
AN OPEN APPROACH TO AUTONOMOUS VEHICLES

AUTONOMOUS VEHICLES ARE AN EMERGING APPLICATION OF AUTOMOTIVE TECHNOLOGY, BUT THEIR COMPONENTS ARE OFTEN PROPRIETARY. THIS ARTICLE INTRODUCES AN OPEN PLATFORM USING COMMODITY VEHICLES AND SENSORS. THE AUTHORS PRESENT ALGORITHMS, SOFTWARE LIBRARIES, AND DATASETS REQUIRED FOR SCENE RECOGNITION, PATH PLANNING, AND VEHICLE CONTROL. RESEARCHERS AND DEVELOPERS CAN USE THE COMMON INTERFACE TO STUDY THE BASIS OF AUTONOMOUS VEHICLES, DESIGN NEW ALGORITHMS, AND TEST THEIR PERFORMANCE.

Shinpei Kato
Eijiro Takeuchi
Yoshio Ishiguro
Yoshiki Ninomiya
Kazuya Takeda
Nagoya University
Tsuyoshi Hamada
Nagasaki University

.....Autonomous vehicles are becoming a new piece of infrastructure. Automotive makers, electronics makers, and IT service providers are interested in this technology, and academic research has contributed significantly to producing their prototype systems. For example, one notable work was published by Carnegie Mellon University.¹

Despite this trend, autonomous vehicles are not systematically organized. Given that commercial vehicles protect their in-vehicle system interface from users, third-party vendors cannot easily test new components of autonomous vehicles. In addition, sensors are not identical. Some vehicles might use only cameras, whereas others might use a combination of cameras, laser scanners, GPS receivers, and milliwave radars.

In addition to hardware issues, autonomous vehicles must address software issues. Because an autonomous-vehicles platform is largescale, it is inefficient to build it up from scratch, especially for prototypes. Open-source software libraries are preferred for that purpose, but they have not yet been inte-

grated to develop autonomous vehicles. The design and implementation of algorithms for scene recognition, path planning, and vehicle control also require multidisciplinary skills and knowledge, often incurring a significant engineering effort. Finally, a basic dataset, such as a map for localization, must be provided to drive on a public road.

Overall, autonomous vehicles are composed of diverse technologies. Building their platform requires a multidisciplinary collaboration in research and development. To facilitate this collaboration, we introduce an open platform for autonomous vehicles that many researchers and developers can study to obtain a baseline for autonomous vehicles, design new algorithms, and test their performance, using a common interface.

Vehicles and sensors

We introduce an autonomous-vehicles platform with a set of sensors that can be purchased in the market. We assume that intelligent modules of autonomous driving, such as scene recognition, path planning, and vehicle control,



Figure 1. ZMP Robocar “HV” and examples of sensors and plug-in computers. Omni view cameras, Lidar sensors and GNSS receivers are equipped outside of the vehicle, while single-view cameras and computers are set inside.

are located in a plug-in computer connected to vehicular Controller Area Network (CAN) bus networks. Given that commercial vehicles are not designed to employ such a plug-in computer, we need to add a secure control gateway to the CAN bus networks. This is often a complex undertaking that prevents researchers and developers from using real-world vehicles.

The ZMP Robocar product provides a vehicle platform containing a control gateway through which a plug-in computer can send operational commands (such as pedal strokes and steering angles) to the vehicle. Figure 1 shows one of their product lines, the ZMP Robocar “HV,” which is based on the Toyota Prius. (Details about the product are available at www.zmp.co.jp.) We added a plug-in computer and a set of sensors, such as cameras and light detection and ranging (Lidar) sensors, to this basic platform so that the results of scene recognition, path planning, and vehicle control could apply to autonomous driving.

