Development of Autonomous Car—Part II: A Case Study on the Implementation of an Autonomous Driving System Based on Distributed Architecture

Kichun Jo, *Member, IEEE*, Junsoo Kim, *Student Member, IEEE*, Dongchul Kim, Chulhoon Jang, *Student Member, IEEE*, and Myoungho Sunwoo, *Member, IEEE*

Abstract—Part I of this paper proposed a development process and a system platform for the development of autonomous cars based on a distributed system architecture. The proposed development methodology enabled the design and development of an autonomous car with benefits such as a reduction in computational complexity, fault-tolerant characteristics, and system modularity. In this paper (Part II), a case study of the proposed development methodology is addressed by showing the implementation process of an autonomous driving system. In order to describe the implementation process intuitively, core autonomous driving algorithms (localization, perception, planning, vehicle control, and system management) are briefly introduced and applied to the implementation of an autonomous driving system. We are able to examine the advantages of a distributed system architecture and the proposed development process by conducting a case study on the autonomous system implementation. The validity of the proposed methodology is proved through the autonomous car A1 that won the 2012 Autonomous Vehicle Competition in Korea with all missions completed.

Index Terms—Autonomous car, development process, distributed system, FlexRay, system platform.

I. INTRODUCTION

SAFETY regulations in the automotive industry have increasingly become stringent along with the growing customer demands for comfort functionality. For this reason, the development of intelligent vehicle technology has been accelerated. The ultimate goal of intelligent vehicle technology is the realization of an autonomous car that can drive itself without collisions by perceiving the surrounding environment.

The Defense Advanced Research Projects Agency opened the Grand and Urban Challenge competition in the USA to stimulate research for autonomous cars [1]–[4]. The competition helped promote the feasibility of autonomous cars; consequently, global automakers and information technology companies have endeavored to develop commercial autonomous cars.

In Korea, two autonomous vehicle competitions (AVCs) were held in 2010 and 2012 by Hyundai Motor Group in order to establish the foundation of autonomous driving technology. The 2010 AVC focused on fundamental technology such as waypoint tracking and static obstacle avoidance [5], [6]. After two years, the 2012 AVC promoted the development of autonomous driving technology in urban driving environments containing elements such as traffic signals, moving vehicles, and pedestrians [7], [8]. This paper (Part II of a two-part paper) represents the results of autonomous car A1, which was the winner of the 2012 AVC.

In Part I of this two-part paper [9], we proposed a development process for a distributed system. The development process was reformed to improve flexibility and scalability and to discard unnecessary processes based on an AUTomotive Open System ARchitecture (AUTOSAR) methodology and platform [10]–[14]. The process consists of three steps (software component design, computing unit mapping, and the implementation of functions). In the software component design step, software components for autonomous driving were designed, and the flow of information between the components was defined with a virtual functional bus (VFB), which is an abstraction of communication. Second, the designed software components were mapped to each distributed computing unit by considering the computing power, the mounting position, and operating systems (OSs). Finally, a software platform implemented the software components in the assigned computing unit.

Part II of this paper presents a case study on the implementation of an autonomous driving system based on the *development process and the distributed system architecture* that were introduced in Part I by using an autonomous car, i.e., A1. Various autonomous driving algorithms (perception, localization, planning, and control) and system components

TABLE I
TIME PENALTIES AND CREDITS OF MISSIONS DEFINED IN AVC

Missions	Penalty (credit)
Traffic light and crosswalk detection	+2 min
Passenger pickup	+2 min
Overtaking	+2 min
School zone and sudden obstacle	+2 min
Barrier	+2 min
Construction site	+2 min
Split road	+2 min
Complex obstacle avoidance	+1 min
Parking (reverse)	+2 min (-2 min)

of autonomous cars (many heterogeneous sensors, actuators, and computers) are integrated and implemented based on the proposed development process. Each step of the development process (software component design, computing unit mapping, and the implementation of functions) is explained using the implementation example of autonomous car A1. The implementation results show that we can obtain many benefits by using the proposed development process and distributed system architecture.

This paper is organized as follows. Section II presents the 2012 AVC, and Section III provides a system overview and the autonomous driving algorithm of A1. Section IV presents the implementation of the autonomous driving algorithm with the proposed process and platform. Section V presents the autonomous driving results of A1, and Section VI provides the conclusions. The logical flow and experimental results of Part II are tightly coupled with Part I; therefore, it is recommended to read the development methodology of Part I before referring to this paper.

II. INTRODUCTION OF 2012 AVC

To enhance the research of autonomous driving systems, the 2012 AVC aimed at the development of autonomous cars for urban driving. The missions are composed of urban driving scenarios such as traffic signal detection and overtaking slow vehicles, as shown in Table I. Therefore, a participating autonomous car should not only drive a racetrack by itself but also complete the various missions. The racetrack consists of 1.6 km on-road and 1.8 km off-road, as shown in Fig. 1. The red and blue dots indicate the start and finish positions, respectively, and the arrows in the racetrack represent the driving directions. Track-related map information (including waypoints) was provided by the AVC organizers. During the race, most urban-related missions were performed on-road; alternatively, harsh and emergency driving conditions were performed off-road.

Penalties or additional credits are defined for each mission, as shown in Table I. The purpose of the traffic-light and crosswalk detection missions is to demonstrate the ability to detect the status of traffic lights and crosswalk locations, respectively. The vehicle should stop at or pass the crosswalk according to the traffic-light status. The passenger pickup mission aims at detecting a passenger and stopping within 5 m. In the overtaking mission, the vehicle must safely overtake a slow vehicle. In the school zone and sudden obstacle missions, the vehicle should not only comply with the speed limit but also
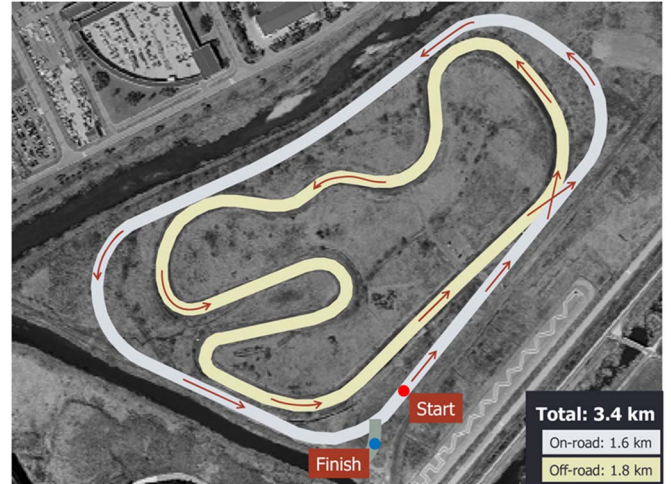

Fig. 1. Race track of the 2012 AVC.

perform an emergency stop that avoids accidents with unexpected obstacles. In the barrier mission, a vehicle should detect a barrier that blocks the road and should stop within 5 m of the barrier. In the construction-site driving mission, a vehicle must pass a construction site that was not provided on the map. The objective of the split road mission is recognizing and following the direction of the traffic light for a split road. In the complex obstacle avoidance mission, the vehicle should find a drivable path without colliding with the complex obstacles. The parking mission aims at detecting a parking lot number and a parking region; the vehicle then parks by itself in the corresponding parking position. If the vehicle succeeds in reverse parking, an additional credit is provided, as shown in Table I. The final winner of the AVC was determined by the sum of the travel time and the penalties for each mission.

## III. AUTONOMOUS CAR A1: SYSTEM AND ALGORITHM

A. System Overview of Autonomous Car A1

Fig. 2 describes autonomous car A1 and its sensor configuration. The vehicle platform was equipped with an electronics stability control (ESC) system. ESC is used to improve the vehicle dynamic stability by detecting the abnormal vehicle motion and controlling the vehicle motion using a brake system. In order to detect the abnormal vehicle motion, many types of onboard sensors are used for ESC, including wheel-speed sensors, a steering-wheel-angle sensor, and a yaw-rate sensor. The onboard ESC sensor information is shared through the in-vehicle network, i.e., the controller area network (CAN). Therefore, we can access the onboard sensor information of the ESC by connecting to the CAN. A motor-driven power steering system is used to control the steering, and the acceleration pedal signal is emulated for the acceleration. A direct-current motor that is mechanically connected to the brake pedal is used for braking. The gear shift lever is controlled by a motor to determine the direction of the vehicle.

