Development of Autonomous Car—Part I: Distributed System Architecture and Development Process

Article *in* IEEE Transactions on Industrial Electronics · December 2014

DOI: 10.1109/TIE.2014.2321342

5 authors, including:

Kichun Jo
Konkuk University
33 PUBLICATIONS **460** CITATIONS
SEE PROFILE

Chulhoon Jang
Hanyang University
9 PUBLICATIONS **161** CITATIONS
SEE PROFILE

Myoungho Sunwoo
Hanyang University
194 PUBLICATIONS **1,839** CITATIONS
SEE PROFILE

Development of Autonomous Car—Part I: Distributed System Architecture and Development Process

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

***Abstract*—An autonomous car is a self-driving vehicle that has the capability to perceive the surrounding environment and navigate itself without human intervention. For autonomous driving, complex autonomous driving algorithms, including perception, localization, planning, and control, are required with many heterogeneous sensors, actuators, and computers. To manage the complexity of the driving algorithms and the heterogeneity of the system components, this paper applies distributed system architecture to the autonomous driving system, and proposes a development process and a system platform for the distributed system of an autonomous car. The development process provides the guidelines to design and develop the distributed system of an autonomous vehicle. For the heterogeneous computing system of the distributed system, a system platform is presented, which provides a common development environment by minimizing the dependence between the software and the computing hardware. A time-triggered network protocol, FlexRay, is applied as the main network of the software platform to improve the network bandwidth, fault tolerance, and system performance. Part II of this paper will provide the evaluation of the development process and system platform by using an autonomous car, which has the ability to drive in an urban area.**

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

I. Introduction

TODAY, a paradigm in the automotive industry has been shifted from high-performance vehicles to safe and comfortable vehicles. This paradigm shift has accelerated the development of various intelligent vehicle technologies. Ultimately, maximizing safety and comfort can be achieved by autonomous cars. In order to realize autonomous driving, the car has to perceive driving environments and should also perform path planning and control without human intervention.

For the development of these autonomous driving technologies, the Defensive Advanced Research Projects Agency opened the Grand Challenge and Urban Challenge competitions in the U.S. The Grand Challenge competition focused on the development of autonomous cars that can traverse off-road terrain by themselves [1]. Based on the results of the Grand Challenge, the Urban Challenge competition aimed at the advancement of autonomous cars with urban driving technology [2]. Through both of these competitions, the feasibility of the autonomous car realization has been confirmed. As a result, global automakers and information technology companies such as General Motors, Volkswagen, Toyota, and Google have made an enormous investment in the commercialization of autonomous cars.

In Korea, in order to stimulate autonomous car research, two autonomous vehicle competitions (AVCs) were held in 2010 and 2012 by the Hyundai Motor Group. The purpose of the 2010 AVC was to establish the foundation of autonomous driving technology in Korea. The missions of the 2010 AVC concentrated on waypoint tracking and static obstacle avoidance. Based on the fundamental technology, the 2012 AVC tried to develop techniques for urban driving environments; the missions were related to urban driving such as traffic signal detection, overtaking, crosswalk stops, and passenger detection. This paper is based on the results of the 2012 AVC.

Developing an autonomous car refers to the integration of technologies from two industry fields: the automotive industry and the mobile robot industry. The robust and reliable mechanical and electrical platform for the autonomous car can be achieved from the automotive industry. Many autonomous driving algorithms have been researched for a long time in the robot industry, and they can be applied to the autonomous car.

Each industry field has their own environments in which to develop an intelligent vehicle system. In the automotive industry, there are many standard development platforms because the automotive industry consists of manufacturers and suppliers. In order to communicate and cooperate with each other, standard software and hardware platforms are essential. AUTomotive Open System ARchitecture (AUTOSAR) is a standardized software architecture in the automotive industry [3], [4]. AUTOSAR is based on a component-based software design model and provides the design methodology and standard software platform for designing automotive software. OSEK/VDX is a standard real-time operating system (RTOS) for automotive embedded software. LIN, CAN. FlexRay, and MOST are developed for the in-vehicle network of cars [5]. Although standard platforms are useful for developing automotive systems, there are some limitations that apply for developing autonomous cars. Since the automotive industry is very conservative with regard to the safety and robustness of the system, it is not flexible to changes and additions of technologies. However,

Manuscript received November 27, 2013; revised February 22, 2014; accepted April 1, 2014. Date of publication May 1, 2014; date of current version September 12, 2014. This work was supported in part by a National Research Foundation of Korea (NRF) Grant funded by the Korean government (MEST) under Grant 2011-0017495, in part by the Industrial Strategy Technology Development Program of the Ministry of Knowledge Economy (MKE) under Grant 10039673, and in part by the BK21 plus program under Grant 22A20130000045 of the Ministry of Education, Republic of Korea.

The authors are with the Automotive Control and Electronics Laboratory, Department of Automotive Engineering, Hanyang University, Seoul 133-791, Korea (e-mail: jokihaha@hanyang.ac.kr; junskys@gmail.com; gulajima@hanyang.ac.kr; chulhoonjang@gmail.com; msunwoo@hanyang.ac.kr).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIE.2014.2321342

the investigation of automotive car technologies is in progress, and many technical attempts will occur for autonomous driving systems.

In the robot industry, development environments and system platforms were researched and developed for a long time in order to manage the complexity of an autonomous robot system [6]–[8]. The complexity of a robot system originates from the interaction between numerous heterogeneous sensors, computers, and actuators. There are also many other reasons to use the robot development environment and platform such as promoting the integration of new technologies, simplifying system design, abstracting the low-level components, improving software quality, and reusing software. Some of these development environments are freely available and provide excellent functions. Therefore, developers can easily access the robot development environments and platforms. However, since the robot development environments and platforms lack support for automotive technologies such as OSEK/VDX, CAN, and FlexRay, their application toward developing autonomous cars is limited.