The specification of sensors and computers highly depends on the functional requirements of autonomous driving. Our prototype system uses various sensors and computers (see Figure 1):

- Velodyne Lidar sensors produce 3D point-cloud data, which can be used for localization and mapping, while also being used to measure the distance to surrounding objects.
- Ibeo Lidar sensors produce long-range 3D point-cloud data, although their vertical resolution is lower than that of the Velodyne Lidar sensors.
- Hokuyo Lidar sensors produce short-range 2D laser scan data and are useful for emergent stopping rather than for localization and mapping.
- Point Grey Ladybug 5 and Grasshopper 3 cameras can detect objects. The Ladybug 5 camera is omnidirectional, covering a 360-degree view, whereas the Grasshopper 3 camera is single-directional, operating at a high rate. The former can be used to detect moving objects, and the latter can be used to recognize traffic lights.
- Javad RTK sensors receive global positioning information from satellites. They are often coupled with gyro sensors and odometers to fix the positioning information.

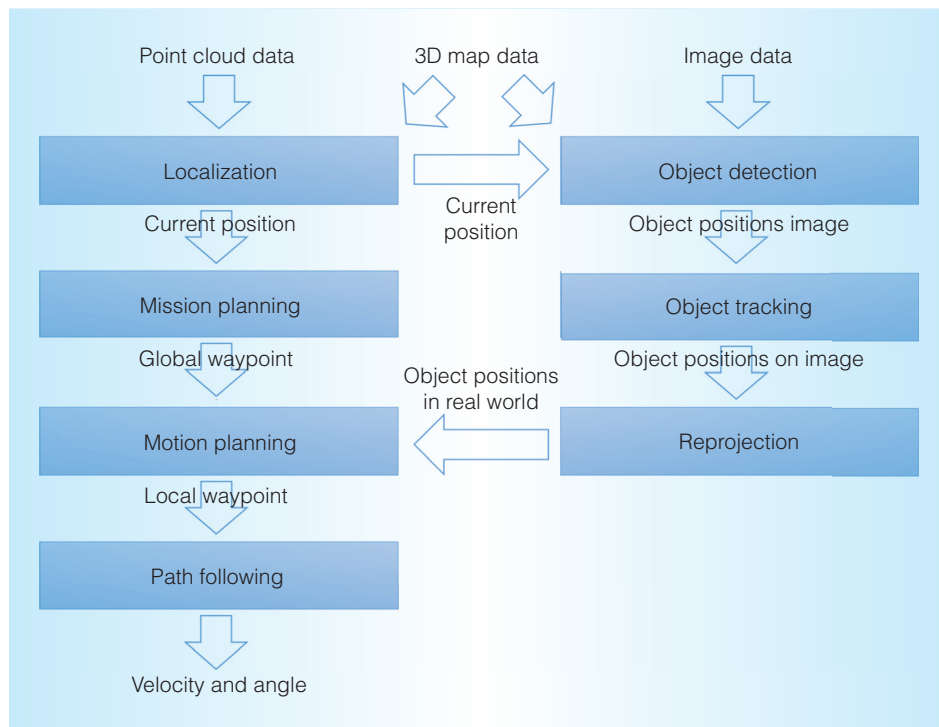


Figure 2. Basic control and dataflow of algorithms. The output velocity and angle are sent as commands to the vehicle controller.

These sensors can be connected to commodity network interfaces, such as Ethernet and USB 3.0. Note that sensors for autonomous vehicles are not limited to them. For example, milliwave radars and inertial measurement units are often preferred for autonomous vehicles.

Regarding a plug-in computer, because commercial vehicles support only a DC 12-V power supply, we add an inverter and battery to provide an AC 100-V power supply for the prototype vehicle in Figure 1. Note that the choice of a plug-in computer depends on performance requirements for autonomous vehicles. In our project, we started with a high-performance workstation to test algorithms. Once the algorithms were developed, we switched it to a mobile laptop to downsize the system. Now, we are aiming to use an embedded system on a chip (SoC), taking into account the production image.

Algorithms

We can roughly classify autonomous-driving components into scene recognition, path planning, and vehicle control. Each class com-

prises a set of algorithms. For instance, scene recognition requires localization, object-detection, and object-tracking algorithms. Path planning often falls into mission and motion planning, whereas vehicle control corresponds to path following.

Figure 2 shows the algorithms' basic control and data flow. Here, we introduce examples of the algorithms used in our autonomous-vehicles platform.