Eight laser scanners are installed on A1, as shown in Fig. 2(b). Two multilayer laser scanners (Ibeo LUX) that measure obstacles up to 200 m away in an optimal environment

(a)
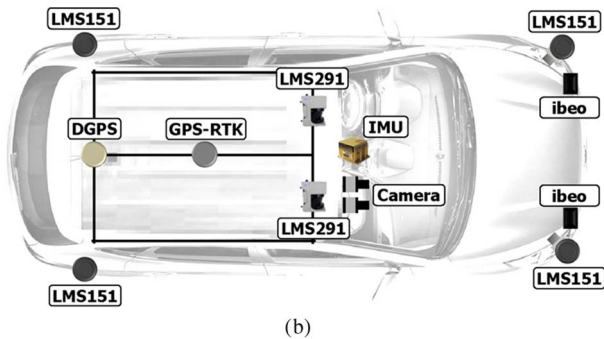
(b)

Fig. 2. Vehicle platform and sensor configuration of A1.

are mounted on the front bumper to detect distant objects. To detect the adjacent objects around the ego vehicle, four single-layer laser scanners (LMS 151) are installed on each corner. Two single-layer laser scanners (LMS 291) that scan vertically to the ground are installed on the roof to detect barriers, which are thin bars placed horizontally to the ground.

To detect and classify the mission objects, one color camera and three monocameras are installed on the inside of the windshield. The two types of GPS receivers, i.e., a Real-Time Kinematics GPS (RTK-GPS) and a Differential GPS (DGPS), are equipped in the vehicle to measure the position of the ego vehicle in a global coordinate frame. An inertial measurement unit (IMU) that is located at the center of the vehicle is used to estimate the vehicle's dynamic motion.

B. Autonomous Driving Algorithm

In order to autonomously drive without human intervention, an autonomous car requires five basic functions, i.e., localization, perception, planning, vehicle control, and system management (see Fig. 3). The localization is responsible for the estimation of the vehicle position, and the perception derives a model of the driving environment from multisensor fusion-based information. Based on the localization and perception information, the planning function determines the maneuvers of the autonomous car for safe vehicle navigation. The vehicle control function follows the desired command from the planning function by steering, accelerating, and braking the autonomous car. Finally, the system management supervises the overall autonomous driving system. Detailed descriptions of the autonomous driving algorithm of A1 are shown in the following.
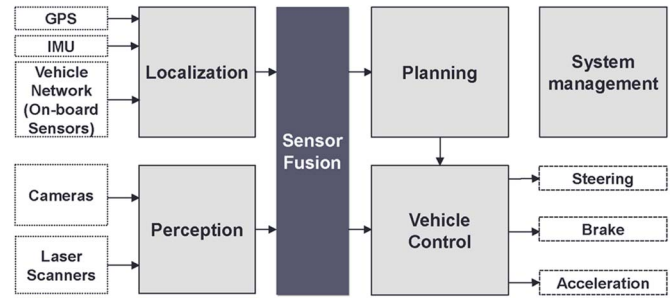

Fig. 3. Structure of the autonomous driving algorithms in A1.

1) Localization: A localization system is an essential component of an autonomous car since autonomous cars find optimal paths and control the vehicle motion only if the ego-vehicle position from the localization system is available. A GPS is widely used for localization systems because it directly provides a global position and the ego vehicle's velocity. However, the raw data position of the GPS cannot be used for an autonomous driving system since the quality of the GPS position is significantly affected by satellite signal conditions. The accuracy, reliability, and continuity of the measured GPS position data will rapidly deteriorate when GPS satellite signal conditions are unstable due to blockage and multipaths.

A lot of previous research focused on the fusion of a GPS with additional information such as vehicle motion sensors (wheel-speed sensors, gyros, accelerometers, and magnetic sensors) [15], [16], environment perception data [17], and digital maps [18] in order to compensate for GPS vulnerabilities. The basic principle of an information fusion-based localization system is that GPS position errors are corrected by another information source such as vehicle motion constraints and correction data from matching the perceived landmark with a digital map.

In this paper, in order to cover the various driving conditions, an interacting multiple model (IMM)-filter-based information fusion system is used for the localization system [16], [19]. The localization system can adapt to changing vehicle dynamic characteristics under various driving conditions since the IMM filter selects the kinematics and dynamics model according to driving conditions. A GPS-bias correction algorithm was also applied to the localization system in order to improve the accuracy and reliability of the localization system [18].

2) Perception: The perception system offers information on surrounding environments using several types of sensor components such as cameras, radars, and laser scanners. A range-sensor-based (a radar and a laser scanner) perception system detects and tracks static and dynamic obstacles [20], [21], whereas a vision-based perception system recognizes various visual objects [22], [23]. The data of the detected and recognized objects are used for the situation assessment of the autonomous driving system.

The perception system of A1 consists of a laser-scanner-based moving object tracking algorithm and a vision-based object detection and classification algorithm. The object tracking system of A1 uses four LMS151s and two Ibeo LUXs to track moving vehicles around the ego vehicle. In particular, an integrated-probabilistic-data-association-filter-based

TABLE II
SEVERAL VISUAL OBJECTS INCLUDING TWO TYPES OF TRAFFIC LIGHTS, TWO TYPES OF TRAFFIC SIGNS, AND PATTERN OBJECTS
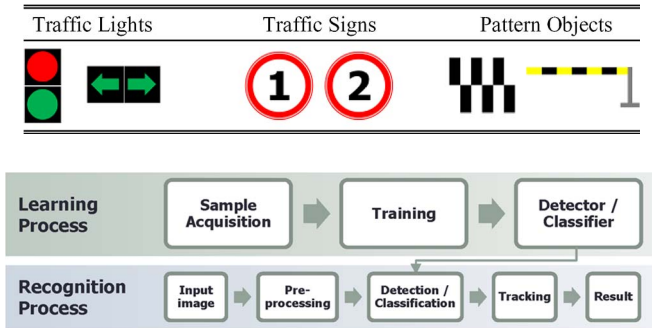

Fig. 4. Machine-learning-based scheme for object recognition.

Fig. 5. FSM for behavior reasoning.

[24] tracking algorithm is implemented using raw data. Tracked dynamic objects are integrated using a covariance-based track-to-track fusion algorithm to reduce the conflict detection from the six laser scanners [25].

There are several visual objects that should be detected and classified by the vision system, i.e., two types of traffic signs, two types of traffic lights, and the pattern objects, as described in Table II. A machine-learning-based scheme is employed to perceive the visual objects. The scheme consists of two main parts (see Fig. 4).

We construct large training samples for each object at first in the learning process. The diversity of the training samples should be satisfied to establish the high performance of the detector or the classifier; therefore, the training samples are obtained from various illuminations, poses, and background conditions. Representative features are selected for each object in the training step; for instance, Haar-like features are used for traffic-sign detection, and a histogram of oriented gradients (HOG) is employed for the traffic-sign classification. Finally, machine learning algorithms are conducted, such as Adaboost and support vector machines (SVMs), to build detectors and classifiers [26], [27].

The input image is preprocessed for noise reduction and feature extraction in the recognition process. Next, the detector finds the location of the object in the image. The classifier identifies what types of objects are detected based on the detected image. Finally, the object tracking method is applied to filter out false positives and integrate temporal recognition results. For instance, the Adaboost-based traffic-sign detector finds where the circular traffic sign is in the image, and then, the circular traffic sign is categorized as the number one, the number two, or others, which are false positives by the SVM classifier with the HOG feature. The traffic sign is tracked by the nearest neighbor filter, which estimates the position, width, and height of the traffic sign in the image [28]. The performance of the traffic-sign recognition is over 95% under various lighting and weather conditions. From over 30 000 sample images, we evaluate that the recognition system performed well as long as the traffic-sign image is not occluded with rain or saturated with direct light.

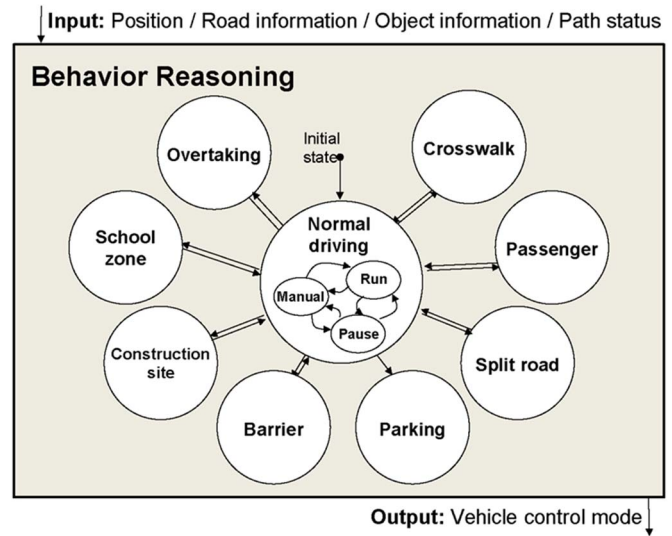*3) Planning:* The planning system determines the maneuvers for autonomous cars. Planning algorithms for autonomous cars can be divided into three stages in order to provide safe and reliable maneuvers under various driving situations [29], [30], i.e., global routing, behavior reasoning, and local motion planning. Global routing finds the fastest and safest way to get from the initial position to the goal position. In this stage, a digital map management system and a data searching algorithm are essential for fast routing [31]. Behavior reasoning assesses the driving situation and determines the overall behavior of the autonomous car based on the global route and perception information (see Fig. 5) [32], [33]. The local motion can be then generated in the local motion planning stage based on the global route and the determined behavior. In the last stage, the generated local motion should avoid static and dynamic obstacle collisions for safe autonomous driving [34]–[36].

