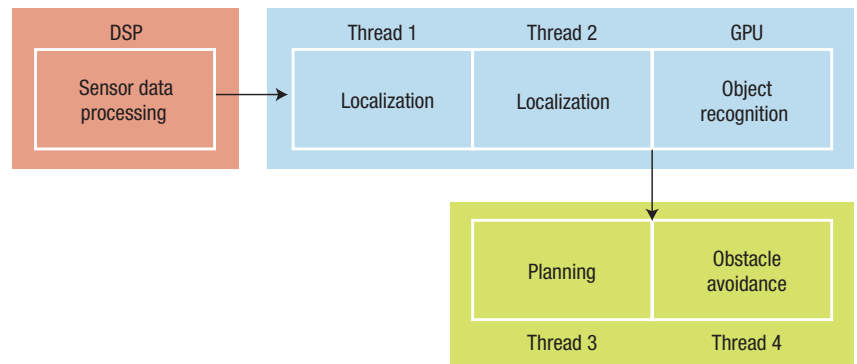


FIGURE 3. Autonomous navigation system implemented on an ARM mobile system on chip (SoC). The DSP handles tasks that involve processing sensor data, such as feature extraction and optical flow. The GPU takes care of deep-learning tasks, such as object recognition. This design has four CPU threads: two CPU threads are dedicated to localization tasks in real time, one CPU thread is for real-time path planning, and one CPU thread is for obstacle avoidance.

One solution is to recognize that functions such as object tracking, lane-change prediction, cross-road traffic prediction, and data uploading are needed sporadically, not all the time. For example, object tracking is triggered by object recognition, and traffic prediction is triggered by object tracking. Data uploading need not be continuous because uploading data in batches usually improves throughput and reduces bandwidth use.

Thus, we reasoned that an ASIC chip for each of these tasks would be a waste of chip area, but an FPGA would be a perfect fit. These tasks could

then time-share the FPGA. Indeed, researchers have demonstrated that partial-reconfiguration techniques¹³ allow an FPGA core to be changed within less than a few milliseconds, enabling sharing in real time.

Figure 4 shows our computing stack for autonomous driving, which is already being shipped in PerceptIn's autonomous-driving product line. In the computing-platform layer is an SoC architecture consisting of an I/O subsystem, a CPU, and shared memory through which the computing and I/O components communicate. The DSP preprocesses the image

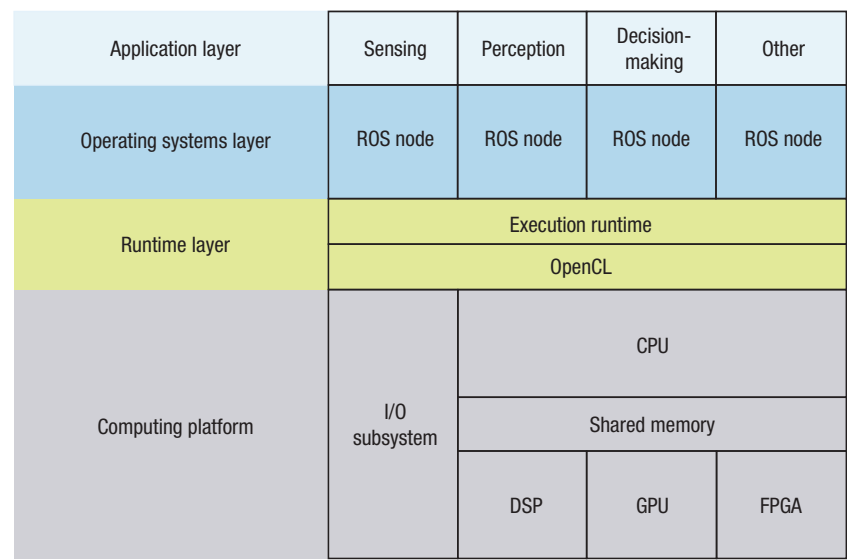


FIGURE 4. Computing stack for autonomous driving. The stack consists of application, operating systems, runtime, and computing platform layers. At the computing-platform level is an SoC architecture consisting of an I/O subsystem that interacts with front-end sensors, a DSP to preprocess the image stream for feature extraction, a GPU to perform deep-learning tasks, and an FPGA for data compression and uploading. Computing components interact through shared memory. ROS: robot operating system.

stream to extract features; the GPU performs object recognition and other deep-learning tasks; the multi-core CPU is for planning, control, and interaction tasks; and the FPGA can be dynamically reconfigured and time-shared for data compression and uploading, object tracking, and traffic prediction.