This paper proposes a development process and a system platform to develop an autonomous car that has distributed system architecture. The process and the platform are based on the methodology and platform of AUTOSAR; however, in order to improve the flexibility and scalability of the development of an autonomous driving system, unnecessary processes of AUTOSAR are discarded and reformed. In addition, FlexRay, which is the next-generation in-vehicle network in the automotive industry, is applied for the backbone network of the system platform in order to increase network bandwidth, fault tolerance, and system performance. The proposed development process and system platform are evaluated through the autonomous car A1, which participated in the 2010 and 2012 AVCs. A1 successfully completed all missions of the AVCs and won in 2010 and 2012.

This paper is organized as follows. Section II introduces the distributed system architecture for the development of an autonomous car; the development process of the distributed system is described in Section III; the system platform is explained in Section IV. Part II of this paper will present the autonomous driving algorithm of A1 and the algorithm implementation based on the proposed process and platform. The benefits of the process and platform will be discussed using implementation results of the autonomous car A1.

II. Distributed Architecture for Autonomous Cars

A. Basics of Autonomous Cars

An autonomous car is a self-driving car that has the ability to drive by itself without human intervention. There are five basic functions that drive the autonomous car: perception, localization, planning, control, and system management. The conceptual description of each function is shown in Fig. 1. Perception is a process that senses the surrounding environment of the autonomous car using various types of sensor techniques such as RADAR, LIDAR, and computer vision. The localization finds the position of the autonomous car using the techniques of a Global Positioning System, dead reckoning, and roadway maps.

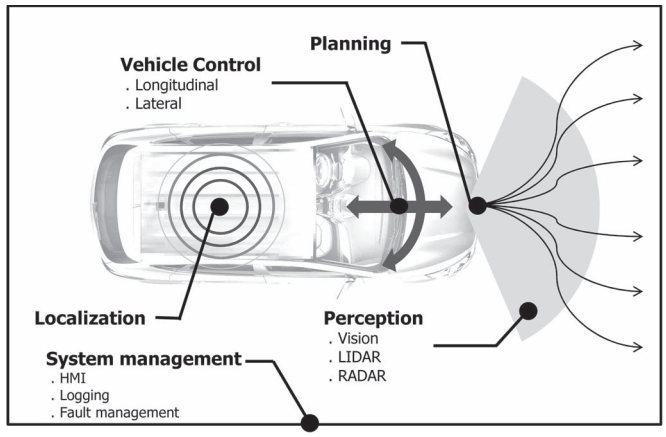

Fig. 1. Basic functions of autonomous cars.

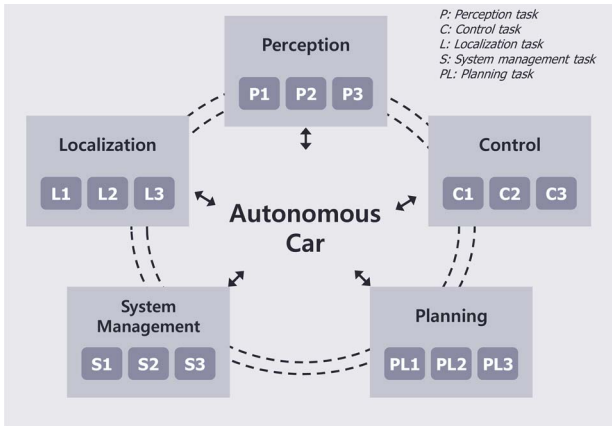

Fig. 2. Functional components of the autonomous driving system.

The planning function determines the behavior and motion of the autonomous car based on the information from perception and localization. The 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. The example functions of the system management are the fault management system, logging system, and human–machine interface (HMI).

Almost all autonomous cars have the five basic functions, and each function has different functional subcomponents according to the purpose and complexity of the autonomous car. The functional subcomponent represents the actual implementation of autonomous driving functions, as shown in Fig. 2. In order to implement the functional components on the computing units of the autonomous car, there are two types of approaches: centralized architecture and distributed architecture [9].

B. Centralized System Architecture

In the centralized system architecture, most functional components of the autonomous driving system are implemented into a single computing unit. All of the sensors and actuators of the autonomous driving system are connected into the single centralized computing unit, as shown in Fig. 3. The centralized system architecture has the benefit of simplifying the system configuration because all of the devices, including the sensors

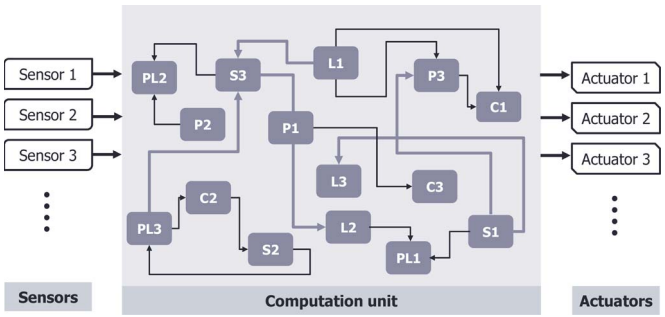

Fig. 3. Centralized system architecture for an autonomous car.

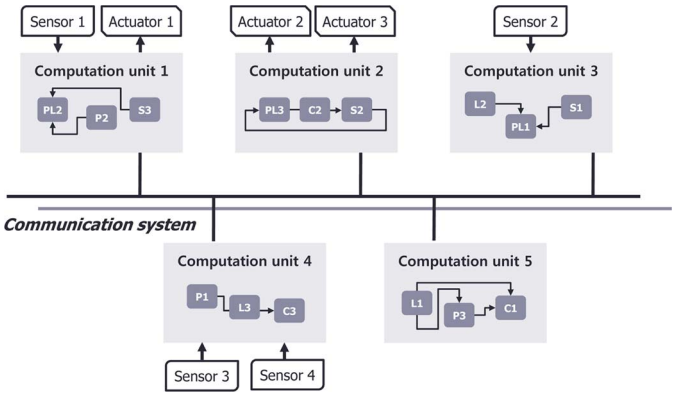

Fig. 4. Distributed system architecture of autonomous cars.

and actuators, are connected to the centralized computing unit. It is also possible to share information without an additional network, and there is minimum delay and information loss due to the communication.

