IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received March 26, 2018, accepted April 16, 2018, date of publication April 18, 2018, date of current version May 9, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2828260

Architecture Design and Implementation of an Autonomous Vehicle

WENHAO ZONG, CHANGZHU ZHANG, ZHUPING WANG, JIN ZHU, AND QIJUN CHEN, (Senior Member, IEEE)
Robotics and Artificial Intelligence Laboratory, Tongji University, Shanghai 201804, China

Corresponding author: Qijun Chen (qjchen@tongji.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61733013 and in part by the Basic Research Project of Shanghai Science and Technology Commission under Grant 16JC1401200.

ABSTRACT Architecture design is one of the most important problems for an intelligent system. In this paper, a practical framework of hardware and software is proposed to reveal the external configuration and internal mechanism of an autonomous vehicle—a typical intelligent system. The main contributions of this paper are as follows. First, we compare the advantages and disadvantages of three typical sensor plans and introduce a general autopilot for a vehicle. Second, we introduce a software architecture for an autonomous vehicle. The perception and planning performances are improved with the help of two inner loops of simultaneous localization and mapping. An algorithm to enlarge the detection range of the sensors is proposed by adding an inner loop to the perception system. A practical feedback to restrain mutations of two adjacent planning periods is also realized by the other inner loop. Third, a cross-platform virtual server (named project cocktail) for data transmission and exchange is presented in detail. Through comparisons with the robot operating system, the performance of project cocktail is proven to be considerably better in terms of transmission delay and throughput. Finally, a report on an autonomous driving test implemented using the proposed architecture is presented, which shows the effectiveness, flexibility, stability, and low-cost of the overall autonomous driving system.

INDEX TERMS Intelligent vehicles, system architecture, sensors, mapping and planning, data transmission.

I. INTRODUCTION

Autonomous vehicles and related techniques have recently become one of the hottest research topics. As is known, the two most famous competitions to promote autonomous vehicle techniques are the 2005 DARPA Grand Challenge and the 2007 Urban Challenge. Stanford University's Junior is a well-known autonomous car, and Google's car emerged based on Junior [1]. This car was equipped with two SICK LMS 291-S14 sensors, one RIEGL LMS-Q120 laser sensor, one Velodyne HDL-64E sensor, two ibeos and cameras. The winner of the 2007 competition, "Boss" from Carnegie Mellon University, was equipped with six SICK LMS 291-S14 sensors, one Velodyne HDL-64E sensor, two Continental ISF 172 LiDARs and two Point Grey Firefly cameras [2]. Almost all of the other teams, such as Team MIT [3], "Skynet" from Cornell University [4] and "Odin" from Embry-Riddle Aeronautical University [5], used similar sensor plans to turn a vehicle into a robot. Since 2009, the National Science Foundation of China (NSFC) has organized an annual competition called "Intelligent Vehicle Future Challenge (IVFC)" [6]. In this challenge, every team needs to finish both outskirts and urban track for over 30 kilometers. Over twenty teams, mostly from universities and institutions, including Tongji University, participate in this competition every year. The appearances of most vehicles are similar to those of DARPA vehicles, and the price of sensors is typically approximately 10 times the price of the vehicle itself. For example, Xi'an Jiao Tong University [7], [8], Shanghai Jiaotong University [9], Tsinghua University [10], National University of Defense Technology [11], [12] designed their vehicles with LiDARs, Radar, GPS, cameras and other vehicle sensors. Consequently, some breakthroughs must be achieved for acceptance of the autonomous vehicle. The solution to this problem is the revolution of both the hardware and software of the autonomous vehicle. In 2014, Mercedes Benz presented a successful demonstration on a Class S 500 equipped with close-to-production sensor hardware and relied solely on vision and radar sensors in combination with accurate digital maps to obtain a comprehensive understanding of complex

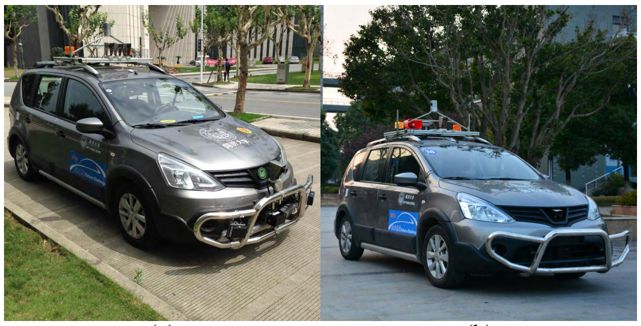


FIGURE 1. Autonomous vehicle developed by Tongji University. (a) Before 2013, most of the sensors were placed outside the vehicle and set up improperly. (b) Since 2013, the sensors have been rearranged more appropriately.

traffic situations [13]. Also, teams all over the world keep thinking of new design and algorithms of the self-driving car several years after DARPA Challenge, such as [14]–[17], etc. Currently, both automobile companies, such as Tesla [18], Mercedes-Benz, Audi, and BMW, and IT companies, such as Google, Uber, and Baidu, are all devoting considerable efforts to developing autonomous vehicles. More importantly, with the development of wireless communication, autonomous vehicles enable cooperation among multiple vehicles, which is commonly known as multi-agent systems [19], [20]. With the development of intelligent vehicles, new traffic and transportation systems are also being studied worldwide [21]–[24], etc. To date, state-of-the-art algorithms have been revealed in a large number of papers, including vision perception, local planning, navigation, localization, control theories and so on [21], [25]–[31]. However, few papers place an emphasis on system architectures, without which algorithms are simply impractical independent accessories. Although there exists some papers talking about the system and architecture design, such as [1], [2], [4], [9], [17], [32]–[36], the problems can be summarized as follows.

- They only talked about how they built the vehicle platforms, but did not give detailed analysis on the mechanism such as the sensor arrangement.
- The actuators of the auto driving system is important, but most of the papers put emphasis on the drive-by-wire techniques, which is infeasible to an old vehicle or a vehicle which can not be lossily modified.
- Although the system diagram was given, not enough quantitative inner links were revealed. The readers only know which part of the modules should be connected together, but how to execute from the data level was neglected.
- Not enough quantitative experiments were demonstrated in the above papers. Although some of these had the analysis on control performance, the other modules such as perception and data transformation did not prove to cooperate well on the entire system level.