On top of the computing-platform layer is a runtime layer to map different workloads to the heterogeneous computing units through OpenCL and to schedule different tasks at runtime. Next is the operating systems layer, which uses robot OS (ROS; ros.org) design principles. ROS is a distributed system consisting of multiple ROS nodes, each encapsulating a task in autonomous driving. The last layer consists of the sensing, perception, and decision-making applications.

Our architecture has several key benefits. It is *modular*, in that ROS nodes can be added if more functions are required. It is *secure*, in that each ROS node has a mechanism that prevents other nodes from impacting it. It is *highly dynamic*, in that the runtime

layer can schedule tasks for maximum throughput, lowest latency, or lowest energy consumption. It is capable of *high performance* because each heterogeneous computing unit is used for the most suitable task. Finally, it is *energy-efficient*, in that the most energy-efficient computing unit is used for each task.

Existing computing solutions for level 4 autonomous driving often consume thousands of watts, dissipate enormous amounts of heat, and cost tens of thousands of dollars. These power, heat, and cost barriers make autonomous-driving technologies difficult to transfer to the general public.

PerceptIn has already shipped products with our autonomous-driving computing stack, and we are confident that it can deliver several the benefits we have described. As more work is completed on computing for autonomous driving, we foresee level 4 vehicles becoming affordable and widespread, simplifying daily life for many more people. ■

ACKNOWLEDGMENTS

This work is partly supported by the National Science Foundation (NSF) under grant XPS-1439165. Any opinions, findings, and conclusions or recommendations expressed in this article are those of the authors and do not necessarily reflect the views of the NSF. Jie Tang is the corresponding author for this article.

REFERENCES

1. Nat'l Highway Traffic Safety Admin., *Preliminary Statement of Policy Concerning Automated Vehicles*; nhtsa.gov/staticfiles/rulemaking/pdf/Automated_Vehicles_Policy.pdf.
2. J. Levinson et al., "Towards Fully Autonomous Driving: Systems and Algorithms," *Proc. IEEE Intelligent Vehicles Symp.* (IV 11), 2011, pp. 163–168.
3. J. Levinson and S. Thrun, "Robust Vehicle Localization in Urban Environments using Probabilistic Maps," *Proc. IEEE Int'l Conf. Robotics and Automation* (ICRA 10), 2010, pp. 4372–4378.
4. A. Teichman, J. Levinson, and S. Thrun, "Towards 3D Object Recognition via Classification of Arbitrary Object Tracks," *Proc. IEEE Int'l Conf. Robotics and Automation* (ICRA 11), 2011; cs.stanford.edu/people/teichman/papers/icra2011.pdf.
5. S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*, MIT Press, 2005.
6. Y. LeCun, Y. Bengio, and G. Hinton, "Deep Learning," *Nature*, vol. 521, no. 7553, 2015, pp. 436–444.
7. N. Wang and D.-Y. Yeung, "Learning a Deep Compact Image Representation for Visual Tracking," *Proc. 26th Int'l Conf. Neural Information Processing Systems* (NIPS 13), 2013, pp. 809–817.
8. E. Galceran et al., "Multipolicy Decision-Making for Autonomous Driving via Changepoint-Based

- Behavior Prediction: Theory and Experiment," *Autonomous Robots*, 2017; doi:10.1007/s10514-017-9619-z.
9. E. Frazzoli, M.A. Dahleh, and E. Feron, "Real-Time Motion Planning for Agile Autonomous Vehicles," *AIAA J. Guidance, Control, and Dynamics*, vol. 25, no. 1, 2002, pp. 116–129.
 10. M. Althoff, O. Stursberg, and M. Buss, "Model-Based Probabilistic Collision Detection in Autonomous Driving," *IEEE Trans. Intelligent Transportation Systems*, vol. 10, no. 2, 2009, pp. 299–310.
 11. D. Scaramuzza and F. Fraundorfer, "Visual Odometry Part I: The First 30 Years and Fundamentals [Tutorial]," *IEEE Robotics & Automation*, vol. 18, no. 4, 2011; rpg.ifi.uzh.ch/docs/Visual_Odometry_Tutorial.pdf.
 12. P.J. Besl and D.M. Neil, A Method for Registration of 3D Shapes," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, 1992, pp. 239–256.
 13. S. Liu et al., "Achieving Energy Efficiency through Runtime Partial Reconfiguration on Reconfigurable Systems," *ACM Trans. Embedded Computing Systems*, vol. 12, no. 3, 2013; doi:10.1145/2442116.2442122.