The planning system of A1 focused on behavior reasoning and local motion planning since the road map contains the global route of the track provided by the competition organizers. The map data represent the road geometry based on WGS84-type information with centimeter-level accuracy. The objectives of A1's behavior reasoning and local motion planning are to perform autonomous driving and to accomplish various missions in real time. Behavior reasoning executes a rule-based decision process based on finite-state machines (FSMs) for an overall driving strategy based on the localization and perception information (see Fig. 5). Incorporated in the decision process, the rule is predefined to follow traffic regulations (e.g., lane keeping, obeying traffic lights, and keeping under speed limits) and to accomplish various tasks.

In order to drive in various driving environments, local motion planning is composed of two types of path planning algorithms, i.e., road-map-based path planning and free-form path planning (see Fig. 6) [5], [37], [38]. Road-map-based path planning mainly works in normal situations such as lane driving, whereas free-form path planning generates a complex path in unstructured environments such as roads under construction. The selected path planning algorithm with the behavior planner iteratively generates a safe and feasible path.
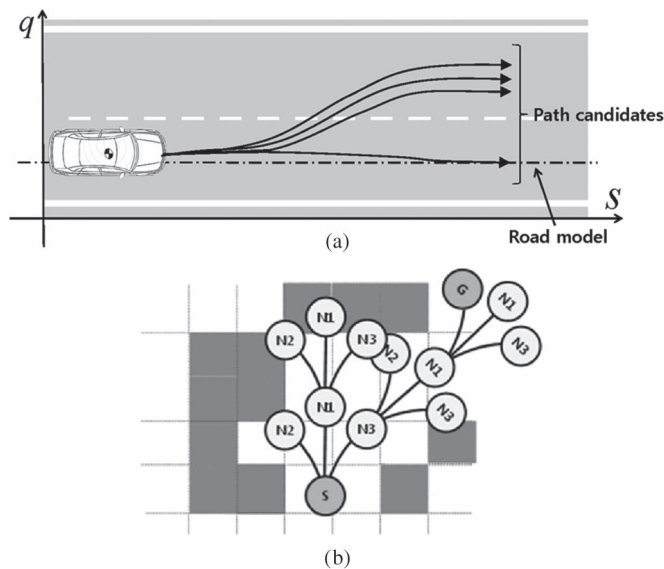
Fig. 6. (a) Lane-keeping and changing-path candidates in the road-map-based path planning algorithm. (b) Graph-structure-based path candidates in the free-form path planning algorithm.

4) Vehicle Control: Vehicle control is an essential function for guiding the autonomous vehicle along the planned trajectory. The vehicle control should be accurate for safe driving and should be robust under various driving conditions [39], [40]. In order to meet these requirements, the control system should be able to deal with several vehicle characteristics, such as nonholonomic constraints [41], vehicle dynamics [42], and physical limitations (e.g., constraints on the steering system and maximum allowable tire forces). In addition, a controller is required to solve the tradeoff problem between the tracking performance and the ride comfort [43].

To deal with these vehicle characteristics in a practical way, the vehicle control system of A1 is divided into a lateral controller and a longitudinal controller (see Fig. 7). The lateral control algorithm assumes that the vehicle moves along the Ackermann steering geometry [42]. Based on this assumption, the target steering angle is obtained from a lateral error of the preview points in the generated path. In order to accurately track the path, the preview points and the feedback gains are scheduled according to the vehicle speed and the path curvature information.

The longitudinal control algorithm derives the target position of the acceleration and brake pedals from reference inputs. In order to cope with various driving situations, the longitudinal controller is composed of three modes, i.e., speed control, distance control, and emergency stop. The speed control mode, which is based on the proportional–integral–derivative-control-law-based feedback controller and the vehicle-powertrain-model-based feedforward controller, derives the target acceleration from the target and current vehicle velocities. Based on the target acceleration, the desired wheel torque is converted to manipulate the acceleration and brake pedals. In the distance control mode, an additional desired velocity calculation function is added for position control. The function generates the desired velocity using a position error between the current and target locations. When the autonomous car meets unexpected situations such as system faults and sudden obstacles, the emergency stop mode produces maximum deceleration for accident avoidance.

5) System Management: For the development and operation of an autonomous car, the system manager is essential for supervising the entire autonomous driving system. Basically, the system manager of A1 performs the following functions: as a human–machine interface (HMI), driving mode management, and fault management. The HMI consists of an operating interface (a remote controller and an e-stop switch) and a display system that indicates the health status of the car, the position, the path, and the surrounding object information. For the operation of A1, the driving mode is divided into three states, i.e., manual, run, and pause. In the manual mode, A1 is manipulated by the human; in the run mode, A1 drives by itself. If the operator pushes the emergency stop switch or a system fault occurs, the system mode is converted to the pause mode. The fault management system monitors the health status of all modules for safe driving. If the health status of the critical module for autonomous driving fails or does not update for a certain period of time, fail management algorithms, which are embedded in each module, determine the state of the system health as failed and convert the autonomous mode to the pause mode.

IV. IMPLEMENTATION OF DISTRIBUTED SYSTEM ARCHITECTURE OF A1 BASED ON SOFTWARE DEVELOPMENT PROCESS AND PLATFORM

The autonomous driving algorithm of A1 has been developed based on the proposed system development process and platform. The autonomous driving algorithms of A1 are first organized as software components. The software components are then mapped to the proper computing units. Finally, the implementations of the software components are performed on the computing units based on the common software platform and in-vehicle network.

A. Software Component Design

Fig. 8 describes the designed software components and information flow of autonomous car A1. The A1 software architecture is composed of four parts, i.e., a sensor interface, autonomous driving algorithms, an actuator interface, and a development interface. Through the sensor interface, the various types of sensor data are entered into the autonomous driving algorithms. The GPS provides the global position, speed, and heading angle data for the positioning algorithm. The IMU and onboard sensors provide the dynamic information of the ego vehicle to the estimation algorithm of the vehicle motion. The cameras and laser scanners measure the information about the external environment around the ego vehicle.

Based on the sensor information, the autonomous driving algorithms generate the control inputs of the vehicle actuators to drive the vehicle autonomously. The vehicle state estimation algorithm estimates the vehicle's dynamic states by integrating the motion sensor information with the vehicle system models. The estimates of the vehicle state are incorporated into the GPS
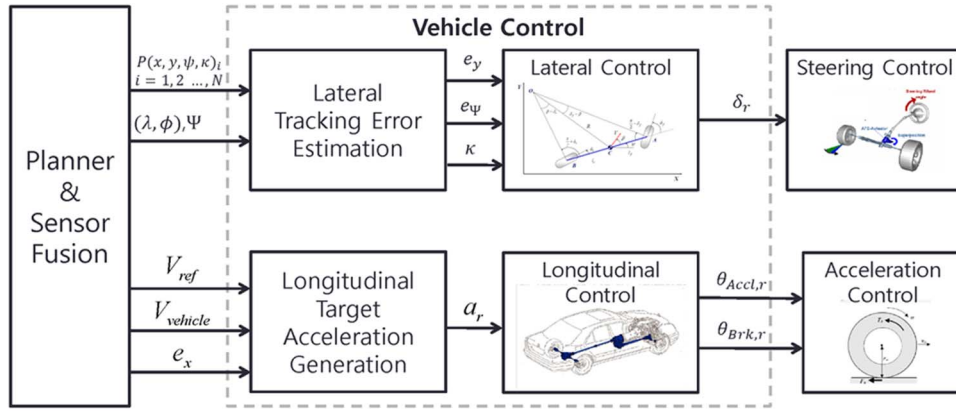
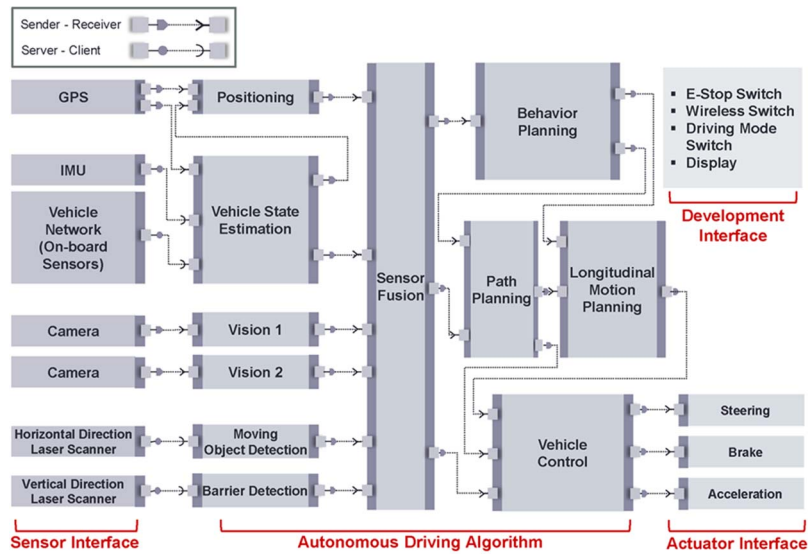Fig. 7.   Structure of the vehicle control system.