Localization

Localization is one of the most basic and important problems in autonomous driving. Particularly in urban areas, localization precision dominates the reliability of autonomous driving. We use the Normal Distributions Transform (NDT) algorithm to solve this localization problem.² To be precise, we use the 3D version of NDT to perform scan matching over 3D point-cloud data and 3D map data.³ As a result, localization can perform at the order of centimeters, leveraging a high-quality 3D Lidar sensor and a high-precision 3D map. We chose the NDT algorithms because they can be used in 3D forms

and their computation cost does not suffer from map size (the number of points).

Localization is also a key technique to build a 3D map. Because 3D Lidar sensors produce 3D point-cloud data in real time, if our autonomous vehicle is localized correctly, a 3D map is created and updated by registering the 3D point-cloud data at every scan. This is often referred to as simultaneous localization and mapping.

Object detection

Once we localize our autonomous vehicle, we next detect objects, such as vehicles, pedestrians, and traffic signals, to avoid accidents and violation of traffic rules. We focus on moving objects (vehicles and pedestrians), although our platform can also recognize traffic signals and lights. We use the Deformable Part Models (DPM) algorithm to detect vehicles and pedestrians.⁴ DPM searches and scores the Histogram of Oriented Gradients features of target objects on the image captured from a camera.⁵ We chose these algorithms because they scored the best numbers in past Pattern Analysis, Statistical Modeling, and Computational Learning (Pascal) Visual Object Classes challenges.

Apart from image processing, we also use point-cloud data scanned from a 3D Lidar sensor to detect objects by Euclidean clustering. Point-cloud clustering aims to obtain the distance to objects rather than to classify them. The distance information can be used to range and track the objects classified by image processing. This combined approach with multiple sensors is often referred to as *sensor fusion*.

Operating under the assumption that our autonomous vehicle drives on a public road, we can improve the detection rate using a 3D map and the current position information. Projecting the 3D map onto the image originated on the current position, we know the exact road area on the image. We can therefore constrain the region of interest for image processing to this road area so that we can save execution time and reduce false positives.

Object tracking

Because we perform the object-detection algorithm on each frame of the image and point-cloud data, we must associate its results

with other frames on a time basis so that we can predict the trajectories of moving objects for mission and motion planning. We use two algorithms to solve this tracking problem. Kalman Filters is used under a linear assumption that our autonomous vehicle is driving at constant velocity while tracking moving objects.⁶ Its computational cost is lightweight and suited for real-time processing. Particle Filters, on the other hand, can work for nonlinear tracking scenarios, in which both our autonomous vehicle and tracked vehicles are moving.⁷

In our platform, we use both Kalman Filters and Particle Filters, depending on the given scenario. We also apply them for tracking on both the 2D (image) plane and the 3D (point-cloud) plane.

Projection and reprojection

We augment scene recognition supported by our platform with sensor fusion of a camera and a 3D Lidar sensor. To calculate the extrinsic parameters required to make this sensor fusion, we calibrate the camera and the 3D Lidar sensor. We can then project the 3D point-cloud information obtained by the 3D Lidar sensor onto the image captured by the camera so that we can add depth information to the image and filter out the region of interest of object detection.

The result of object detection on the image can also be reprojected onto the 3D point-cloud coordinates using the same extrinsic parameters. We use the reprojected object positions to determine the motion plan and, in part, the mission plan.

Mission planning

Our mission planner is semiautonomous. Under the traffic rules, we use a rule-based mechanism to autonomously assign the path trajectory, such as for lane change, merge, and passing. In more complex scenarios, such as parking and recovering from operational mistakes, the driver can supervise the path. In either case, once the path is assigned, the local motion planner is launched.

Our mission planner's basic policy is that we drive on the cruising lane throughout the route provided by a commodity navigation application. The lane is changed only when our autonomous vehicle is passing the

preceding vehicle or approaching an intersection followed by a turn.

Motion planning

The motion planner is a design knob for autonomous driving that corresponds to the driving behavior, which is not identical among users and environments. Hence, our platform provides only a basic motion-planning strategy. A high-level intelligence must be added on top of this platform, depending on the target scenarios.