Fig. 1 shows the evolution of the fully autonomous vehicle developed by Tongji University (named Tongji's autonomous vehicle in the following paper) before and after 2013. This vehicle has been tested on Tongji University's campus and on specific testing fields in Changshu, China, for over 5 years, during which solutions for both external sensors and internal architecture have changed considerably to significantly improve the overall system performance. The solutions mentioned here are at the system level rather than algorithms, such as perception, planning or control. For example, the solutions include the rationality of the sensor plan, the efficiency of data transmission, the fusion of algorithms in separate modules, the stability of the entire system and so on.

In the following sections, we consider the overall architecture of Tongji's autonomous vehicle. In Section II, two main parts of the hardware framework are introduced to analyze the sensor plan from different perspectives and to solve the problems of external automated chassis control. In Section III, the software system architecture is introduced to explain how the modularized system works and two key loops of applying SLAM to ensure better performance of perception and planning with inadequate sensors. In Section IV, a detailed introduction to the data transmission module is presented, and its performance comparison with the Robot Operating System (ROS) [37] is shown in Section V. In Section VI, a short report of an autonomous driving test is presented. The conclusions and future work are discussed in Section VII. We sincerely hope to provide anyone who wants to research fully autonomous vehicles some good advice and a mature template in this paper.

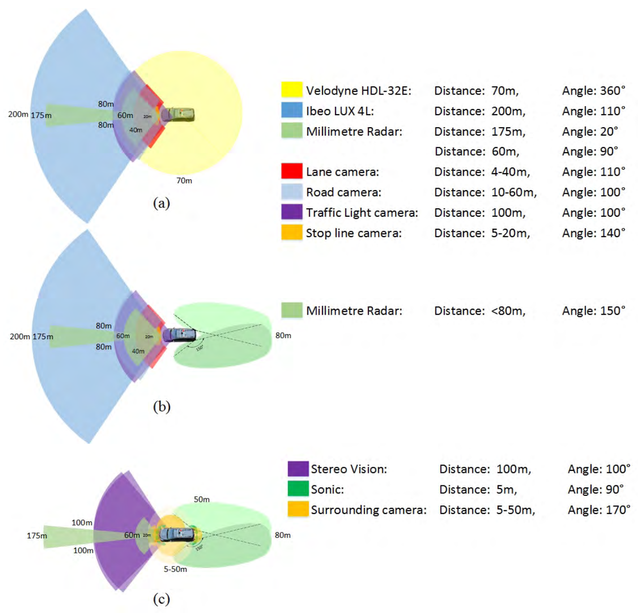
II. HARDWARE FRAMEWORK

For a vehicle to drive by itself, it needs "eyes", a "brain", and a "body". Consequently, sensors, CPUs and actuators are added to a regular vehicle such that it can perceive, think and do as humans do. In this section, we primarily focus on two aspects. The first aspect is the sensor placement plan, which is a difficult and vital link, and the second aspect is a general autopilot that serves as the actuator to control the steering wheel, accelerator pedal, break pedal and gear selector.

A. COMPARISON OF THREE SENSOR PLANS

The sensors placed on this vehicle can almost cover every typical type available on the market, including cameras, millimeter-wave RADAR, ultrasonic sensors, LiDAR, GPS, and inertial senors, which considerably differ from each other in many aspects, such as range, distance, function, performance and so on. In this subsection, three sensor plans with different costs, performances and background algorithms will be demonstrated.

A suitable sensor plan will balance function, range and cost; otherwise, serious accidents may occur caused by weather or blind spots, among others. Therefore, a brief introduction to some typical sensors is given as follows. For cameras, different sight ranges and distances can be covered through different lenses. Generally, a 16 mm focal length will be used for front detection to balance distance and field angle (FOV). If the FOV is the most important



Velodyne HDL-32E: Distance: 70m, Angle: 360°
Ibeo LUX 4L: Distance: 200m, Angle: 110°
Millimetre Radar: Distance: 175m, Angle: 20°
Distance: 60m, Angle: 90°
Lane camera: Distance: 4-40m, Angle: 110°
Road camera: Distance: 10-60m, Angle: 100°
Traffic Light camera: Distance: 100m, Angle: 100°
Stop line camera: Distance: 5-20m, Angle: 140°
(a)
Millimetre Radar: Distance: <80m, Angle: 150°
(b)
Stereo Vision: Distance: 100m, Angle: 100°
Sonic: Distance: 5m, Angle: 90°
Surrounding camera: Distance: 5-50m, Angle: 170°
(c)

FIGURE 2. **From top to bottom: Sensor ranges and distances of plans A, B and C. Different colors stand for different sensors.**

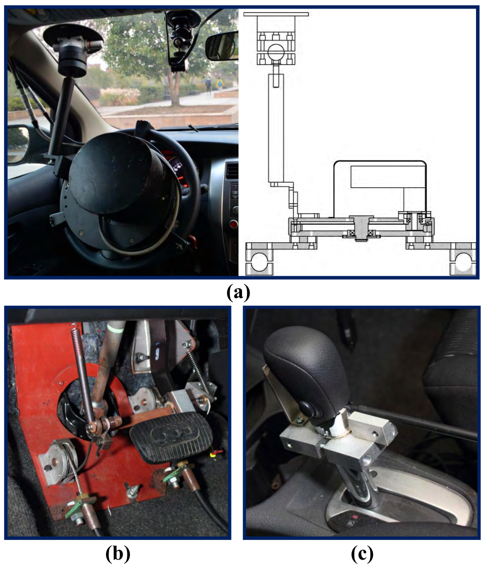


FIGURE 3. **(a) Steering wheel controller is installed on the steering wheel and fixed on the front window. (b) Accelerator and brake pedals are pulled by steel cable and pulleys placed on the pedals. (c) Gear selector is pulled by lever that is connected to the motor in the trunk.**

concern, a 6 mm lens or even a fish-eye lens may be an alternative. However, note that distortion will sacrifice the detection distance. LiDAR can provide point clouds of the environment directly and is much more stable compared with cameras thanks to its insensitivity to luminance. However, this type of sensor is not applicable for heavy rain or snow, and it is typically 10 times the price of a camera. For millimeter RADAR, this type of sensor generally works in combination with cameras for detecting moving metal objects. Although the sensitivity of millimeter wave RADAR is relatively weak, the resolution is inversely proportional to range; thus, the RADAR generally has to detect range and distance. In addition, this is an all-weather sensor because of its good penetrability. Ultrasonic sensors appear on almost every modern vehicle and are mainly used for low-speed parking due to the short distance detection ability, typically less than 5 meters. This type of sensor is clearly the cheapest sensor. Three sensor plans are shown in Fig. 2.