Fig. 8.   Software architecture of autonomous car A1.

data to more accurately and reliably estimate the ego vehicle's position. The vision algorithms detect and classify the objects in the driving environment based on the image information from the color cameras and monocameras. The road barriers and the static and dynamic obstacles are detected and tracked using the detection and tracking algorithms. The sensor fusion algorithm integrates all of the information from the perception algorithms and improves the accuracy and integrity of the primitive percep- tion data. The planning algorithm uses the integrated perception data to determine the behavior and motion of the vehicle. The vehicle control algorithm calculates the control inputs from the vehicle actuator, such as steering, braking, and acceleration, to follow the desired behavior and motion from the planning algorithm.

The actuator interface conveys the actuator control input that is generated from the autonomous driving algorithm to the low-level controllers for steering, braking, and accelerating. The development interface provides the safety and debugging functions for developing the autonomous system, including the emergency stop, the wireless stop, the driving mode selection, and monitoring.

The information flows between each software component are described with a VFB, as shown in Fig. 8. There are two types of communication between each software component, i.e., client–server and sender–receiver. At the client–server interface, the server provides the information to clients when the clients request the service. At the sender–receiver interface, the sender provides the information to the receiver without any request. Since the information flows are abstracted with the VFB, the software component designer does not need to be concerned with the constraints of the computing hardware and network. The designer can only concentrate on the design of the functional aspects of the software components.

B. Computing Unit Mapping

The layout of the distributed computing units and the results of the software component mapping are represented in Fig. 9. The software components of the autonomous driving algo- rithm and the system interfaces are distributed into the various types of computing units. The computing units of A1 consist of two embedded industrial computers, i.e., a rapid controller
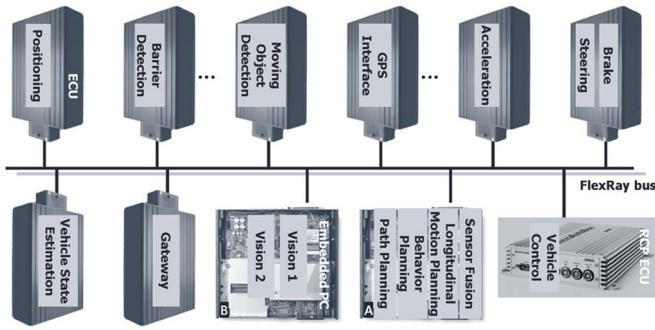
Fig. 9. Mapping of the autonomous driving algorithms and the computing units.

Fig. 10. Example of the implementation of the software components using the common software platform.

TABLE III
EXECUTION TIME OF SOFTWARE ON INDUSTRIAL COMPUTER

Software Components	Execution Time (Full-resource used)	Assigned Embedded PC
Sensor fusion	10-15 ms	A
Longitudinal motion Planning	10-20 ms	A
Behavior Planning	5-15 ms	A
Path Planning	25-50 ms	A
Vision 1	15-25 ms	B
Vision 2	15-25 ms	B

prototyping (RCP) electronic computing unit (ECU) and 13 ECUs that are based on 32-bit microcontroller units (MCUs).

The mapping of the software components into computing units is performed by an experienced system designer. The designer considers several of the constraints for mapping the software components.

The sensor fusion, planning, and vision software components are mapped into two embedded industrial computers (Intel Core 2 Duo 1.50 GHz with 2-GB random access memory using Windows 7) for several reasons. First, the software components require high computing power to run; consequently, the software components are mapped onto a high-performance computer system. In addition, all of the software components cannot be assigned into one computing unit; therefore, the software components are divided into two groups in consideration of the algorithm execution time and dependence (see Table III). The second reason is the hardware and software dependence of the component systems. Many commercial sensors and monitoring devices support general-purpose OS (GPOS) environments such as Windows and Linux. Most software development kits of sensors (such as device drivers and libraries) are supplied as the GPOS version. Furthermore, some specific libraries such as OpenCV and BOOST, which can allow to accelerate the implementation of vision or planning applications, are easy to use in the GPOS environments [44], [45].

In addition, the vehicle control software component is mapped in an RCP-ECU. The RCP-ECU has high real-time performance for time-critical systems such as a vehicle control algorithm and provides development flexibility obtained from a well-organized user interface between a model-based control algorithm and implementation. Finally, the steering and brake software components are placed in one ECU because the components should simultaneously operate longitudinal and lateral control actuations with real-time constraints. The ECU is
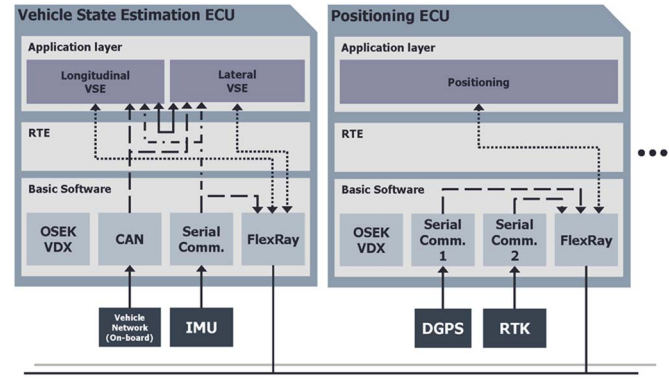
installed next to the actuators to reduce the wire length between the ECU and the actuators as well as to obtain a fast response. Other software components are mapped in a similar way.

C. Implementation of Functions

The software components assigned to the computing units are implemented using the proposed software platform and in-vehicle network. An example of the implementation of the software components for the vehicle state estimation and positioning is described in Fig. 10. Two ECUs are used for the implementation of the example software components, and each ECU is connected by FlexRay networks.

The software component of each function can be divided into several functional subcomponents. The vehicle state estimation component of the example is divided into longitudinal and lateral vehicle state estimations. The functional components are independent of the computing hardware and network because there is a runtime environment (RTE). The RTE layer provides the standard interface between the application layers and the basic software layer; therefore, the dependence between the software components and the hardware can be minimized. The RTE also provides the scheduling interface between the application software component and the OS of the computing unit. The basic software layer provides the basic software components that are highly dependent on the computing unit and networks. In the example, the OSEK/VDX real-time OS (RTOS) is used for the scheduler, and serial communication is used for the sensor networks for the IMU, the DGPS, and the RTK-GPS. A CAN is used for accessing the vehicle network to obtain the onboard sensor information. FlexRay, which is the backbone network of A1, is used to share the processing results with the other computing units.

As proposed in Part I, the FlexRay network design process has two parts, i.e., one part is a network parameter configuration, and the other is a network message scheduling.

In the network parameter configuration step, the FlexRay network has more than 70 configuration parameters in order to operate properly, including cycle configuration parameters, start-up- and wake-related parameters, and clock-correction-related parameters. These parameters should be appropriately configured in order to take advantage of the time-triggered
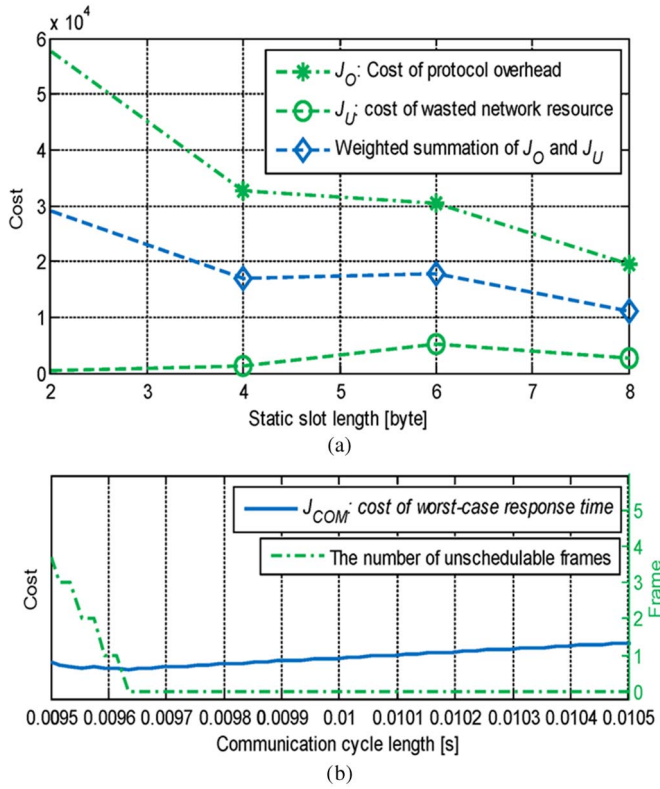
Fig. 11. Optimization results of the FlexRay parameters.

protocol. In the early stage of the autonomous car development, the network parameter is heuristically configured to allow for the modification of the network design since a change in the system configuration frequently occurs. We can determine which data should be transferred through the FlexRay network after the system configuration is fixed. The static slot length and the communication cycle length can be obtained using the FlexRay network parameter optimization method based on the determined network data [46]. This method performs the optimization of the static slot length and the communication cycle length to minimize the cost of the protocol overhead J_O, the cost of the wasted network resources J_U, and the cost of the worst case response time J_{COM}. Fig. 11 shows the optimization costs (J_O, J_U, and J_{COM}) of the FlexRay parameters of A1.