In contrast to the benefits, the centralized system has several weaknesses in many ways. First of all, the centralized system requires a high computational capability because the algorithms of the autonomous driving system may be large and complicated. The centralized architecture may also lack scalability. Even if there are any functions or devices that should be added to the centralized system, a developer should consider various constraints, including computational abilities, hardware capabilities, and installations. Moreover, it is hard to guarantee reliable fault tolerance since building a backup system and handling a malfunction are not easy for the centralized system architecture. Furthermore, since all sensors and actuators are attached to the central computing unit, wire harnesses may become longer, thereby causing unexpected weights and noises. Finally, with regard to the development and test, collaboration is very difficult for the centralized system because the developers cannot access the development and test environment concurrently.

C. Distributed System Architecture

In the distributed system architecture, the functional subcomponents of the autonomous driving system are implemented into several local computing units, as shown in Fig. 4. Information from each computing unit is shared with the common communication system. There are many benefits of the distributed system architecture compared with the centralized system architecture.

The computational complexity of the system can be reduced through the distributed system architecture. The overall autonomous driving algorithm can be large and complex depending on the mission objectives; therefore, a single computation unit may not be enough to cover all of the computations. By decentralizing the computational load into the subcomputing modules, the computational stability and efficiency of the system can improve. The decentralized computational architecture can also increase the computational performance of the entire system due to the parallel computation of the independent algorithm modules.

Fault management of the autonomous driving system can become more convenient by using the distributed system architecture. Safety is the most important factor for operating and developing an autonomous driving system. If the autonomous system uses only a single computing unit to operate all of the functions of the autonomous driving, it can be very dangerous if the single computing system fails due to the hardware or the software. Since the distributed autonomous driving system consists of several computing modules, each submodule can check the health status of other computing modules and backup the failed function.

The noise and cost of wiring can be reduced by optimally positioning the distributed computing units. In order to minimize noise problems, various grounding and shielding techniques are applied, such as single-point grounding, capacitive shielding, twisted wiring, and cabling length optimization. Among these techniques, the cabling length optimization is important for preventing electromagnetic noise in analog signals. In the cabling length optimization, the distributed system is more advantageous than the centralized system because of the configurable placement of computing units. Furthermore, the shorter length of the cable for transferring signals from the sensors/actuator has the benefit of reducing the cost and weight of cables and preventing electromagnetic noise.

Each module of the autonomous system can be developed, tested, and independently maintained through the distributed system architecture. If the autonomous driving algorithm is implemented into a single computing module, it is impossible for developers to access the development system at the same time. Additionally, modification of local software modules may affect the reliability of the entire system due to the unexpected dependence of each module. The distributed system architecture can solve the dependence problem of the single computing system by decentralizing and encapsulating it into local subcomputing modules. Developers can develop and test each submodule independently and concurrently; therefore, the development efficiency can be improved, and the stability of the unit module can be ensured.

The encapsulation of the subsystem module facilitates the maintenance and extension of an autonomous driving system. Changes to sensor configurations and required functions can frequently occur depending on the changes to the required mission. In the case of the centralized system, if a partial replacement or upgrade is needed for a part of the autonomous driving system, the entire system should be updated and tested. However, in the case of the distributed system architecture, there is no need to update and test the entire system for a partial

replacement. The submodules that need maintenance are the only ones required to be updated and tested.

D. Necessity of a Development Process and a Common Software Platform for the Distributed System Architecture

Although the distributed system has many benefits for developing an autonomous car, it also has some drawbacks to be improved upon. First, the functional configuration of the distributed system is not simple because it should consider many constraints, such as the computing power of each computer module, characteristics of the network for communication, and the location of sensors/actuators. Second, compatibility problems can occur when the different types of computing units and operating environments are used for the different subsystem modules of an autonomous driving system. Finally, network delay and jitter of the common communication systems may decrease the entire performance of the driving system.

In order to address the problems of distributed system architecture, this paper proposes a development process and a common software platform for developing a distributed autonomous driving system. The complex configuration problem of the distributed system architecture is simplified by using the proposed development process, which consists of three steps: software component design, computing unit mapping, and implementation of function. The compatibility problem of different computing units in a distributed system can be solved using a common software platform. Furthermore, to minimize the problem of network delay and jitter, the software platform supports the time-triggered protocol (FlexRay). The detailed contents of the development process and software platform will be explained in Sections III and IV, respectively.

III. Development Process for Distributed System

A. AUTOSAR

In the automotive field, there is an open and standardized automotive software architecture named AUTOSAR (see Fig. 5). The AUTOSAR is based on a component-based software design model for developing vehicular software and provides the methodology for designing the automotive software. The goals of the AUTOSAR are scalability of the software to different vehicle and platform variants, transferability of the system functions throughout the network, integration of functional modules from various suppliers, maintainability throughout the whole product life cycle, and software updates and upgrades over the product's lifetime [3], [4].

A standard AUTOSAR is used to develop automotive products with automobile manufacturers and suppliers; subsequently, there are many standard rules and interfaces available to help produce reliable automobile parts with minimum defects. These standard rules and interfaces are managed with a standard description file established by the AUTOSAR consortium. The description file should include all the configurations of the distributed system. Subsequently, it contains large amounts of detailed information on the automotive distributed system, which includes software components description, system constraints description, system resources description, and

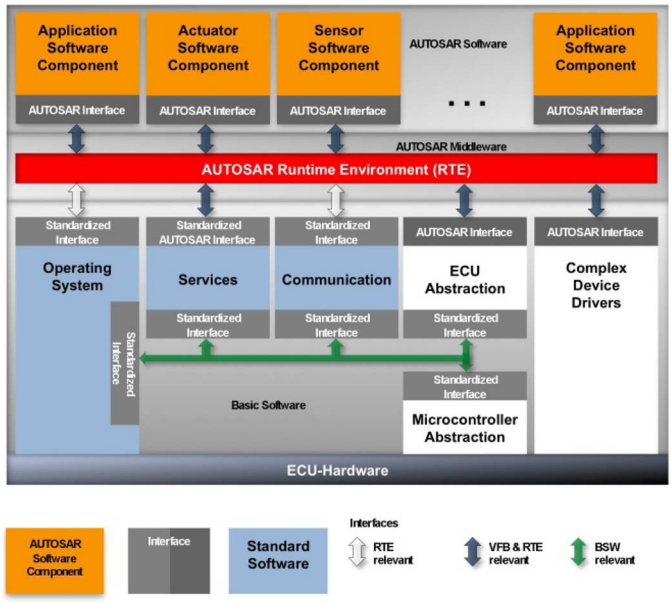

Fig. 5. AUTOSAR software architecture.

electronic control unit (ECU) configuration description [3], [4]. These highly detailed description files are acceptable for the manufacture of high-reliability automobile products; however, there is a large overhead cost when it is applied to autonomous car research. New studies on autonomous driving algorithms and system architectures will result in flexible changes and system structure extensions.