1) PLAN A

Four cameras, one millimeter wave RADAR, one 32-layer LiDAR, one 4-layer LiDAR and one GPS+inertial sensor are utilized in this plan. The cameras can be used for many types of detections with the help of LiDARs and millimeter wave RADARs. This plan is the closest to Google's. The advantage of this plan is that it can cover most of the area surrounding the car, as well as be adaptive to different traffic scenes and weather conditions. However, the disadvantages of this plan are its extremely high cost and inadaptability for parking tasks as distances <1 m are undetectable. In conclusion, this plan is the best for regular city driving, but it is a redundancy in highway scenes.

2) PLAN B

The difference between this plan and Plan A is that the 32-layer LiDAR is replaced by two back-side millimeter wave RADARs. A considerable enhancement in economy is obtained at the expense of losing some city driving performance since the surrounding information is greatly reduced. Similarly, parking is still a challenging problem under this framework. Based on the overall qualities, we recommend this plan for most users.

3) PLAN C

From the bottom image in Fig. 2, it is easy to observe that this plan is considerably different from Plans A and B. Only two regular cameras, three millimeter wave RADARs, one surrounding camera and one twelve-unit ultrasonic sensor are utilized. From the perspective of economy, this is the cheapest plan and can be accepted by most vehicle manufacturers. However, since the system relies mostly on computer vision, stability is the largest problem. To this end, much related work is being conducted worldwide. In addition, by using the surrounding camera and ultrasonic sensor, parking is not a problem. As long as some satisfactory results can be obtained with vision-based SLAM, drivable area detection and so on, this inexpensive and lightweight plan will be the best choice in the near future. In recent research, deep-learning-based end-to-end vision solutions have been proposed by companies such as NVIDIA and Comma.ai [38].

B. GENERAL AUTOPILOT OF A VEHICLE

An autopilot is a device that enables a vehicle to automatically turn, shift, accelerate and decelerate. This autopilot is applied under the circumstance that the vehicle is not equipped with

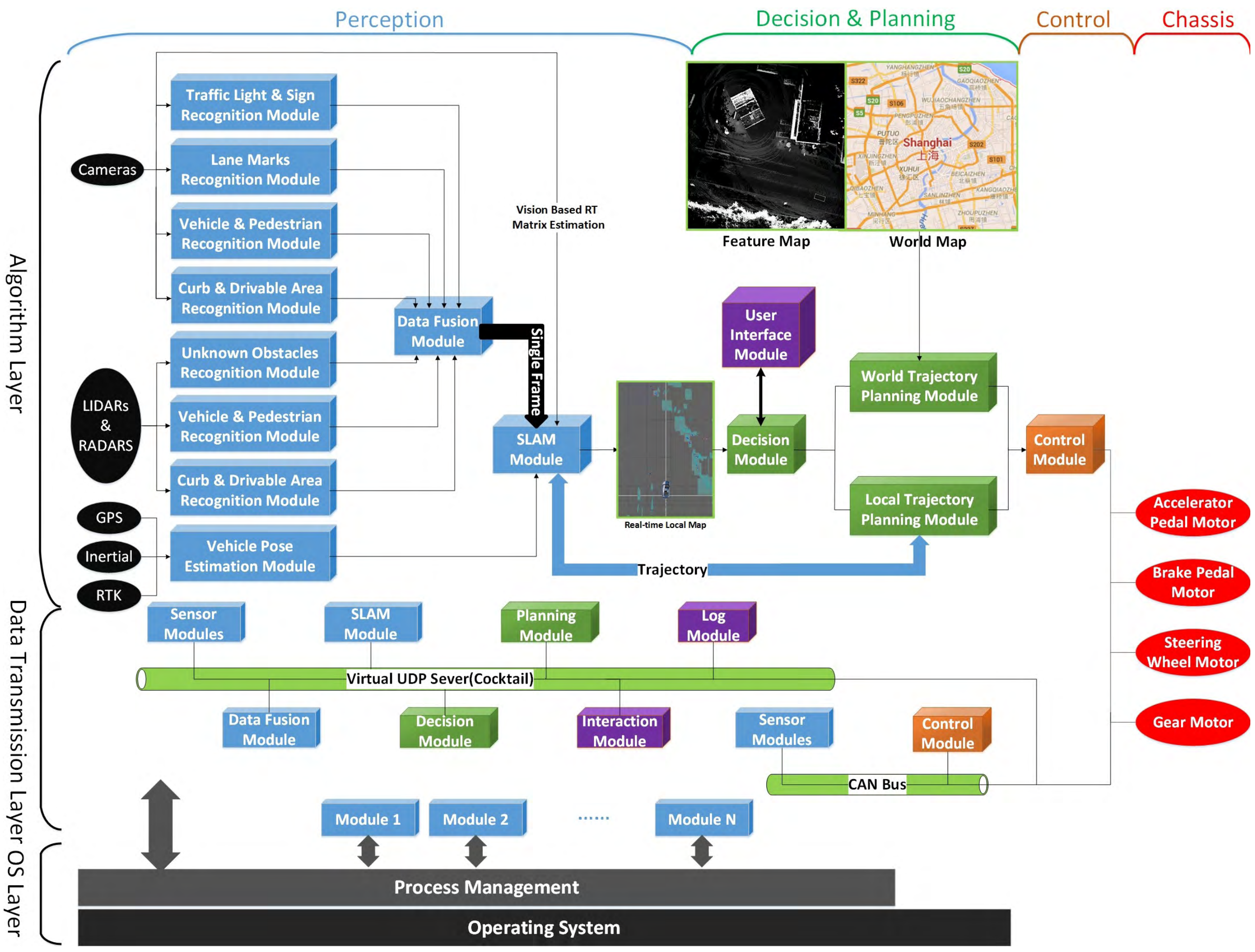


FIGURE 4. Software system architecture.

a wired-control chassis. Although wired-control must be the future trend, most of the vehicles need such a device now since the vehicle updating period is typically 5 to 10 years. As shown in Fig. 3, five main parts, including steering wheel, accelerator/brake pedals, gear activator and a central controller, constitute the chassis control device. The principal of our design is efficiency, lossless and feasible to most of the situations. We do not want the actuators be like a robot sitting on the driver's seat [39].