In order to optimize the static slot length in the FlexRay network, the optimization problem is formulated using J_O and J_U as follows:

$$\min \quad J_{STslot} = J_O + J_U$$
$$= (C_0 + x \cdot BSS) \cdot \sum_{k=1}^{n} \left\lceil \frac{msgST_k}{x} \right\rceil$$
$$+ \sum_{k=1}^{n} \left(x \cdot \left\lceil \frac{msgST_k}{x} \right\rceil - msgST_k \right) \quad (1)$$

$$\text{subject to} \quad X = \left\{ x \mid x = 2 \cdot i, \right.$$
$$\left. \text{for } i = 1, 2, \ldots, \max \left(\left\lceil \frac{msgST_k}{2} \right\rceil \right) \right\},$$
$$k = 1, 2, \ldots, n \quad (2)$$

TABLE IV
WORST CASE EXECUTION TIME OF SAFETY-RELATED APPLICATIONS

Application software	Worst-case execution time
On-board sensor	127 usec
GPS interfaces	2.38 msec
Vehicle state estimation	1.02 msec
Positioning	1.48 msec
Vehicle control	3.14 msec
Brake and steering	2.51 msec
Acceleration	3.16 msec

where $msgST_k$ is the byte length of each static (ST) message, C_0 is the sum of the protocol overhead length, including the header, the trailer, the transmission start sequence, the frame start sequence, the frame end sequence, the idle delimiter, the action-point offset, and the safety margin. The byte start sequence (BSS) is the additional coding element for each byte of the frame. As described in the aforementioned optimization problem, the cost for the protocol overhead is closely related to the total number of ST frames. If we choose a long ST slot length x, the ST messages are divided into a smaller number of ST frames; thus, we can reduce the protocol overhead cost. However, if ST slot length x is long, a large amount of unused network resource will result. Therefore, the protocol overhead and the unused network resources must be considered concurrently. In Fig. 11(a), the two dotted green lines represent the costs of the protocol overheads J_O and the wasted resources J_U; in addition, the dotted blue line describes the weighted sum of J_O and J_U. From the results, the static slot length is selected as 8 B, which is the minimal point of the cost.

The communication cycle length is optimized to minimize the worst case response time cost J_{COM} of the static and dynamic frames while satisfying the schedulability of all messages. The optimization problem is denoted as follows:

$$\min \quad J_{COM} = \sum_{k=1}^{n} R_{ST_i} \quad (3)$$
$$= \sum_{i=1}^{n} (C_{com.cycle} + C_i)$$
$$= \sum_{i=1}^{n} \{ (C_{ST} + C_{DYN} + C_{NIT}) + C_i \}$$
$$= \sum_{k=1}^{n} \{ (ST_{slot} \cdot n_s + d \cdot MS + 200 \cdot MT) + C_i \}$$

$$\text{subject to} \quad n_{ST_{fail}} = 0 \quad (4)$$

where R_{ST_i} is the worst case response time of ST frame i. C_{ST}, C_{DYN}, and C_{NIT} are the durations of the ST segment, the dynamic (DYN) segment, and network idle time (NIT), respectively, and C_i is the communication time of frame i. $n_{ST_{fail}}$ is the number of unschedulable ST frames. In cost function J_{COM}, R_{ST_i} is proportional to the length of the communication cycle; thus, C_{ST} is determined by the duration of ST slot ST_{slot} and the number of ST slots n_s, and C_{DYN} is defined by the number of minislots d and the number of bits in a
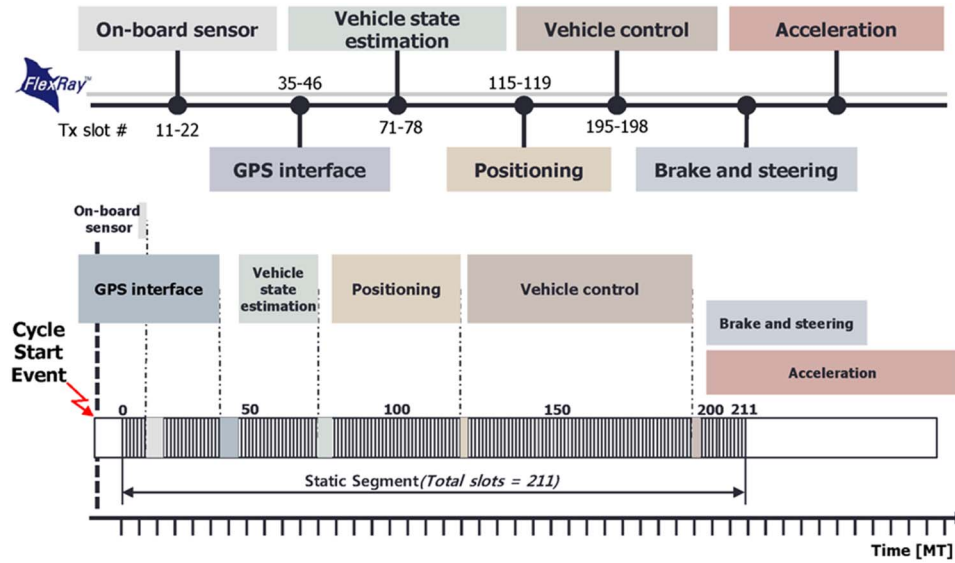
Fig. 12. Synchronization of the safety applications of A1.

minislot (MS). C_{NIT} is assumed as 200 macroticks (MTs). Since the communication cycle is determined by the summation of C_{ST}, C_{DYN}, and C_{NIT}, the optimal communication cycle is determined as the minimum communication cycle within; unschedulable ST frames do not exist. Fig. 11(b) shows the cost of the worst case response time and the number of unschedulable frames. The results show that a communication cycle length over 9.634 ms is acceptable; consequently, we selected 10 ms as the communication cycle length in the FlexRay network of A1.

Network messages are scheduled to synchronize the application software with the FlexRay network after the network parameters are optimized. The synchronization of the network messages can minimize network delays and make the delay predictable. This is a powerful advantage, particularly for the safety-related applications of autonomous cars because the unexpected behavior caused by an unpredictable long network delay can be eliminated. The worst case execution time of the application software should be measured in order to synchronize the messages. The worst case execution time can be found from the execution time measurements of several test cases. The test cases are generated by a code coverage analysis tool provided by MATLAB and Simulink. The measured worst case response time is represented in Table IV. The measured worst case execution time allows the network messages to be assigned to each static slot for the synchronization of the application software.

Fig. 12 describes the synchronization results of the network message for the safety-related applications of A1 (see Table IV). The vehicle state estimation ECU starts computing after receiving the sensor data that are from the onboard sensor and GPS. The FlexRay network is updated with the results after the calculations for the vehicle state estimation are completed. The positioning algorithm then uses the vehicle state estimation results for the position estimation and updates the FlexRay network with the position estimates. In the same manner, the vehicle control operates after receiving the positioning information and generates control inputs for accelerating, braking, and

steering. Finally, the acceleration, braking, and steering ECUs receive the control input and control each actuator.

Ethernet and CANs, which are event-triggered protocols, are also used for the communication system as the local sensor networks of A1. The point-to-point network topology between a sensor and a computer is applied for the event-triggered network to minimize the effect of jitter in the event-triggered protocol.

D. Implementation Results

1) Development Process and Software Platform: The proposed software development process and platform have various advantages for the development of autonomous car A1. The distributed system architecture could reduce the computational complexity of A1. When the development of A1 began, we used a centralized system architecture. Since the initial requirement was waypoint tracking in an obstacle-free region, there was no problem with the computational resource. However, as requirements increased, such as moving obstacle tracking, traffic signal detection, and so on, the centralized system could not schedule all of the functions. In order to resolve this problem, we considered a more high-performance computational unit, but it could not resolve the computational complexity. In addition, it required additional costs and power sources. Therefore, a distributed system architecture was applied for decentralizing the computational load. Consequently, we reduced the computational complexity through decentralized computation, as well as saved costs and power sources.