In unstructured environments, such as parking lots, we provide graph-search algorithms, such as A*,⁸ to find a minimum-cost path to the goal in space lattices.⁹ In structured environments, such as roads and traffic lanes, on the other hand, the density of vertices and edges is likely high and not uniform, which constrains the selection of feasible headings. We therefore use conformal spatiotemporal lattices to adapt the motion plan to the environment.¹⁰ State-of-the-art research encourages the implementation of these algorithms.¹

Path following

We control our autonomous vehicle to follow the path generated by the motion planner. We use the Pure Pursuit algorithm to solve this path-following problem.¹¹ According to the Pure Pursuit algorithm, we break down the path into multiple waypoints, which are discrete representations of the path. At every control cycle, we search for the close waypoint in the heading direction. We limit the search to outside of the specified threshold distance so that we can relax the change in angle in the case of returning onto the path from a deviated position. The velocity and angle of the next motion are set to such values that bring the vehicle to the selected waypoint with predefined curvature.

We update the target waypoint accordingly until the goal is reached. The vehicle keeps following updated waypoints and finally reaches the goal. If the control of gas and brake stroking and steering is not aligned with the velocity and angle output of the Pure Pursuit algorithm because of some noise, the vehicle could temporarily get off the path generated by the motion planner. Localization errors could also pose this problem. As a result, the vehicle could come

across unexpected obstacles. To cope with this scenario, our path follower ensures a minimum distance to obstacles, overwriting the given plan.

Software stack

We build the software stack of autonomous driving on the basis of open-source software. As a result, even the code implementation of the algorithms we have discussed becomes accessible to the public. The software-stack framework is called Autoware, and it can be downloaded from the project repository (<https://github.com/cpf/autoware>).

In this section, we introduce the core components of open-source software used in Autoware. Our autonomous vehicle is entirely operated by Autoware. It has already run over a few hundred miles in Nagoya, Japan.

Robot Operating System

Autoware is based on Robot Operating System (www.ros.org), a component-based middleware framework developed for robot applications. In ROS, the system is abstracted by nodes and topics. The nodes represent individual component modules, whereas the topics hold input and output data between nodes. This is a strong abstraction model for compositional development.

ROS nodes are usually standard C++ programs. They can use any other software libraries installed in the system. Meanwhile, ROS nodes can launch several threads implicitly. Topics are also managed by first-in, first-out queues when accessed by multiple nodes simultaneously. Real-time issues, however, must be addressed.

ROS also provides an integrated visualization tool called RViz. Figure 3 shows an example of visualization for perception tasks in Autoware. The RViz viewer is useful for checking the status of tasks.

Point-Cloud Library

Point-Cloud Library (<http://pointclouds.org>) was developed to manage point-cloud data. It supports many algorithms in the library package, including the 3D NDT algorithm.³ Autoware uses this PCL version of 3D NDT for localization and mapping. In our experience, the localization or mapping error can be capped at 10 to 20 cm.