myCS Read your subscriptions through the myCS publications portal at <http://mycs.computer.org>

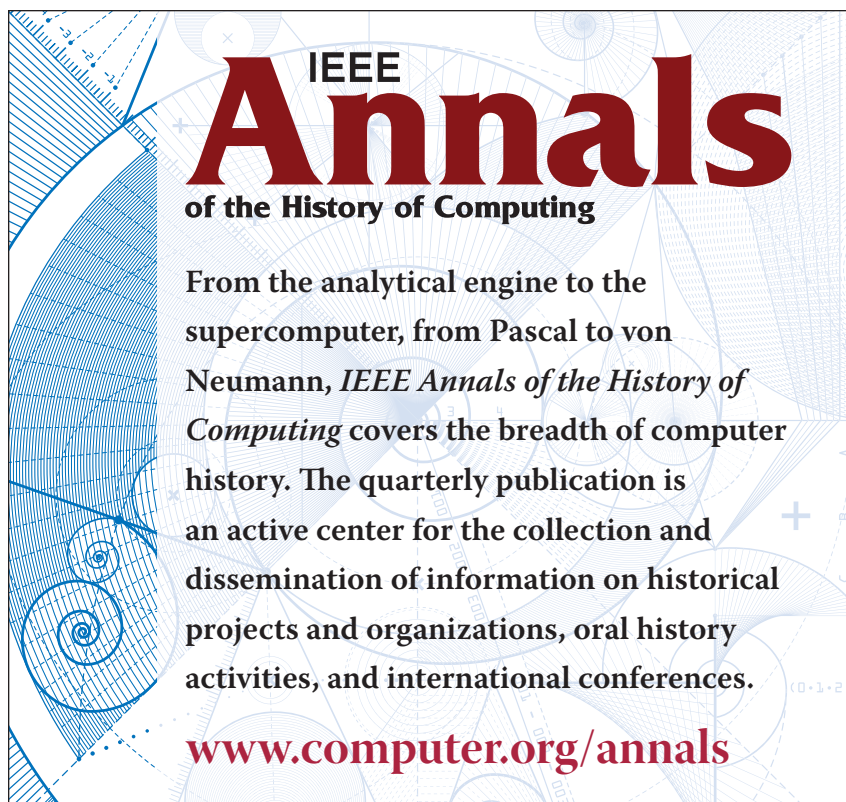
ABOUT THE AUTHORS

SHAOSHAN LIU is a cofounder of PerceptIn. His research interests include computer architecture, big data platforms, deep-learning infrastructure, and robotics. Liu received a PhD in computer engineering from the University of California, Irvine (UCI). He is a member of IEEE. Contact him at shaoshan.liu@perceptin.io.

JIE TANG is an associate professor in the School of Computer Science and Engineering at the South China University of Technology. Her research interests include autonomous driving, computer architecture, and cloud computing. Tang received a PhD in computer science from the Beijing Institute of Technology. She is a member of IEEE. Contact her at cstangjie@scut.edu.cn.

ZHE ZHANG is a cofounder of PerceptIn. His research interests include vision-based robotic 3D mapping and localization, navigation and self-recharge solutions for consumer robotics, and sparse and dense mapping and pose tracking. Zhang received a PhD in robotics from the State University of New York. Contact him at zhe.zhang@perceptin.io.

JEAN-LUC GAUDIOT is a professor in the Electrical Engineering and Computer Science Department at UCI. His research interests include multithreaded architectures, fault-tolerant multiprocessors, and the implementation of reconfigurable architectures. Gaudiot received a PhD in computer science from the University of California, Los Angeles. He is a Fellow of IEEE and president of the IEEE Computer Society. Contact him at gaudiot@uci.edu.



IEEE
Annals
of the History of Computing

From the analytical engine to the supercomputer, from Pascal to von Neumann, *IEEE Annals of the History of Computing* covers the breadth of computer history. The quarterly publication is an active center for the collection and dissemination of information on historical projects and organizations, oral history activities, and international conferences.

www.computer.org/annals