The fault management system was applied to A1 in order to operate the autonomous car safely. Every computing unit in A1 basically generates a life signal for a health status check. Based on the signal, the fault management algorithm diagnoses the status and will back up the failed unit. By applying the fault management algorithm, the distributed autonomous driving system of A1 prevented some accidents due to negligence or software error.
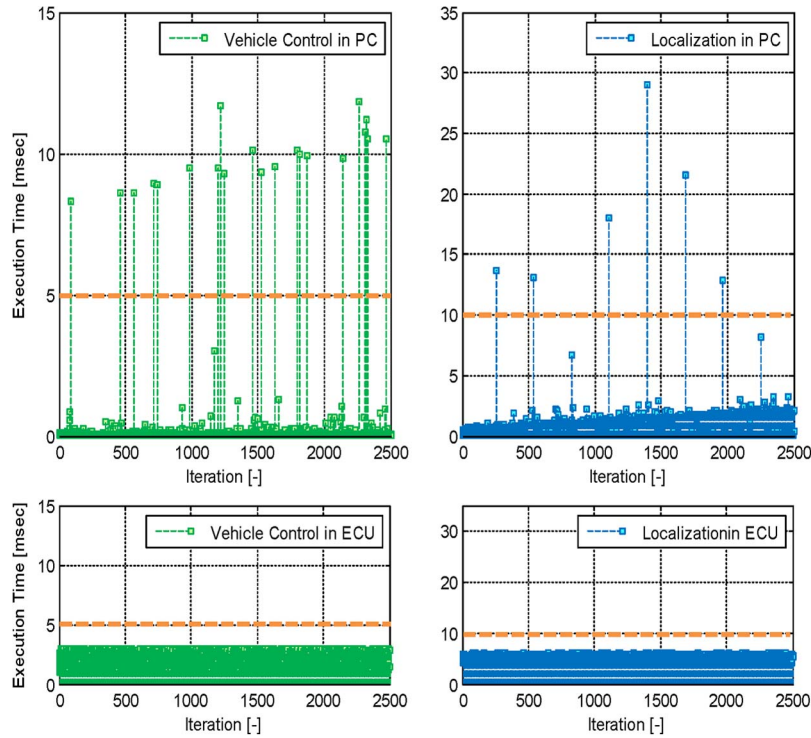
Fig. 13. Execution time of the vehicle control and localization algorithms. (Upper) Centralized system architecture's cases with a single PC. (Lower) Distributed system architecture's cases with multiple ECUs.

The distributed system architecture of A1 was able to reduce the noise and the cost from the wiring. Initially, the acceleration pedal position sensor in A1 was too far from the vehicle controller; hence, the wire between the sensor and the controller was too long. The long wire length created noise in the measurement of analog signals and caused the degradation of the vehicle control performance. In order to resolve these problems, a measurement function for the acceleration pedal position sensor was transferred to another ECU located near the sensor, and the length of the wires was greatly reduced. As a result, not only was the noise problem resolved, but the stability of the controller was also increased. In addition, this reconfiguration work was not difficult due to the transferability of the proposed software architecture.

The proposed distributed architecture could support the development and testing being carried out simultaneously. In the development of A1, many developers were involved. Depending on the role of each development part, each individual person developed and tested his/her own software. In order to improve the development efficiency, the individual software could be independently developed and tested by considering the functional and temporal characteristics.

The proposed development process and system platform enabled flexible system changes and the extension of the system structure of A1. For instance, when the brake controller of A1 was developed, there were frequent hardware changes due to performance limitations. However, since the system development process is not dependent on the hardware, the development of the brake controller software was not influenced by the hardware change. As another example, there were several changes to the sensor layout in A1. When two laser scanners

TABLE V
COMPARISON BETWEEN FLEXRAY AND CANS OF A1

	FlexRay	CAN
Baud Rate	10 Mbps	500 kbps
Network Nodes	16	16
Messages	156	154
Bus Loads	73.93 % (Fixed bus load)	81.23% (100% peak bus load)
Others	Jitter free	Unpredictable latency, Jitter

were added to detect rear objects, additional software was not required to be developed. The reason for this is that the existing software can be reused and modified easily.

With respect to the real-time performance, the proposed distributed system architecture exhibits more reliable computational processing than the centralized system architecture. Fig. 13 depicts the execution time of the localization and vehicle control algorithms in each system architecture. Complete execution within an accurate period is an important factor for reliable autonomous driving in real time since the localization and vehicle control algorithms are safety-critical functions for autonomous driving systems. In Fig. 13, the centralized platform generally has excellent computational performance in both the localization and vehicle control operations. However, there exist several peak execution times that do not satisfy the execution time constraints (see Fig. 13). The reason for this is that the unexpected and sudden increase in the computation load for the other task affects the operation of the localization and control tasks.

The operation of the localization and the vehicle control in the distributed system consumes more execution time than
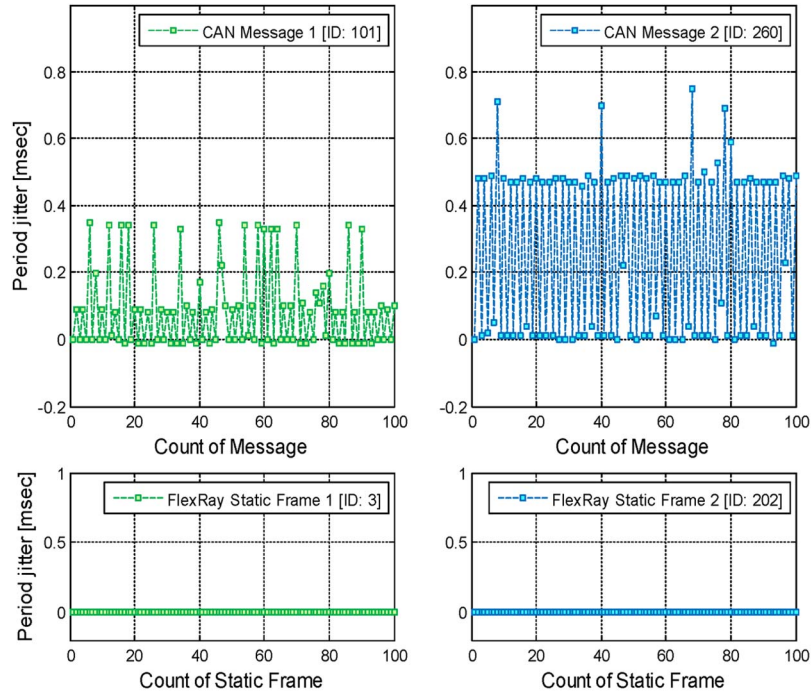
Fig. 14. Period jitters in the FlexRay and CANs of A1.

the centralized case due to the computation power limitations of the implemented ECUs. However, the tasks of the localization and the vehicle control in the decentralized real-time embedded system are schedulable within the execution time constraint since the distributed ECUs relatively run few tasks and operate with the OSEK-OS that is a standard RTOS for automotive embedded software [47], [48] (see Fig. 13). The distributed system architecture is more suitable for hard real-time applications.

2) FlexRay Network: There are several advantages to using the FlexRay-network-based distributed system. A high bandwidth is one of most obvious advantages. The high bandwidth of the FlexRay network allows for sharing a large amount of information, as compared with the CAN, which is the most widely used for in-vehicle networks in the automotive industry. For example, the FlexRay network is able to send 211 messages every 10 ms. However, the number of messages that can be transmitted by the CAN is limited to 74 every 10 ms. Since a total of 156 messages for A1 are used in the final network configuration, a periodic message transmission is possible without delay. On the other hand, if the CAN is used as the backbone network, a transmission delay of a message will occur, and the overall system performance can be reduced, as shown in Table V.

The time-triggered scheme of the FlexRay network ensures exact periods by using the global time. In addition, since the time of the transmission and reception of network messages can be predicted, the application software can be synchronized with the global time. Otherwise, the event-triggered scheme of the CAN cannot guarantee the periodic transmission of a message. As shown in Fig. 14, the static frames in the FlexRay network are transmitted without delay, whereas CAN messages are transmitted with period jitter. In addition, the period jitters
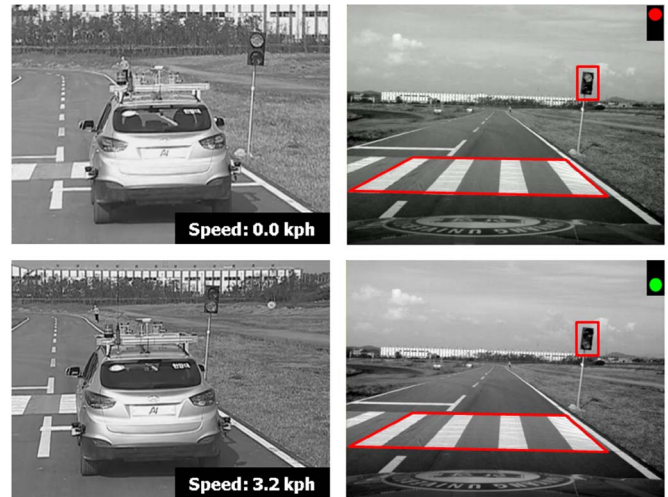

Fig. 15. Results of the crosswalk/traffic-light detection missions.

of the CAN differ according to the priority of a message. Therefore, when compared with the CAN, the time synchronized application software of FlexRay can be expected to have better performance.