The steering wheel control is the most difficult among these five parts. Considering that it should be convenient for users to install and remove this device without causing any damage to the original vehicle, it is driven by a motor in the three-talon mechanism clutched on the wheel and attached to the front window, as demonstrated in Fig. 3 (a). It only takes a novice less than 5 minutes to dismount and remount this mechanism. Users can also choose to conveniently drive by themselves with this mechanism. The pedals and gear selector are all controlled by steel cables connected to the rockers on the central controller, also driven by three inside motors installed in the back trunk. These mechanisms do not affect human driving; thus, remounting is unnecessary after being equipped. Here, data transmissions are all through a controller area network (CAN) bus to make it equivalent for a wired-control chassis. The advantage of this autopilot is its practicability and universality, and it can be equipped on almost any vehicle in less than an hour without modifying the original vehicle structure. The performance indices are shown in Table 1.

III. SOFTWARE SYSTEM ARCHITECTURE

The software system architecture is shown in Fig. 4. From left to right, the automated system is divided into five parts by task, which are perception, decision, planning [40], control [41] and chassis. Perception is the most colorful but difficult task of all, and without any perception, the vehicle is like a blind man. Hence, researchers have placed great efforts on detecting vehicles, pedestrians, lane markers and traffic signs since the last century. Another essential task belonging to perception is localization, and it cannot be separated from

TABLE 1. Autopilot performance indices.

	Steering Wheel	Accelerator Pedal	Brake Pedal	Gear
Motor Drive (A/V)	15/60	5/60	10/60	5/60
Motor Volt (V)	24	24	12	24
Torque $(N \cdot m)$	13	4	14	30
Output Tension (N)	/	114	312	402
Adjust Range	$-500° \sim 500°$	$70mm$	$100mm$	$130mm$
Adjust Rate $(\leqslant \cdot/100ms)$	$20°$	$14mm$	$20mm$	$13mm$
Response Time $(\leqslant ms)$	30	100	50	500

SLAM, which will be discussed in detail in the following subsections. If perception modules are the "eyes" and "ears" of an autonomous vehicle, then the decision and planning modules can be deemed as the "driving brain", still closely related with SLAM, which will also be discussed in detail. Although Tongji's autonomous vehicle is fully automated, it is still possible for a human to interact with it through the user interface module. As long as the order follows the safety rules evaluated by the "driving brain", a human is able to tell the vehicle to do what he requires or even take control actions on the vehicle. In addition, the log module makes the system more dependent. This module consists of two parts: exception logs to record errors of unhandled cases and offline data logs to save environment, trajectories, control output data, and so forth on hard disks for a simulator to playback typical scenes. From bottom to top, the automated system is a three-layer model (OS layer, data transmission layer and algorithm layer). To make the system more flexible, a hierarchical architecture is necessary to isolate the logic, data management and operating system from each other such that it is easy for developers to modify any part of the system without influencing other parts. The OS layer consists of the operating system and process management module (PM) monitoring every other module by a heart beat signal to judge whether their states are normal. For our software system, the data transmission module is of the greatest priority. If the occurrence of a fault is detected by the PM, then a restart is necessary for all the modules after the PM restarts first. Among these three layers, data transmission interacting with almost every other module is considered in Section III. In the following subsections, two key inner loops will be introduced to demonstrate the relationship between real-time SLAM and other parts of the system.

A. REAL-TIME MAPPING AND FUSION LOOP

Occasionally, the perception module is unable to cover a sufficient range for self-driving, such as sensor plans B and C

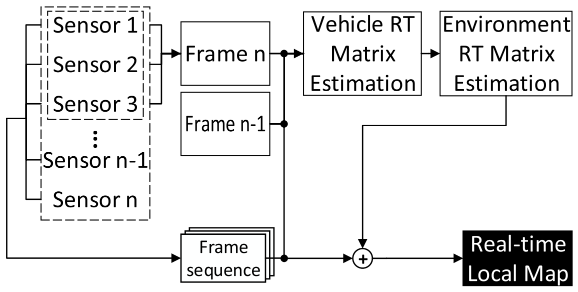


FIGURE 5. Environment mapping loop diagram.

with inadequate sensors because of the cost; thus, an algorithm to solve this problem by estimating the probability of the positions is proposed in this subsection. First, the outputs of some sensors will be chosen to perform fusion as the input of the vehicle rotation and transmission (RT) matrix $\boldsymbol{E}_{RT}$ estimation. Note that the sensor type can be completely arbitrary, and the data for fusion also include cloud points, feature points and detection results such as lane markers, road curbs and so forth. After a generalized fusion map frame is obtained, together with history frames, the vehicle $\boldsymbol{E}_{RT}$ can be estimated by algorithms such as bundle adjustment (BA). Then, the matrix is transferred to the environment perspective to renew the fusion data map. By combining current and renewed data together, the local driving map can be obtained for the decision and planning modules. This process is demonstrated in Fig. 5. To describe the algorithm, a concept named "probability ellipse (PE)" should first be introduced, in which every point is evaluated by a probability value that decreases to zero from the center to the edge. Additionally, different ellipses can overlap with each other, for which the overlapping section is evaluated by the average probability values $C_{(x,y)} = \sum_{i \in L} C_{(x_i,y_i)}/n$, where L represents sets of overlapping ellipses and n is the number of elements in set L.