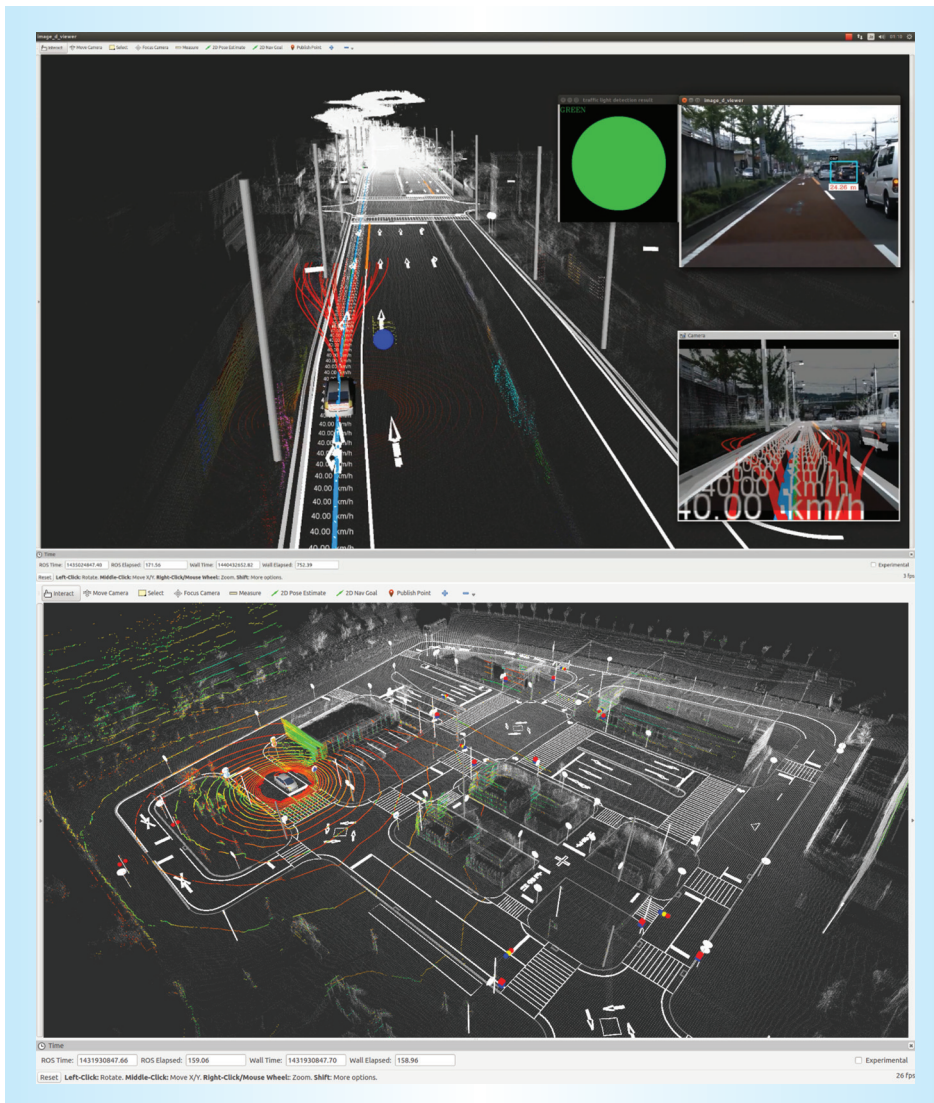


Figure 3. Visualization tools for developers. Both a 3D map plane and a 2D image plane can be displayed. The results of traffic light recognition and path generation are also displayed.

Another usage of PCL in Autoware is to implement the Euclidean clustering algorithm for object detection. In addition, ROS uses PCL internally in many places. For example, the RViz viewer provided by ROS is a PCL application.

OpenCV

OpenCV (<http://opencv.org>) is a popular computer vision library for image processing. It supports the DPM algorithm⁴ and the Histogram of Oriented Gradients features.⁵ Apart from algorithm implementation, OpenCV provides API library functions (such as load-

ing, converting, and drawing images), which are useful to construct a framework of image-processing programs. Autoware combines OpenCV with the RViz viewer of ROS for visualization.

CUDA

CUDA (<https://developer.nvidia.com/cuda-zone>) is a framework for general-purpose computing on GPUs (GPGPU). Because complex algorithms of autonomous driving, such as NDT, DPM, and A*, are often computing intensive and data parallel, their execution speeds can be improved significantly using



Figure 4. User interface tools for drivers. Smart tablets are used to input the destination of the trip, displaying a route candidate. To present more valuable information to drivers, smart glasses and displays can be used together.

CUDA. For example, the DPM algorithm contains a nontrivial number of computing-intensive and data-parallel blocks that can be significantly accelerated by CUDA.¹² Because autonomous vehicles need real-time performance, CUDA is a strong way to speed up the algorithms.

Android

Autware uses Android (www.android.com) applications for the human-driver interface. The driver can search for the route using a navigation map on an Android tablet (see Figure 4). The result of a searched route is reflected to the 3D map used in Autware.

openFrameworks

Another piece of software used in Autware for the driver interface is openFrameworks (www.openframeworks.cc). Although Android applications are functional to receive an input from the driver, openFrameworks provides creative displays to visualize the status of autonomous driving to the driver. For exam-

ple, it can be used with smart glasses, as shown in Figure 4.