A commercial development tool chain will be indispensable to manage all the description files and follow the standard methodology of AUTOSAR. However, an AUTOSAR tool chain is too expensive to purchase by small-scale laboratories, and there are compatibility problems between the different tool chains. In order to solve this problem, previous studies [10]–[13] proposed a lightweight version of AUTOSAR called AUTOSAR-Lite. AUTOSAR-Lite reduces overhead problems caused by excessive standard AUTOSAR specifications, which retains the advantages of AUTOSAR such as scalability, reusability, reliability, and transferability. Although this study only considers single ECU components for engine control, the concept of reduced AUTOSAR can be applied to develop the distributed system of an autonomous car described in this paper.

In this paper, system development process and platform are introduced for the flexible and stable development of a distributed system for an autonomous car. The process adopts the advanced development methodology and the AUTOSAR software platform and improves the system design flexibility, omits overhead processes such as large amount of standards, documentation for system descriptions, and design tool chains.

B. Development Process for Distributed System Development of Autonomous Car

The development methodology of standard AUTOSAR is highly dependent on standard description files that have a strict template with an XML format. However, generating a standard description file requires significant work and an expensive tool chain; it is not acceptable for the products being researched that,

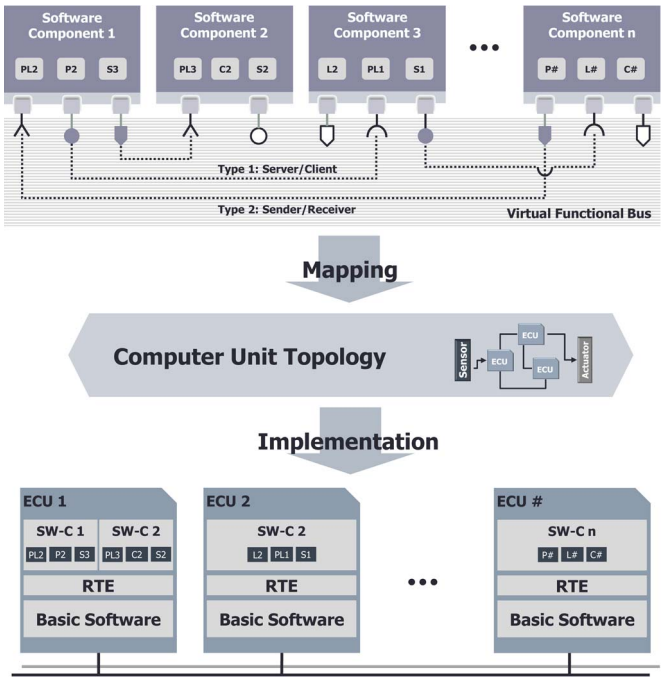

Fig. 6. Development process for distributed system of an autonomous car.

frequently, changes are needed for their specifications and system structures. Therefore, the proposed development process for the distributed system of an autonomous car advocates the philosophy of AUTOSAR development methodology; however, strict description rules are mitigated for development costs and flexibility. The developers focused on the main idea and key algorithm of the autonomous driving system with simple document files engaged in the research group instead of following the complex descriptions of AUTOSAR. The development team can designate the document rules according to the nature of developments.

The development process, which is a reduced version of the AUTOSAR development methodology, can be summarized into the following: software component design, computing unit mapping, and function implementation (see Fig. 6). The main advantage of the development process is that it can provide a common design approach to develop a distributed system.

C. Software Component Design

The software component represents the encapsulated part of the functionality in the autonomous driving application. At the software component design step, the software components of the autonomous driving functions are designed, and the flow of information between the components is defined. The connection for sharing information between the software components, sensors, and actuators can be defined by using a virtual functional bus (VFB), which is an abstraction of the communication.

The VFB has two types of communication. One of the VFBs is a client–server communication. The client is a service requester, whereas the server is a service provider, which waits for incoming requests from any client. The other type of communication is a sender–receiver one. The sender does not know how many receivers are in the network and provides the information without any requests. In general, the type of VFB chosen depends on the requirements of the communication system.

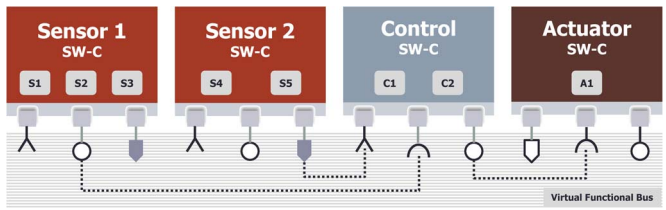

Fig. 7. Example of the soft component design including two sensors, one control, and one actuator.

The consideration of the hardware and network can be excluded by using the VFB. Therefore, the software component developer can concentrate on designing the functional elements. The abstraction of the hardware and network also helps the developer to flexibly cope with changes to hardware and network specifications.

Fig. 7 depicts an example of the software component design that includes two sensors, one control, and one actuator. For each software component, individual runnables are predefined as S1–S5, C1–C2, and A1. Next, data flows are determined using VFB. The VFB in Fig. 7 shows that the results of two sensor acquisitions are conveyed to the control unit and that the actuator receives commands from the control unit.

D. Computing Unit Mapping

The designed software components should be assigned to each distributed computing unit. In order to map the software components into the computing units, the software designer empirically deploys the software components. There are several guides or constraints to assign the software components; however, we consider the following.