Here, the vehicle position, velocity and acceleration in the i^{th} frame of the environment are denoted by sets $\boldsymbol{M}_{p_i}, \boldsymbol{M}_{v_i}, \boldsymbol{M}_{a_i}$, respectively, and $\hat{\boldsymbol{M}}_{p_i}, \hat{\boldsymbol{M}}_{v_i}, \hat{\boldsymbol{M}}_{a_i}$ represent the data observed by sensors in the *ith* frame. Hence, by assuming that the vehicle acceleration does not change between two consecutive sampling instants, the environment position estimation is given by (1)

$$\begin{aligned} \boldsymbol{M}_{p_i} &= \boldsymbol{E}_{RT_i}\left(\boldsymbol{M}_{p_{i-1}} + \Delta t \boldsymbol{M}_{v_{i-1}} + 0.5\Delta t^2 \boldsymbol{M}_{a_{i-1}}\right) \cup \hat{\boldsymbol{M}}_{p_i}, \\ \boldsymbol{M}_{v_i} &= \boldsymbol{M}_{v_{i-1}} \cup \hat{\boldsymbol{M}}_{v_i}, \\ \boldsymbol{M}_{a_i} &= \boldsymbol{M}_{a_{i-1}} \cup \hat{\boldsymbol{M}}_{a_i}, \end{aligned} \tag{1}$$

where δt is the sampling period. If the k^{th} point $M_{P_i}^{(k)} \in M_{P_i}$ and $M_{P_i}^{(k)} \notin \hat{\boldsymbol{M}}_{p_i}$, then this point is either out of sensor range or misdetected in this frame, for which PE should be applied, given by

$$\boldsymbol{M}_{p_i} = \boldsymbol{M}_{p_i} \cup \left\{(x, y)\, | x^{(k)^2}/R_{x^{(k)}}^2 + y^{(k)^2}/R_{y^{(k)}}^2 < 1\right\}$$

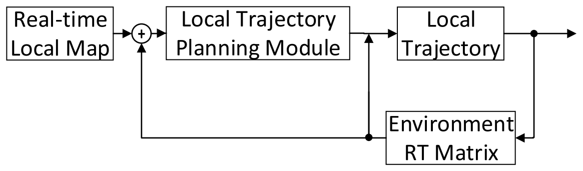


FIGURE 6. Detailed local planning loop diagram.

$$s.t.\begin{cases} R_{x^{(k)}} \propto D_{a_{x^{(k)}}} \\ R_{y^{(k)}} \propto D_{a_{y^{(k)}}} \\ C_{(x^{(k)},y^{(k)})} \propto \left(1/D_{a_{x^{(k)}}} + 1/D_{a_{y^{(k)}}}\right)\Big/2 \end{cases} \tag{2}$$

where $C_{(x^{(k)},y^{(k)})}$ is the probability of center point $\left(x^{(k)}, y^{(k)}\right)$, $R_{x^{(k)}}$ and $R_{y^{(k)}}$ are two axes of PE, and $D_{a_{x^{(k)}}}$ and $D_{a_{y^{(k)}}}$ are the variance of acceleration of $\left(x^{(k)}, y^{(k)}\right)$ in a period of time. After each point in $\left\{(x, y)\,|M_{P_i}^{(k)} \in M_{P_i} \cap M_{P_i}^{(k)} \notin M_{P_i}\right\}$ is expanded by PE, the probability of each point M_{a_i} will be calculated with consideration of overlap. Finally, a probability threshold δ is selected for determining the final general perception result.

The advantage of this algorithm is tremendous when the vehicle sensor plan has some blind spots. With integration of the environment, the current blind spots can actually be covered by history frames. In addition, sensor data will be greatly increased, which favors some low-density sensors such as SICK LMS 291-S14 or ibeo LUX.

B. LOCAL PLANNING LOOP

One of the most challenging problems in solving dynamic trajectory planning is trajectory mutation in two adjacent planning periods, which results in substantial steering wheel shaking during a serious crash. This is because the vehicle pose changes between two planning periods. To make the trajectory transit smoothly, a feedback is introduced where the last planning trajectory should be renewed by multiplying by $\boldsymbol{E}_{RT}$ if the trajectory is not updated. This process is demonstrated in Fig. 6. The general equation for feedback planning is given as follows:

$$\begin{gathered} \boldsymbol{E}_{SSD}\left(\boldsymbol{x}_n, \boldsymbol{y}_n, \boldsymbol{\psi}_n\right) = \boldsymbol{p}_n - \boldsymbol{E}_{RT}\boldsymbol{p}_{n-1} \\ \underset{\boldsymbol{x}_n,\boldsymbol{y}_n,\boldsymbol{\psi}_n}{\arg\min} \left\|\boldsymbol{E}_{SSD}\left(\boldsymbol{x}_n, \boldsymbol{y}_n, \boldsymbol{\psi}_n\right)\right\|_2 \end{gathered} \tag{3}$$

where the 2D position and heading of the vehicle are denoted by $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\psi}$, respectively. The environment RT matrix is denoted by $\boldsymbol{E}_{RT}$. $\boldsymbol{p}_n\left(\boldsymbol{x}_n, \boldsymbol{y}_n, \boldsymbol{\psi}_n\right)$ and $\boldsymbol{p}_{n-1}\left(\boldsymbol{x}_n, \boldsymbol{y}_n, \boldsymbol{\psi}_n\right)$ stand for two adjacent plans.

IV. VIRTUAL SERVER FOR DATA TRANSMISSION

As shown in the above section, every module is independent from each other except for data; thus, data transmission plays a vital role in the system. Thus, the data transmission module should be stable, safe, efficient and bug-free. Every involved module can be viewed as a node connecting to it, with no influence on others, when the corresponding node is out of work. If module A only needs data from B, then the others will not send any data to A. Since a transmission delay is fatal for automated driving, efficiency should receive the greatest attention. Although the Robot Operating System (ROS) is a well-known system that has been widely used for robots, it is found that this system is not very suitable for an autonomous vehicle after some testing because of its real-time performance and cross-platform ability. Thus, to solve these problems, a data transmission module for an autonomous vehicle is developed, named "Project Cocktail". Project Cocktail is a pivotal data transmission and exchange system that can receive/forward messages from/to network peers (using user datagram protocol). Although designed for this specific application, Project Cocktail is actually a general-purpose intermediary that will treat all data flows just as binary streams, and it does not rely on any concrete data format of the network peers. Its design allows it to be quite resilient in the sense that it works efficiently in the face of many types of data generated from a large number of network peers.