Datasets

In our platform, a 3D map is required to localize the autonomous vehicle. As we mentioned earlier, we can generate a 3D map using Autware. However, this turns into a time-consuming routine. It should be provided by mapmakers as part of general datasets for autonomous driving. We are collaborating with Aisan Technology, a mapmaker in Japan, for field-operational tests of autonomous driving in Nagoya. Autware is designed to be able to use both their 3D maps and those generated by Autware itself. This compatibility to the third-party format allows Autware to be deployed widely in industry and academia.

Simulation of autonomous driving also needs datasets. Field-operational tests always take some risk, and they are not always efficient in time, so simulation is desired in most cases. Fortunately, ROS provides an excellent

record-based simulation framework called ROSBAG. We suggest that ROSBAG is used primarily to test functions of autonomous vehicles, and field-operational tests are performed for a real verification.

Autoware lets researchers and developers use such structured 3D maps and record data. Simulation of autonomous driving on a 3D map can be easily conducted using Autoware. Samples of these datasets can also be downloaded through the Autoware website. Given that vehicles and sensors are often expensive, simulation with the sample datasets might be preferred to on-site experiments in some cases.

Performance requirements

Considering that cameras and Lidar sensors often operate at 10 to 100 Hz, each task of autonomous driving must run under some timing constraints.

We provide a case study of our autonomous driving system on several computers employing Intel CPUs and Nvidia GPUs. On Intel CPUs, such as the Xeon and Core i7 series, the A* search algorithm consumes the most time, requiring a scale of seconds or more, if the search area is large. Although faster is better, this might not be a significant problem given that A* search is often used for mission planning, which launches only when the path needs to change. A more significant problem resides in real-time tasks, such as motion planning, localization, and object detection. The DPM algorithm spends more than 1,000 ms on VGA-size images with the default parameter setting in OpenCV, whereas the NDT and the State Lattice algorithms can run in several tens of milliseconds. Their execution times must be reduced to drive autonomous vehicles in the real world.

We implemented these algorithms using Nvidia GPUs. As reported elsewhere,¹² Nvidia GPUs bring 5 to 10 times performance improvements over Intel CPUs for the DPM algorithm. In fact, we also observed equivalent effects on the A*, NDT, and State Lattice algorithms, although the magnitude of improvement depends on the parameters.

One particular case study of our autonomous driving system used a laptop composed of an Intel Core i7-4710MQ CPU and an Nvidia GTX980M GPU. We demonstrated that most real-time tasks, including the NDT

and State Lattice algorithms, can run within 50 ms, whereas the DPM algorithm still consumes more than 100 to 200 ms on the GPU. This implies that our system currently exhibits a performance bottleneck in object detection. Assuming that the vehicle is self-driving at 40 km/hour in urban areas and that autonomous functions should be effective every 1 m, the execution time of each real-time task must be less than 100 ms, although a multitasking environment could make the problem more complicated. If our autonomous driving system remains as is, the performance of GPUs must improve by at least a factor of two to meet our assumption. Application-specific integrated circuit and field-programmable gate array solutions could also exist.

Real-time tasks other than planning, localization, and detection are negligible in terms of execution time, but some tasks are very sensitive to latency. For example, a vehicle-control task in charge of steering and acceleration and brake stroking should run with a few-milliseconds latency.

Finally, because such throughput- and latency-aware tasks are coscheduled in the same system, operating systems must be reliable to multitask real-time processing. This leads to the conclusion that codesign of hardware and software is an important consideration for autonomous driving.

This article has introduced an open platform for commodity autonomous vehicles. The hardware components, including ZMP Robocars, sensors, and computers, can be purchased in the market. The autonomous-driving algorithms are widely recognized, and Autoware can be used as a basic software platform comprising ROS, PCL, OpenCV, CUDA, Android, and openFrameworks. Sample datasets, including 3D maps and simulation data, are also packaged as part of Autoware. To the best of our knowledge, this is the first autonomous-vehicles platform accessible to the public.