First, a computing unit in charge of the sensor or the actuator should be installed adjacent to the device and contain specific software components to operate the device. Second, all mapped software components in one computing unit should work at a prescheduled time. Third, some software components might have operating system (OS) dependence due to device interfaces or available libraries. Fourth, the amount of information exchanged between different computing units should not exceed the network capability. Finally, the available hardware peripheral is limited according to the type of computing unit; for instance, a universal serial bus might not be available for the embedded computing unit.

Fig. 8 shows that the computing unit mapping methodology is applied for the previous example. Four software components can be integrated into two computing units after considering hardware installations, execution time for all software components, or other constraints.

Optimization techniques can be applied to the computing unit mapping problem in order to optimally assign the software components to the computing units [14], [15]. However, the current optimization method has the limitation that it cannot consider all types of mapping constraints. Therefore, in this paper, the software component mapping is dependent on the

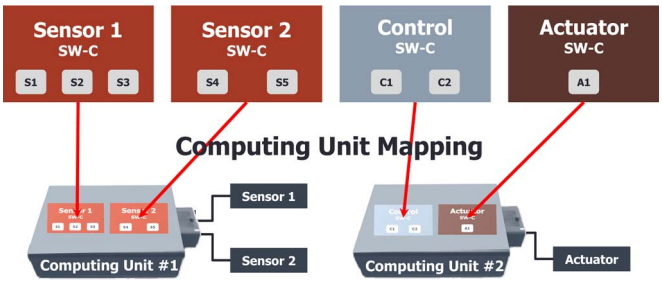

Fig. 8. Example of computing unit mapping.

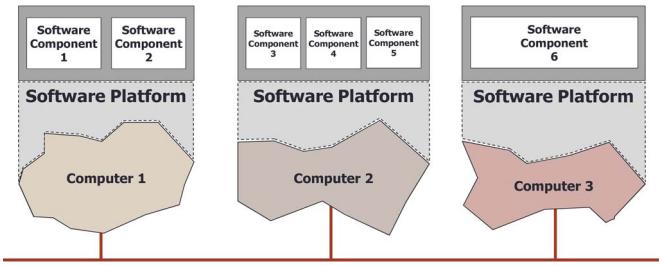

Fig. 9. Implementation of software components based on the software platform.

experience of the software designer. In future work, the authors plan to develop an efficient optimization method for the computing unit mapping problem.

E. Implementation of Functions

The software components should be implemented into an assigned computing unit platform by considering the computing performance, OS, sensor and actuator interface, and network. Therefore, the implementation of the software components may depend on the characteristic of the computing unit platform; this dependence can increase the cost of the implementation and maintenance of the software. To minimize the dependence problem of the software components, a common software platform, which can apply to different computing units, should be developed.

The common software platform is located between the application software components and the computing unit hardware in order to isolate the impact of the hardware and network on the software components, as shown in Fig. 9. The independence of the software platform can improve the reusability, transferability, scalability, and maintainability of the software and reduce the development cost. In this paper, a common system platform for a distributed system of autonomous cars is proposed for the independence of the hardware and software. The structure details of the software platform and in-vehicle network will be introduced in the next chapter.

IV. System Platform

Various types of computing units can be utilized for the distributed system of an autonomous car. Each computing unit operates on different OSs and hardware peripherals. Therefore, the developers of the autonomous driving functions should be familiar with the development environment of the corresponding computing unit and OS. However, when the functions implemented on a computing unit have to move to another type of computing unit due to a change in system requirements, it will take a long time for developers to transfer the functions to the changed computing system and to adapt to the new computing development environment. To improve the reusability of the software functions and provide a consistent development environment, a common system platform for each local computing unit is necessary.

The system platform of the distributed system architecture can be classified into a software platform and an in-vehicle network. The software platform provides a common software development environment that minimizes the dependence of the software on the hardware and OS. Therefore, the developer can achieve the reusability, reliability, transferability, and maintainability of the software. The network platform allows the distributed system to share the information for autonomous driving hardware independently.

A. Structure of Software Platform

The proposed software platform follows the basic AUTOSAR concepts described in Fig. 5. The overall AUTOSAR software structure contains too many constraints for the creative and flexible development of an autonomous car, which includes the limits of a standardized interface with huge amounts of configuration parameters and functions that use standardized templates. The AUTOSAR standard was developed for an automotive embedded system, and it cannot support a general computing system. The proposed software platform omits standard parameter configuration and auxiliary functions that overcome the AUTOSAR limitations. Instead, it focuses on interface functions and adopts a general OS for system flexibility; subsequently, the proposed software platform reduces the AUTOSAR overhead and follows the basic philosophy of the AUTOSAR.

The software platform of the distributed system for an autonomous car is a layered architecture, as described in Fig. 10. There are three layers: the application layer, the run-time environment (RTE), and the basic software layer. The application layer contains the implementation of software components. The basic software layer is a hardware-dependent layer and consists of the OS, communication modules, input and output (I/O) hardware abstraction module, and a complex driver. The RTE lies in the middle of the application layer and the basic software layer in order to remove the dependence of the software components from the hardware and networks.

1) Application Layer: The basic concept of the application layer design is a software-component-based design that is identical to AUTOSAR standards. Software components assigned to a certain computing unit are implemented in the application layer of the software platform. At this layer, only the structure of the I/O interface and functional elements are focused on to implement the software components; the information sources of the I/O, such as networks, computing unit hardware I/O, and other software components, are not considered. Therefore, a software component in the application layer is independent to the hardware, networks, and other software components. This component-based software development architecture allows the

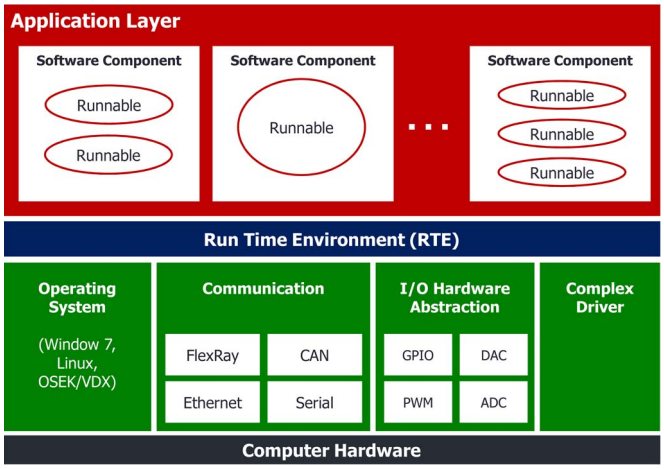

Fig. 10. Software architecture for a computing unit.

developed software to be scalable and transferable to other computing platforms.