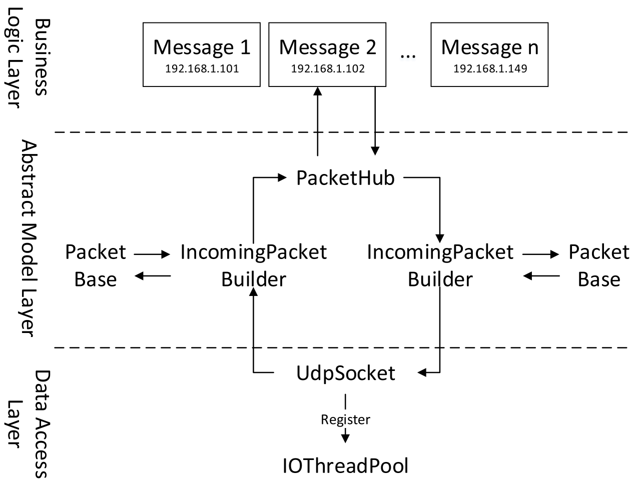


FIGURE 7. The three-layer model of Project Cocktail.

To address the objectives of Project Cocktail identified above, it is necessary to become organized. Layered design has proven to be particularly effective in the computer software literature. The primary idea of layers is that each layer hides the operation of the layer below from the layer above and instead provides its own interpretation of all the important features of the lower layer. This allows us to replace any layer with another implementation without influencing the logic of other layers. This is also the reason for the name "Cocktail", which means that the system is a fusion of data along with a clear layer logic design. The architecture of Project Cocktail is shown in Fig. 7. The server and client logic are given in Algorithms 1 and 2.

A. DATA ACCESS LAYER

At the bottom of Project Cocktail, there must be some underlying mechanism that leverages the operating system

Algorithm 1 Project Cocktail Server Logic

Input: Packets from Modules
Output: Packets to Modules

```
Listen for every module $m$ in parallel
while module $m$ received packet $P_m$ do
    if $P_m$ is Connection and $m$ is allowed to be connected then
        set TargetAddress[$m$] to the IP:port from $P_m$
    else if $P_m$ is DisConnection and TargetAddress[ $m$ ]is IP:port from $P_m$ then
        set TargetAddress[$m$] to NIL
    else if $P_m$ is DataUpdate then
        if $P_m$ is SinglePacketData then
            Send $P_m$ to TargetAddress[Target specified in $P_m$]
        else if $P_m$ is PartialPacketData then
            Send $P_m$ to TargetAddress[Target specified in $P_m$]
        end if
    end if
end while
```

Algorithm 2 Project Cocktail Client Logic

Input: Packages from Server and Local Sensors
Output: Packages to Server

```
Send Connection Packet $P_{Connection}$ to Server
while listen $P_i$ from Local Sensors and Server do
    if $P_i.prop$ is SensorData then
        if $P_i.size$ is smaller than 60K then
            Construct SinglePacketData DataUpdate and Send to Server
        else
            Split $P_i$ into pieces
            Send every piece of data as PartialPacketData DataUpdate to Server
        end if
    else if $P_i.prop$ is ServerData then
        if $P_i.prop$ is DataUpdate then
            if $P_i.prop$ is SinglePacketData then
                Deal with the data from SinglePacketData
            else if $P_i.prop$ is PartialPacketData then
                if All other PartialPacketData has been received then
                    Combine all PartialPacketData and deal with it
                else
                    Cache this PartialPacketData in local storage
                end if
            end if
        end if
    end if
end while
Send Disconnection Packet to Server
```

(OS)-provided features such as “socket” and “thread”. The *data access layer* is responsible for managing these low-level operations. The goal of the data access layer is to provide easy-to-use and robust utilities to the upper layers, hiding the particular mechanics of the OS services involved.

Concurrency is always the key to the high-performance computing of modern hardware with multi-core processors. Thread is a typical approach that is widely used across many modern OSes. In Project Cocktail, we introduce the “IoThreadPool” class, which could be treated as an intelligent task scheduler. Whenever an IO operation (e.g., socket data received, file read, and so on) completes, this scheduler is smart enough to determine what the best strategy for the current scenario is, i.e., whether to create a new thread to process the data or reuse an existing but already terminated thread to process the data, and which processor the thread will be assigned to such that the system throughput is maximized.

Another underlying infrastructure is the network IO operations (i.e., sending/receiving). BSD socket is the de facto universal interface standard for such operations. Taking Windows for example, Project Cocktail utilizes the WinSock APIs the Windows-specific implementation of BSD Socket, which is also highly optimized for Windows OS to wrap all dirty works such error handling (e.g., DNS not found, target not available, timeout, and so forth), data marshaling (e.g., mapping from little-endian to big-endian, conversion between bytes array and UDP packets, and so on) and IoThreadPool integration. This design ensures that it is not necessary for the upper layer to worry about all these low-level operations; what they will see is only a simple interface for sending/receiving data to/from the network, and the data access layer smartly handles all performance-related and robustness-related issues to allow the entire computer system exert its full power.

B. ABSTRACT MODEL LAYER

The *abstract model layer* is the middle layer of the three-layer model, which interprets the raw binary stream (i.e., the byte array used by the data access layer) in the sense of a more logically organized data structure called a “packet”. This layer consists of three main parts: OutgoingPacketBuilder, IncomingPacketBuilder, and PacketHub.

1) OUTGOINGPACKETBUILDER

OutgoingPacketBuilder is intended to be used by the PacketHub, which needs to specify the destination and the packet to be sent. The builder will compose a byte array from the packet of the sender and then pass this byte array to the data access layer to actually send the data to the destination module.

2) INCOMINGPACKETBUILDER

Once the *data access layer* of the destination module receives data from the network, it will upcall the IncomingPacketBuilder and pass the byte array just received to that builder. The task of IncomingPacketBuilder is just to decompose from the byte array and constructs a new packet instance, which is then transferred to the PacketHub.

3) PACKETHUB

PacketHub is the heart of this layer, and it is also the only interface that can be used by the components of the upper layers. By using PacketHub, the invoker is able to attach listeners to different types of packets, and it is also able to send all types of packets to the destination module. All the works, such as composing messages, maintaining packet order, and allocating buffer for different types of packets, will be considered at this layer and be transparent to all the invokers.