V. Driving Results of A1 in 2012 AVC

In order to evaluate the validity of the proposed system architecture and development process, A1 drove itself on the racetrack and performed the missions in the 2012 AVC, as shown in Fig. 1 and Table I. The pictures in Figs. 15–23 represent the mission results of A1. Each picture is a snapshot from the driving videos, which are taken from inside and outside of A1. As shown in these results, A1 not only drove the entire

Fig. 16.   Results of the split-road mission.

Fig. 20.   Results of the construction-site mission.
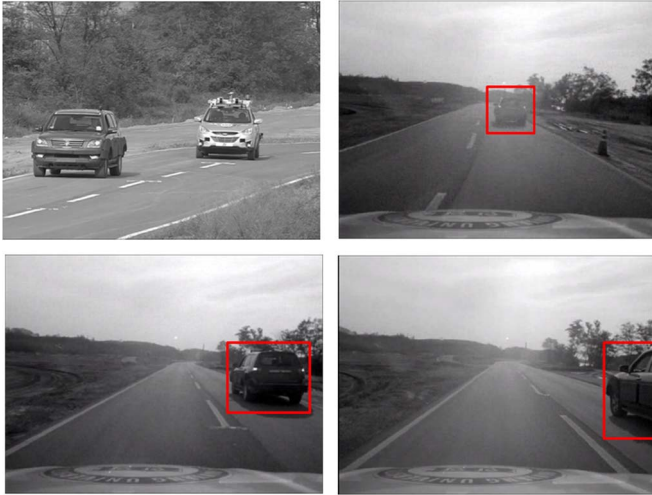

Fig. 17.   Results of the overtaking mission.
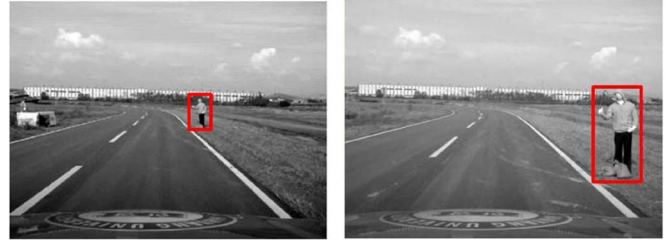

Fig. 21.   Results of the passenger pickup mission.

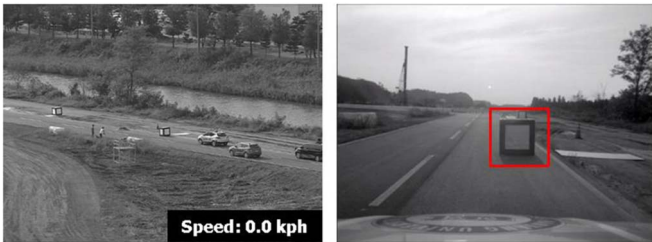Fig. 22.   Results of the complex obstacle mission.

Fig. 18.   Results of the school zone and sudden obstacle missions.

Fig. 23.   Results of the parking mission.

Fig. 19.   Results of the barrier mission.

VI. Conclusion

This paper (Part II) has described the implementation of the autonomous car that follows the development methodology proposed in Part I.

In Part I, we addressed several advantages of the methodology from the following perspectives. First, the distributed system architecture can reduce the computational complexity of the entire system, guarantee fault-tolerant characteristics, and enhance the modularity of the system. Second, the development process provides comprehensive instructions for designing and integrating the distributed systems of an autonomous car. Finally, the layered architecture-based software platform (that originated from AUTOSAR) can improve the reusability,

racetrack safely but also completed the missions perfectly. The maximum speed of A1 is up to 80 km/h during the final race. A1 won the championship of the 2012 AVC with a final time of 6 min and 52 s, with all missions completed. The final driving video is available at http://youtu.be/dzE2WReJeSM.

scalability, transferability, and maintainability of the application software.

In this paper, we have presented the case study of the development methodology by implementing the autonomous car. First, the five essential technologies of autonomous driving (localization, perception, planning, vehicle control, and system management) have been briefly explained. Next, the three steps (the software components design, computing unit mapping, and the implementation of functions) of the proposed development process have been executed according to the instructions of the development methodology. In the procedure, several advantages of the distributed system are emphasized, comparing the centralized system, i.e., the computational load distribution, the enhancement of system safety by cross-checking the life signal, noise, and cost reduction from the wiring, and the achievement of a flexible system against changes or extensions. The accomplishments of the autonomous car A1 that won the 2012 AVC in Korea prove the validity of the proposed development methodology.

However, the case study was only performed based on the specific computing platform, such as Windows and OSEK/VDX. The authors plan to extend the development process to adapt more software platforms such as Linux and the Robot OS for developing autonomous driving systems [49].

REFERENCES

[1] S. Thrun *et al.*, "Stanley: The robot that won the DARPA Grand Challenge," *J. Field Robot.*, vol. 23, no. 9, pp. 661–692, Sep. 2006.

[2] C. Urmson *et al.*, "A robust approach to high-speed navigation for un-rehearsed desert terrain," *J. Field Robot.*, vol. 23, no. 8, pp. 467–508, Aug. 2006.

[3] M. Montemerlo *et al.*, "Junior: The Stanford entry in the urban challenge," *J. Field Robot.*, vol. 25, no. 9, pp. 569–597, Sep. 2008.

[4] C. Urmson *et al.*, "Autonomous driving in urban environments: Boss and the urban challenge," *J. Field Robot.*, vol. 25, no. 8, pp. 425–466, Aug. 2008.

[5] K. Chu, M. Lee, and M. Sunwoo, "Local path planning for off-road autonomous driving with avoidance of static obstacles," *IEEE Trans. Intel. Transport. Syst.*, vol. 13, no. 4, pp. 1599–1616, Dec. 2012.

[6] J. Han, D. Kim, M. Lee, and M. Sunwoo, "Enhanced road boundary and obstacle detection using a downward-looking LIDAR sensor," *IEEE Trans. Veh. Technol.*, vol. 61, no. 3, pp. 971–985, Mar. 2012.

[7] K. Jo *et al.*, "Overall reviews of autonomous vehicle A1—System architecture and algorithms," presented at the 8th IFAC IAV, Gold Coast, Qld., Australia, 2013.

[8] J. Kim, K. Jo, D. Kim, K. Chu, and M. Sunwoo, "Behavior and path planning algorithm of autonomous vehicle A1 in structured environments," presented at the 8th IFAC IAV, Gold Coast, Qld., Australia, 2013.

[9] K. Jo, D. Kim, J. Kim, C. Jang, and M. Sunwoo, "Development of autonomous car—Part I: Distributed system architecture and development process," *IEEE Trans. Ind. Electron.*, vol. 61, no. 12, pp. 7131–7140, Dec. 2014.

[10] H. Heinecke *et al.*, "AUTomotive open system ARchitecture—An industry-wide initiative to manage the complexity of emerging automotive E/E-architectures," presented at the SAE Tech. Paper, 2004.

[11] W. Ruyi, L. Hong, Y. Min, W. Jinbo, and Y. Yuhao, "A hierarchical modeling method for AUTOSAR software components," in *Proc. IEEE Int. Conf. ICCET*, 2010, pp. V4-184–V4-188.

[12] D. Kum, G.-M. Park, S. Lee, and W. Jung, "AUTOSAR migration from existing automotive software," in *Proc. IEEE Int. Conf. Ctrl., Autom. Syst.*, 2008, pp. 558–562.

[13] S. Bunzel, "AUTOSAR—The standardized software architecture," *Informatik-Spektrum*, vol. 34, pp. 79–83, 2011.

[14] "AUTOSAR Documents (V4.1)," AUTOSAR Consortium, 2013. [Online]. Available: http://www.autosar.org/

[15] I. Skog and P. Handel, "In-car positioning and navigation technologies: A survey," *IEEE Trans. Intel. Transp. Syst.*, vol. 10, no. 1, pp. 4–21, Mar. 2009.

[16] K. Jo, K. Chu, and M. Sunwoo, "Interacting multiple model filter-based sensor fusion of GPS with in-vehicle sensors for real-time vehicle positioning," *IEEE Trans. Intel. Transp. Syst.*, vol. 13, no. 1, pp. 329–343, Mar. 2012.

[17] I. P. Alonso *et al.*, "Accurate global localization using visual odometry and digital maps on urban environments," *IEEE Trans. Intel. Transp. Syst.*, vol. 13, no. 4, pp. 1535–1545, Dec. 2012.

[18] J. Kichun, C. Keonyup, and S. Myoungho, "GPS-bias correction for precise localization of autonomous vehicles," in *Proc. IEEE IV*, 2013, pp. 636–641.

[19] M. Gwak, K. Jo, and M. Sunwoo, "Neural-network multiple models filter (NMM)-based position estimation system for autonomous vehicles," *Int. J. Auto. Tech.*, vol. 14, no. 2, pp. 265–274, Apr. 2013.

[20] K. Yeonsik, R. Chiwon, S. Seung-Beum, and S. Bongsob, "A Lidar-based decision-making method for road boundary detection using multiple Kalman filters," *IEEE Trans. Ind. Electron.*, vol. 59, no. 11, pp. 4360–4368, Nov. 2012.