One software component is implemented as a single software module. The software module is composed of several runnables that are the smallest schedulable units of the software. One software module for the software component can contain multiple runnables to execute the corresponding functions. The runnables are described in the single member function of the software module.

2) RTE: In the AUTOSAR standards, the RTE, the run-time representation of a VFB, is a communication center of the software platform. Each of the application software components in the application layer can communicate to each other through the RTE. The application software component can also exchange information with other computers, sensors, and actuators through basic software components such as the network and I/O hardware drivers. The RTE provides a connection between the application layer and the basic software layer.

Furthermore, to minimize the dependence of the application software component on the hardware, network, and other software components, the RTE abstracts the communication interface of the software components in the application layer and the basic software layer. The RTE provides the same interface and services whether the communication is between application software components or between an application software component and the basic software components. Therefore, the software components in the application layer can be designed without consideration of the hardware and network specifications.

The RTE also provides the scheduler mapping function for the application software components in the application layer. At the RTE layer, the runnables of the application software components are assigned the task of scheduler in the OS of the basic software layer. Therefore, the multiple runnables of the software component can concurrently execute due to the scheduling algorithm of the OS.

An RTE layer is adopted in the proposed software platform with significant advantages carried over. However, the proposed software platform only focuses on the port and interface functions for the implementation of the RTE. Additional standard AUTOSAR functions (such as trace and debug mode) are excluded. Commercial development tools for RTE configuration are not required, and developers can directly implement the RTE code with a simple configuration. In addition, code complexity, processor overhead, and memory requirements can be also reduced.

3) Basic Software: Basic software components in the basic software layer are dependent on computing unit hardware and network specifications. The implementation of the basic software differs according to the type of computing unit; subsequently, there are many complicated configurations with standardized parameters in AUTOSAR to meet this requirement. In addition, the standard basic software configurations only cover embedded system applications. However, high-performance general computing systems are essential to develop autonomous driving systems. To overcome the limitations of AUTOSAR standards, the proposed software platform focuses on basic software port and interface functions and allows a general-purpose OS for system flexibility and performance.

The basic software layer consists of four types of basic software components; these are the OS, the communication, the hardware I/O abstraction, and the complex driver. The OS software components provide the scheduler for the runnable of the software components. The different computing units of the distributed system use a different OS. The embedded microcontroller uses the OSEK/VDX RTOS, which is widely applied to the automotive embedded system. The industrial high-performance computing unit uses Windows and Linux OSs for the scheduler. The communication software component provides the common interface of the network such as FlexRay, CAN, LIN, and Ethernet. The I/O hardware abstraction component provides the common interface of the I/O peripherals such as the general-purpose digital I/O (GPIO), pulsewidth modulation (PWM), digital-to-analog converter (DAC), and analog-to-digital converter (ADC). The purpose of the complex driver is to implement complex sensor evaluation and actuator control of a non-AUTOSAR system. The complex driver provides an interface that directly accesses hardware that includes complex microcontroller peripherals and external devices. The functions can be also implemented in the complex driver component if there are hardware-specific functions that cannot be abstracted with a common interface. Therefore, software components can handle the hardware via RTE using the complex driver interface.

We can implement all functions of each computing unit by applying the software platform (see Fig. 11). Applications in charge of sensing are built in the application layer of computing unit 1, and the control algorithm is placed in the application layer of computing unit 2. The software platform provides sensing or control interfaces for the application layer.

B. In-Vehicle Network

1) Requirements of Network for Autonomous Cars: A network used for the distributed system of an autonomous car has three requirements: high bandwidth, deterministic characteristics, and fault tolerance. The high bandwidth is an essential property of the network for an autonomous car because sensors and computing units of autonomous cars generate a large

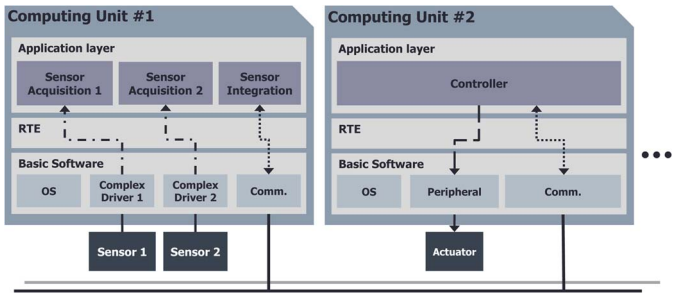

Fig. 11. Example of implementation of functions.

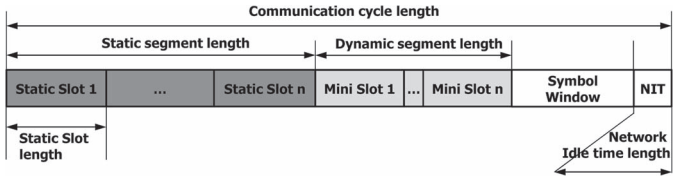

Fig. 12. FlexRay network protocol.

amount of information. It is strongly recommended that the maximum message delay for the network is predictable (deterministic), since an autonomous car is a safety-critical system. The network of safety-critical systems should transfer information within a maximum admissible delay, which is called the deadline. If the maximum delay of the message exceeds the deadline, it can be directly related to degradation of the system performance. Finally, the network of the autonomous driving system should be fault tolerant. The network of the distributed system should be able to detect the network failure and have a back-up system to recover from it.

2) Advantages of FlexRay Network: FlexRay is a time-triggered protocol, which has emerged as the next-generation network of the automotive industry [16], [17]. The FlexRay network protocol has all of the characteristics required for network of autonomous cars. FlexRay provides high bandwidth for the autonomous driving system. Since FlexRay has a robust physical layer, which uses a twisted pair cable with differential signaling, it can reduce the effect of external noises and provide reliable high bandwidth up to 10 Mb/s.