C. BUSINESS LOGIC LAYER

We can now create the complete picture. The data access and abstract model layers together provide a high-performance network packet receiving/sending utility, which meets the requirements that we mentioned above (e.g., generic, resilient, efficient, and so forth). The remaining part is the most interesting one, namely, implementing business logic for the entire application, including the server part and client part.

The business logic layer is divided into several components. Each component, also named a "transaction", is completely independent of other components. This methodology of limiting the interactions among components is a proven way to enforce modularity. Every transaction is either a client transaction or server transaction, depending on how it handles packets.

V. TESTING OF PROJECT COCKTAIL

Generally, the performance of a distributed system can be evaluated by the transmission delay, throughput capacity and loss ratio. In this section, three test cases will be designed to compare Project Cocktail and ROS from these three main perspectives. All of the cases will be tested under two typical transmission modes. The first mode is called single-to-single (s2s), in which each module only sends and receives messages from one module, which forms a chain structure connected end to end. The second mode is called all-to-all (a2a), in which each module sends and receives messages from all the other modules except itself, which forms a complete graph. The sketch maps are shown in Fig. 8(a), (b).

The transmission system can be defined as a directed graph $G(M, T)$, where $M(p)$ stands for the set of n modules, packet size is denoted by p, and edges $T(t)$ stand for the set of connections between each individual module with transmission delay t. $T_{ij}(t) = 0$ when $t = \infty$, which means that a connection between i and j does not exist; otherwise, $T_{ij}(t) = 1$. Additionally, transmission between modules i and j can be denoted as $\langle m_i, m_j \rangle$. Hence, the total data size $Q = \sum_{i=0}^{n} \left(\sum_{j=0}^{n} T_{ij} p_i \right), i \neq j$. Suppose that $p_i = \hat{p}$, $Q_{s2s} = n\hat{p}$, and $Q_{a2a} = n(n-1)\hat{p}$, which is $n-1$ times Q_{s2s}.

The following comparison test has been implemented on a server with two Intel(R) Xeon(R) CPUs @ 2.3 GHz, 32 GB of RAM and a 64-bit operating system. All the modules are run in the same localhost to avoid uncertainty arising from a physical cable. Seven packet sizes, including 100 B, 1 KB, 10 KB, 100 KB, 1 MB, 10 MB and 100 MB, have been chosen to represent data sizes from tiny to large. Moreover, the number of modules that we choose is 5, 10, 20 and 50, among which the 100 MB packet size has been tested only with 5 and 10 modules in mode a2a due to the RAM constraints of the server.

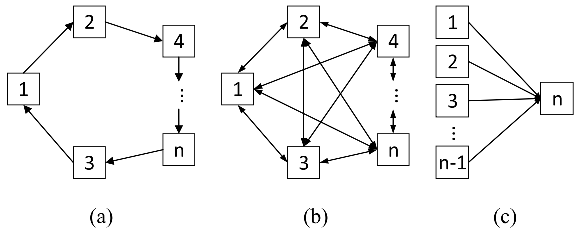


FIGURE 8. (a) Transmission mode s2s. (b) Transmission mode a2a. (c) Transmission mode a2s.

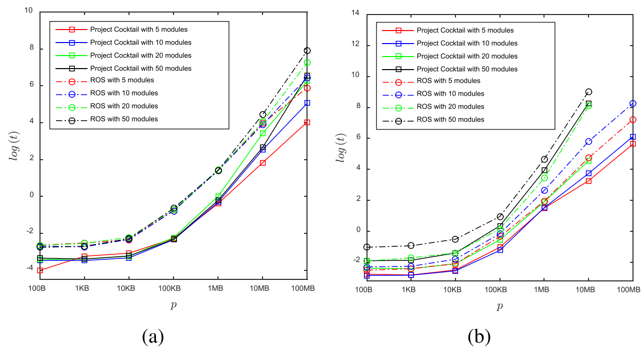


FIGURE 9. Transmission delays of two typical modes. (a) Transmission delay of mode s2s. (b) Transmission delay of mode a2a.

A. TRANSMISSION LATENCY

Transmission latency is the period of time between sending and receiving, and it is one of the most important indices for evaluating the performance of a real-time system. In this test case, latency is evaluated via the sending time stamp within the packet. $log(t)$ is utilized to increase the deviations of different packet sizes. From the test results shown in Fig. 9, the transmission delay increases with increasing packet size and number of modules. In addition, although 5 modules of mode a2a is equivalent to 20 modules of mode s2s in terms of total data amount, the transmission delay in a2a is larger than that in s2s, which indicates that each module should listen to other modules as little as possible; similarly, repeating listening to one module should be avoided as much as possible. The comparison indicates that Project Cocktail performs better than ROS in terms of transmission latency.

B. THROUGHPUT RATIO

Throughput ratio is also a vital index for evaluating the performance of a system. With the explosion of data, irrespective of whether data are from local sensors or the cloud, it is likely that the system needs to process a large amount of data in one control period, particularly for complex systems. Regarding autonomous vehicles, cameras, LiDARs, maps, and so on are all large data sources. Throughput ratio determines the maximum amount of data that can be processed by CPUs, irrespective of how efficient or accurate intelligent algorithms are. Generally, throughput ratio is influenced by both hardware and software, among which software is actually the decisive factor. In this test case, throughput ratio will be compared between Project Cocktail and ROS. Here, $\hat{p} = 10$MB is selected, and the sending interval t_{int} starts from 3000 ms to 1000 ms. In the results shown in Fig. 10, the color bar indicates the sending interval, which decreases 5 ms every single test. In Project Cocktail, since the system is optimized with instant transmission latency, which increases with decreasing sending interval, if the sending interval is less than the transmission delay, it will continuously increase with the arrival of new packets until complete block 10(b). In ROS, this situation is better with the quick increase in latency at first. Since an autonomous vehicle is a real-time system, a continuous latency increase will do harm to almost every module, particularly to those modules that are time relevant, such as SLAM, planning and tracking. Consequently, a sending interval threshold δ is chosen to define a sudden change in the average transmission delay $\bar{t}$ to represent the maximum throughput ratio. Hence, the throughput ratio is defined as

$$TP = \frac{n\,(n-1)\,\hat{p}}{t_{ed} - t_{st}},$$

where $t_{ed} - t_{st}$ is the duration from the first packet sent to the last packet received. In this test, the maximum throughput ratios of Project Cocktail and ROS with our definition are 83.2 MBps and 80.9 MBps, respectively.