MICRO

References

1. C. Urmson et al., "Autonomous Driving in Urban Environments: Boss and the Urban Challenge," *J. Field Robotics*, vol. 25, no. 8, 2008, pp. 425–466.

2. P. Biber et al., "The Normal Distributions Transform: A New Approach to Laser Scan Matching," *Proc. IEEE/RSJ Int'l Conf. Intelligent Robots and Systems*, 2003, pp. 2743–2748.
3. M. Magnusson et al., "Scan Registration for Autonomous Mining Vehicles Using 3D-NDT," *J. Field Robotics*, vol. 24, no. 10, 2007, pp. 803–827.
4. P. Felzenszwalb et al., "Object Detection with Discriminatively Trained Part-Based Models," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 32, no. 9, 2010, pp. 1627–1645.
5. N. Dalal and B. Triggs, "Histograms of Oriented Gradients for Human Detection," *Proc. IEEE CS Conf. Computer Vision and Pattern Recognition*, 2005, pp. 886–893.
6. R. Kalman, "A New Approach to Linear Filtering and Prediction Problems," *ASME J. Basic Eng.*, vol. 82, no. 1, 1960, pp. 35–45.
7. M. Arulampalam et al., "A Tutorial on Particle Filters for Online Nonlinear/non-Gaussian Bayesian Tracking," *IEEE Trans. Signal Processing*, vol. 50, no. 2, 2002, pp. 174–188.
8. P. Hart et al., "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Trans. Systems Science and Cybernetics*, July 1968, pp. 100–107.
9. M. Pivtoraiko et al., "Differentially Constrained Mobile Robot Motion Planning in State Lattices," *J. Field Robotics*, vol. 26, no. 3, 2009, pp. 308–333.
10. N. McNaughton et al., "Motion Planning for Autonomous Driving with a Conformal Spatiotemporal Lattice," *Proc. IEEE Int'l Conf. Robotics and Automation*, 2011, pp. 4889–4895.
11. R. Coulter, *Implementation of the Pure Pursuit Path Tracking Algorithm*, tech. report CMURI-TR-92-01, Robotics Institute, 1992.
12. M. Hirabayashi et al., "Accelerated Deformable Part Models on GPUs," to be published in *IEEE Trans. Parallel and Distributed Systems*, 2015.

include operating systems, cyber-physical systems, and parallel and distributed systems. Kato has a PhD in engineering from Keio University. Contact him at shinpei@is.nagoya-u.ac.jp.

Eijiro Takeuchi is a designated associate professor in the Institute of Innovation for Future Society at Nagoya University. His research interests include autonomous mobile robots and intelligent vehicles. Takeuchi has a PhD in engineering from the University of Tsukuba. Contact him at takeuchi@coi.nagoya-u.ac.jp.


Yoshio Ishiguro is an assistant professor in the Institute of Innovation for Future Society at Nagoya University. His research interests include mixed reality, augmented reality, and human–computer integration. Ishiguro has a PhD in information science and studies from the University of Tokyo. Contact him at ishiy@acm.org.

Yoshiki Ninomiya is a designated professor in the Institute of Innovation for Future Society at Nagoya University. His research interests include machine intelligence and image processing, and its applications. Ninomiya has a PhD in engineering from Nagoya University. Contact him at ninomiya@coi.nagoya-u.ac.jp.

Kazuya Takeda is a professor in the School of Information Science at Nagoya University. His research interests include media signal processing and its applications. Takeda has a PhD in engineering from Nagoya University. Contact him at kazuya.takeda@nagoya-u.jp.

Tsuyoshi Hamada is an associate professor in the Nagasaki Advanced Computing Center at Nagasaki University. His research interests include massively parallel architectures based on FPGAs and GPUs. Hamada has a PhD in engineering from the University of Tokyo. Contact him at hamada@nacc.nagasaki-u.ac.jp.

Shinpei Kato is an associate professor in the Graduate School of Information Science at Nagoya University. His research interests

 Selected CS articles and columns are also available for free at <http://ComputingNow.computer.org>.