In addition, FlexRay offers a deterministic characteristic. The determinism of the FlexRay network protocol is realized by a global timer, which can be used as a common clock by all nodes in the FlexRay network. Based on this global time, network messages are sent by using the time-division multiple-access (TDMA) method. Because the TDMA method sends each message at a specified time, the time of the message transmission and reception can be predictable.

Finally, the dual-channel mode and the active star topology of the FlexRay network enhance the fault tolerance of the system. In the dual-channel mode, one channel can be configured as a replication of the other channel. This configuration is used for checking for fault in received information and hardware backup. The active star topology of FlexRay consists of several individual branches connected to a center active star node. The active star node can block error propagation from a failed branch.

3) Basic Principle of FlexRay Protocol: The FlexRay network protocol is based on a communication cycle that periodically repeats with a fixed length, as shown in Fig. 12. The communication cycle consists of a static segment, a dynamic segment, a symbol window, and a network idle time (NIT). The static segment is divided into the same length of static slots. At each slot, nodes of the FlexRay network send messages in a TDMA manner. In the dynamic segment, event-triggered messages with priority are sent by using the flexible TMDA method. The symbol window is used for network management, and the NIT is the time-correction period at each node to synchronize the global time of the FlexRay network.

4) Design of FlexRay Network: The FlexRay network design is divided into two steps: one is network parameter configuration, and the other is network message scheduling.

The parameters of the FlexRay network should be properly configured in order to take advantage of the previously described FlexRay network protocol. There are several commercial tools for configuring FlexRay that check fault configuration based on the parameter value constraints in the FlexRay protocol specification. However, the tools do not provide optimal parameter configuration for optimal network utilization because the optimal configuration is a difficult problem to solve due to the over 70 parameters and the complex relationships. There are previous studies for design optimization methods for the configuration of FlexRay parameters.

A general FlexRay response time model is defined by a heuristic method based on a FlexRay timing model [18]. This model analyzes the worst case response time of each message, and optimal FlexRay parameters are established to minimize the worst case response time. However, the optimal configuration using this method can waste network resources because this method does not consider network utilization.

Another method focused on the configuration of optimal static slot length and communication cycle length, which are highly related to the temporal behavior of the application system [19]–[21]. Two parameters are optimized by minimizing the protocol overhead, wasted network resources, and worst case response time. This method can be applied to an event-triggered system because the system model is not constrained to the synchronization of application software and communication. The proposed method is suitable to apply to development of autonomous cars since they have an event-triggered system such as emergency stop and fault management. However, this method requires an unchangeable fixed network configuration. Applying this method in the early steps of autonomous car development is unsuitable because the network configuration can be frequently modified due to system configuration changes. Thus, we recommend applying a heuristic configuration of FlexRay parameters (suitable for the frequent modification of a network configuration) before the network configuration is fixed. After the system configuration is stabilized, the optimal configuration of FlexRay parameters is acceptable with a fixed network configuration using the second method.

In the second step, network messages are scheduled to synchronize with the application software based on the following process. First, a sequence of the application software execution is defined. This is already done by the software component design step using the VFB. Second, the execution time of the application software is estimated. There are several methods

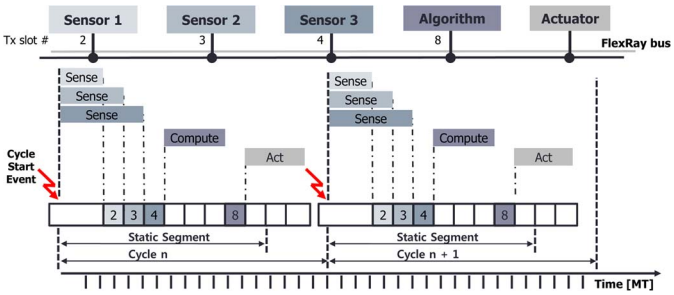

Fig. 13. Synchronization of application software.

such as the measurement-based approach, static analysis, and using commercial tools to estimate the execution time [22]. In this paper, the execution time is directly measured by an oscilloscope, because measurement is a simple and practical method. Finally, the network messages are scheduled in a static slot within the application software. The application software is arranged in order of execution sequence. The arranged application software is represented based on the global time of FlexRay using the measured execution time. After this, the application software and messages are scheduled to start execution right after receiving the results from the preceding application software.

An example of message scheduling is depicted in Fig. 13. In this example, the application software is executed in the following order: three sensor components, one computation component, and one actuator component. The application software is represented in the global time with execution time, as shown in Fig. 13. In the last step, messages from sensor components are allocated in slots 2–4, which are synchronized with the end of the execution of each sensor component. The result of the computation component is sent to slot 8 in the same manner as the sensor messages.

V. Conclusion

This paper has presented the development process and the system platform for the development of the distributed system of an autonomous car. A distributed system architecture is used for the system platform because it has many benefits for developing an autonomous driving system, such as reduction of the computational complexity of the entire system, fault-tolerant characteristics, and modularity of the system. The development process provides the guidelines to design and integrate the distributed systems of an autonomous car. A layered-architecture-based software platform, which originated from AUTOSAR, is applied to the distributed system in order to improve the reusability, scalability, transferability, and maintainability of the application software. A FlexRay network is applied for the main network of the software platform in order to improve the network bandwidth, fault tolerance, and system performance. In Part II of this paper, the development process and the system platform presented will be verified with the autonomous car A1, which is the winner of the 2010 and 2012 AVCs.

The development process of this paper consists of three steps: component design, computing unit mapping, and implementation. Among these, the second step, i.e., computing unit mapping, can have a significant impact on the development cost, period, and system performance [14], [15]. However, the mapping process presented in this paper solely relies on the experience of the designers. Therefore, in the future, the authors are planning to develop an optimization algorithm for mapping the application software components to the distributed computing units.

The proposed development process and platform for the reduced version of the AUTOSAR may be unsuitable for mass product applications because it omits many standard AUTOSAR components related to product reliability. However, the proposed compact development process and platform has advantages that can improve the degree of freedom for the distributed system design and speed up the development process. The proposed method is more acceptable for developing prototype products that require flexible changes and an extension of the system structure. The proposed development process and platform can be applied to many other industrial fields that have distributed system architecture and require a system feasibility check through rapid prototyping, such as unmanned airplanes, unmanned submarines, distributed robot systems, and factory automation.