[21] Z. Huijing *et al.*, "Detection and tracking of moving objects at intersections using a network of laser scanners," *IEEE Trans. Intell. Transp. Syst.*, vol. 13, no. 2, pp. 655–670, Jun. 2012.

[22] L. Feng, D. Xiangxu, B. M. Chen, L. Kai-Yew, and T. H. Lee, "A robust real-time embedded vision system on an unmanned rotorcraft for ground target following," *IEEE Trans. Ind. Electron.*, vol. 59, no. 2, pp. 1038–1049, Feb. 2012.

[23] H. Yonghui, Z. Wei, and W. Long, "Vision-based target tracking and collision avoidance for two autonomous robotic fish," *IEEE Trans. Ind. Electron.*, vol. 56, no. 5, pp. 1401–1410, May 2009.

[24] D. Musicki, R. Evans, and S. Stankovic, "Integrated probabilistic data association," *IEEE Trans. Automat. Ctrl.*, vol. 39, no. 6, pp. 1237–1241, Jun. 1994.

[25] A. Houenou, P. Bonnifait, V. Cherfaoui, and J.-F. Boissou, "A track-to-track association method for automotive perception systems," in *Proc. IEEE IV*, 2012, pp. 704–710.

[26] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Proc. IEEE Comput. Soc. Conf. CVPR*, San Diego, CA, USA, 2005, pp. 886–893.

[27] P. Viola and M. J. Jones, "Robust real-time face detection," *Int. J. Comput. Vis.*, vol. 57, no. 2, pp. 137–154, May 2004.

[28] X. Rong Li and Y. Bar-Shalom, "Tracking in clutter with nearest neighbor filters: Analysis and performance," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 32, no. 3, pp. 995–1010, Jul. 1996.

[29] L. Li and F. Y. Wang, *Advanced Motion Control and Sensing for Intelligent Vehicles*. Berlin, Germany: Springer-Verlag, 2007.

[30] R. Kala and K. Warwick, "Motion planning of autonomous vehicles in a non-autonomous vehicle environment without speed lanes," *Eng. App. A.I*, vol. 26, no. 5/6, pp. 1588–1601, May/Jun. 2013.

[31] T. Ching-Chih, H. Hsu-Chih, and C. Cheng-Kai, "Parallel elite genetic algorithm and its application to global path planning for autonomous robot navigation," *IEEE Trans. Ind. Electron.*, vol. 58, no. 10, pp. 4813–4821, Oct. 2011.

[32] T. Bucher *et al.*, "Image processing and behavior planning for intelligent vehicles," *IEEE Trans. Ind. Electron.*, vol. 50, pp. 62–75, 2003.

[33] C. Baker and J. Dolan, "Street smarts for boss," *IEEE Robot. Autom. Mag.*, vol. 16, no. 1, pp. 78–87, Mar. 2009.

[34] N. Sudha and A. R. Mohan, "Hardware-efficient image-based robotic path planning in a dynamic environment and its FPGA implementation," *IEEE Trans. Ind. Electron.*, vol. 58, no. 5, pp. 1907–1920, May 2011.

[35] Y. Kuwata *et al.*, "Real-time motion planning with applications to autonomous urban driving," *IEEE Trans. Ctrl. Sys. Technol.*, vol. 17, no. 5, pp. 1105–1118, Sep. 2009.

[36] S. Glaser *et al.*, "Maneuver-based trajectory planning for highly autonomous vehicles on real road with traffic and driver interaction," *IEEE Trans. Intel. Transp. Syst.*, vol. 11, no. 3, pp. 589–606, Sep. 2010.

[37] K. Chu, J. Kim, and M. Sunwoo, "Distributed system architecture of autonomous vehicles and real-time path planning based on the curvilinear coordinate system," SAE int., Warrendale, PA, USA, Tech. Papers, 2012.

[38] J. Kim, K. Jo, K. Chu, and M. Sunwoo, "Road-model-based and graph-structure-based hierarchical path-planning approach for autonomous vehicles," in *Proc. Institut. Mech. Eng., Part D: J. Automobile Eng.*, Feb. 21, 2014, pp. 909–928.

[39] V. Milanes, J. Villagra, J. Perez, and C. Gonzalez, "Low-speed longitudinal controllers for mass-produced cars: A comparative study," *IEEE Trans. Ind. Electron.*, vol. 59, no. 1, pp. 620–628, Jan. 2012.

[40] C. Lin, A. B. Rad, and C. Wai-Lok, "An intelligent longitudinal controller for application in semiautonomous vehicles," *IEEE Trans. Ind. Electron.*, vol. 57, no. 4, pp. 1487–1497, Apr. 2010.

[41] G. O. A. De Luca and C. Samson, "Feedback control of a nonholonomic car-like robot," *Robot Motion Planning Control*, vol. 229, pp. 171–249, 1998.

[42] R. Rajamani, *Vehicle Dynamics and Control*. New York, NY, USA: Springer-Verlag, 2012.

[43] A. Hemami and M. Mehrabi, "On the steering control of automated vehicles," in *Proc. IEEE Int. Conf. ITSC*, 1997, pp. 266–271.

[44] G. Bradski, "The OpenCV Library," Dr. Dobb's J. SW Tools, 2000.

[45] J. Siek, L. Q. Lee, and A. Lumsdaine, *The Boost Graph Library: User Guide and Reference Manual*. Reading, MA, USA: Addison-Wesley, 2002.

[46] I. Park and M. Sunwoo, "FlexRay network parameter optimization method for automotive applications," *IEEE Trans. Ind. Electron.*, vol. 58, no. 4, pp. 1449–1459, Apr. 2011.

[47] O. Group, "OSEK/VDX Operating System Specification," 2009.

[48] A. Zahir and P. Palmieri, "OSEK/VDX-operating systems for automotive applications," in *Proc. Inst. Elect. Eng.—Seminar OSEK/VDX Open Syst. Autom. Netw. (Ref. No. 1998/523)*, 1998, pp. 4/1–418.

[49] C. Ainhauser *et al.*, "Autonomous driving needs ROS (ROS as platform for autonomous driving functions)," ROS Developer Conf., Stuttgart, Germany, 2013.

Kichun Jo (S'10–M'14) received the B.S. degree in mechanical engineering and the Ph.D. degree in automotive engineering from Hanyang University, Seoul, Korea, in 2008 and 2014, respectively.

He is currently with the Automotive Control and Electronics Laboratory (ACE Lab), Department of Automotive Engineering, College of Engineering, Hanyang University. His main fields of interest include autonomous driving systems, information fusion theories, distributed control systems, real-time embedded systems, and in-vehicle networks. His current research activities include the system design and implementation of autonomous cars.

Junsoo Kim (S'11) received the B.S. degree in mechanical engineering from Hanyang University, Seoul, Korea, in 2008. He is currently working toward the Ph.D. degree in the Automotive Control and Electronics Laboratory (ACE Lab), Department of Automotive Engineering, College of Engineering, Hanyang University.

His main fields of interest include vehicle control, decision theories, path planning algorithms, and real-time systems. His current research activities include behavior reasoning and the trajectory planning of autonomous cars.

Dongchul Kim received the B.S. degree in electronics and computer engineering and the M.S degree in automotive engineering from Hanyang University, Seoul, Korea, in 2008 and 2010, respectively. He is currently working toward the Ph.D. degree in the Automotive Control and Electronics Laboratory (ACE Lab), Department of Automotive Engineering, College of Engineering, Hanyang University.

His current research interests include the detection and tracking of moving objects and information fusion. His research activities include the design of in-vehicle networks and real-time embedded software.

Chulhoon Jang (S'13) received the B.S. degree in mechanical engineering from Hanyang University, Seoul, Korea, in 2011. He is currently working toward the Ph.D. degree in the Automotive Control and Electronics Laboratory (ACE Lab), Department of Automotive Engineering, College of Engineering, Hanyang University.

His main fields of interest include autonomous driving systems, image processing, and machine learning. His current research activities include object detection and classification using multiple sensor fusion and situation assessment for urban autonomous driving.

Myoungho Sunwoo (M'81) received the B.S. degree in electrical engineering from Hanyang University, Seoul, Korea, in 1979; the M.S. degree in electrical engineering from the University of Texas at Austin, Austin, TX, USA, in 1983; and the Ph.D. degree in system engineering from Oakland University, Rochester, MI, USA, in 1990.

In 1985, he joined General Motors Research (GMR) Laboratories, Warren, MI, USA, and has worked in the area of automotive electronics and control for 28 years. During his nine-year tenure with GMR, he worked on the design and development of various electronic control systems for powertrains and chassis. Since 1993, he has led research activities as a Professor with the Department of Automotive Engineering, College of Engineering, Hanyang University, where he is also the Director of the Automotive Control and Electronics Laboratory (ACE Lab). His work has focused on automotive electronics and controls (such as the modeling and control of internal combustion engines, the design of automotive distributed real-time control systems, intelligent autonomous vehicles, and automotive education programs).