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.*, "Autonomous driving in urban environments: Boss and the Urban Challenge," *J. Field Robot.*, vol. 25, no. 8, pp. 425–466, Aug. 2008.
[3] S. Bunzel, "AUTOSAR—The standardized software architecture," *Informatik-Spektrum*, vol. 34, no. 1, pp. 79–83, 2011.
[4] *AUTOSAR Documents (V4.1)*, AUTOSAR Consortium, Munich, Germany, 2013.
[5] N. Navet, Y. Song, F. Simonot-Lion, and C. Wilwert, "Trends in automotive communication systems," *Proc. IEEE*, vol. 93, no. 6, pp. 1204–1223, Jun. 2005.
[6] A. Elkady and T. Sobh, "Robotics middleware: A comprehensive literature survey and attribute-based bibliography," *J. Robot.*, vol. 2012, pp. 959 013-1–959 013-15, 2012.
[7] J. Kramer and M. Scheutz, "Development environments for autonomous mobile robots: A survey," *Auton. Robots*, vol. 22, no. 2, pp. 101–132, Feb. 2007.
[8] E. de Lellis, V. di Vito, L. Garbarino, C. Lai, and F. Corraro, "Design process and real-time validation of an innovative autonomous mid-air flight and landing system," *World Acad. Sci., Eng. Technol.*, vol. 79, no. 55, pp. 174–184, Jul. 2011.
[9] U. Nunes, J. A. Fonseca, L. Almeida, R. Araújo, and R. Maia, "Using distributed systems in real-time control of autonomous vehicles," *Robotica*, vol. 21, no. 3, pp. 271–281, Jun. 2003.
[10] P. Inseok, L. Wootaik, and S. Myoungho, "Application software modeling and integration methodology using AUTOSAR-ready light software architecture," *Trans. Korean Soc. Autom. Eng.*, vol. 20, no. 6, pp. 117–125, 2012.
[11] L. Kangseok, P. Inseok, S. Myoungho, and L. Wootaik, "AUTOSAR-ready light software architecture for automotive embedded control systems," *Trans. Korean Soc. Autom. Eng.*, vol. 21, no. 1, pp. 68–77, 2013.
[12] A. Monot, N. Navet, B. Bavoux, and F. Simonot-Lion, "Multisource software on multicore automotive ECUs—Combining runnable sequencing with task scheduling," *IEEE Trans. Ind. Electron.*, vol. 59, no. 10, pp. 3934–3942, Oct. 2012.
[13] T. Nolte, S. Insik, M. Behnam, and M. Sjodin, "A synchronization protocol for temporal isolation of software components in vehicular systems," *IEEE Trans. Ind. Electron.*, vol. 5, no. 4, pp. 375–387, Nov. 2009.
[14] P. Wei, L. Hong, Y. Min, and S. Zheng, "Deployment optimization for AUTOSAR system configuration," in *Proc. ICCET*, 2010, pp. V4-189–V4-193.
[15] A. Prayati, C. Koulamas, S. Koubias, and G. Papadopoulos, "A methodology for the development of distributed real-time control applications with focus on task allocation in heterogeneous systems," *IEEE Trans. Ind. Electron.*, vol. 51, no. 6, pp. 1194–1207, Dec. 2004.

[16] F. Baronti *et al.*, "Design and verification of hardware building blocks for high-speed and fault-tolerant in-vehicle networks," *IEEE Trans. Ind. Electron.*, vol. 58, no. 3, pp. 792–801, Mar. 2011.
[17] M. Barranco, J. Proenza, and L. Almeida, "Quantitative comparison of the error-containment capabilities of a bus and a star topology in CAN networks," *IEEE Trans. Ind. Electron.*, vol. 58, no. 3, pp. 802–813, Mar. 2011.
[18] T. Pop, P. Pop, P. Eles, Z. Peng, and A. Andrei, "Timing analysis of the FlexRay communication protocol," *Real-Time Syst.*, vol. 39, no. 1–3, pp. 205–235, Aug. 2008.
[19] K. Jang, I. Park, J. Han, K. Lee, and M. Sunwoo, "Design framework for FlexRay network parameter optimization," *Int. J. Autom. Technol.*, vol. 12, no. 4, pp. 589–597, Aug. 2011.
[20] 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.
[21] K. Minkoo, P. Kiejin, and J. Myong-Kee, "Frame packing for minimizing the bandwidth consumption of the FlexRay static segment," *IEEE Trans. Ind. Electron.*, vol. 60, no. 9, pp. 4001–4008, Sep. 2013.
[22] R. Wilhelm *et al.*, "The worst-case execution-time problem—Overview of methods and survey of tools," *ACM Trans. Embedded Comput. Syst.*, vol. 7, no. 3, p. 36, Apr. 2008.

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, Department of Automotive Engineering, Hanyang University. His main fields of interest are autonomous driving system, information fusion theories, distributed control systems, real-time embedded systems, and in-vehicle networks. His current research activities include system design and implementation of autonomous cars.

Junsoo Kim (S'11) received the B.S. degree in mechanical engineering in 2008 from Hanyang University, Seoul, Korea, where he is currently working toward the Ph.D. degree in the Automotive Control and Electronics Laboratory.

His main fields of interest are vehicle control, decision theories, path planning algorithms, and real-time systems. His current research activities include behavior reasoning and 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, Hanyang University.

His current research interests are in detection and tracking of moving object, and information fusion. His research activities include the design of autonomous driving system, distributed control systems, in-vehicle networks, and real-time embedded software development.

Chulhoon Jang (S'13) received the B.S. degree in mechanical engineering in 2011 from Hanyang University, Seoul, Korea, where he is currently working toward the Ph.D. degree in the Automotive Control and Electronics Laboratory.

His main fields of interest are autonomous driving system, 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, where he 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, Hanyang University. His work has focused on automotive electronics and controls, such as modeling and control of internal combustion engines, design of automotive distributed real-time control systems, intelligent autonomous vehicles, and automotive education programs.