C. LOSS RATIO

After the transmission delay and throughput ratio are tested on localhost, packet loss does not occur for either of the systems except for the complete block when CPU occupancy remains at 100 percent. Since packet loss between terminals connected by wires or even connected wirelessly undoubtedly exists considering the uncertainty of physical links, eve though loss rate is an important index, this test is omitted in this circumstance.

D. PARALLEL TRANSMISSION PERFORMANCE

To evaluate the parallel transmission performance, a special mode called all-to-single (a2s) is simulated, and although a real mode such as this does not exist, data concurrency is a probable event when the number of modules or messages is large. In this mode, all the modules send data at one time, as shown in Fig. 8(c). With the system thread pool, the concurrent mechanism is automatically improved. Both ROS and Project Cocktail achieved almost the same performance.

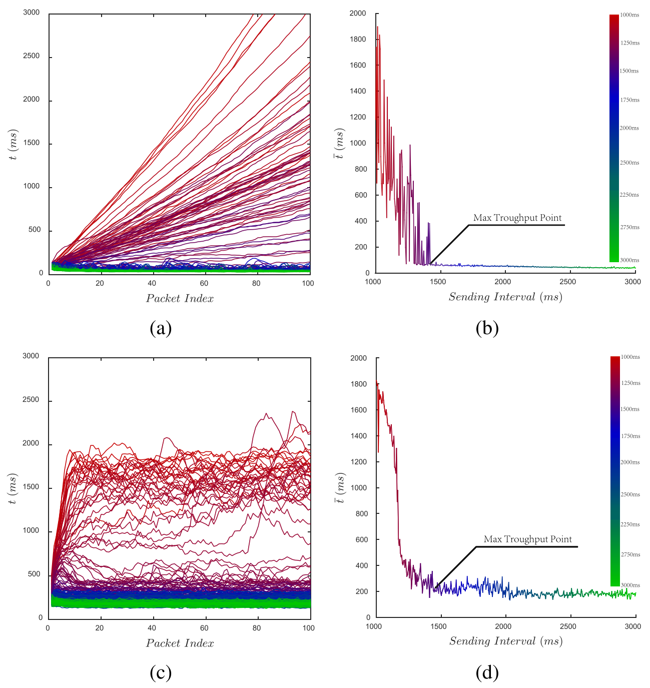


FIGURE 10. Throughput test: (a) Trend of transmission delay with different sending intervals. (b) Relationship between average delay and sending interval. (a) Project Cocktail. (b) Project Cocktail. (c) ROS. (d) ROS.

VI. AUTONOMOUS DRIVING TEST REPORT

A. OVERALL INTRODUCTION

In this section, a short test report about one journey around Jiading Campus, Tongji University, will be provided, which is approximately 23 km in total, with 8 km on campus, 10 km on the highway and 5 km in town. The weather was cloudy and with appropriate light and adequate conspicuity. The top speed was limited to 80 kph. The driving task includes lane keeping and changing, overtaking, traffic lights, crossing, U-turn, bridge, obstacle avoidance, semi-structured road driving and off-road driving. During the test, the vehicle was mainly equipped with 4 IDS UI-5240CP cameras, one Ibeo LUX, one Delphi ESR, two Delphi RSDs, and one GPS and inertial sensor. The processor is an IPC, equipped with an Intel i7 CPU, 8 GB of RAM and without a GPU accelerator. Considering the quite long distance, the data and analysis given below will be limited to the 8 km campus part. The satellite map of the testing routine with the detailed campus SLAM map used for localization is shown in Fig. 11.

B. MAIN PERCEPTION RESULTS

To achieve the goal of autonomous driving, some necessary perception modules are applied to the system, such as obstacle detection, lane detection, traffic light recognition, vehicle and pedestrian detection, and monocular-vision-based drivable region recognition, as well as a data fusion module. In this subsection, these perception results will be demonstrated, with some derived from existing algorithms and others from our unpublished results. The evaluation indices

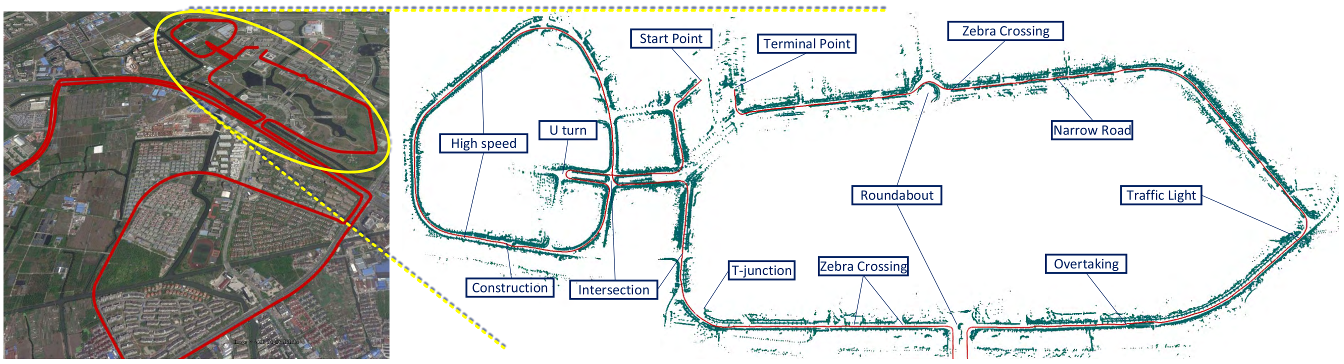


FIGURE 11. The red line in the left image is the 23 km test routine shown on Google Maps. The right image is the enlarged high-precision driving map of the campus part with GPS data, Ibeo data, lane data, road data, traffic sign data, and so forth. When autonomous driving, these information are the input priori knowledge for the vehicle planning and decision module.

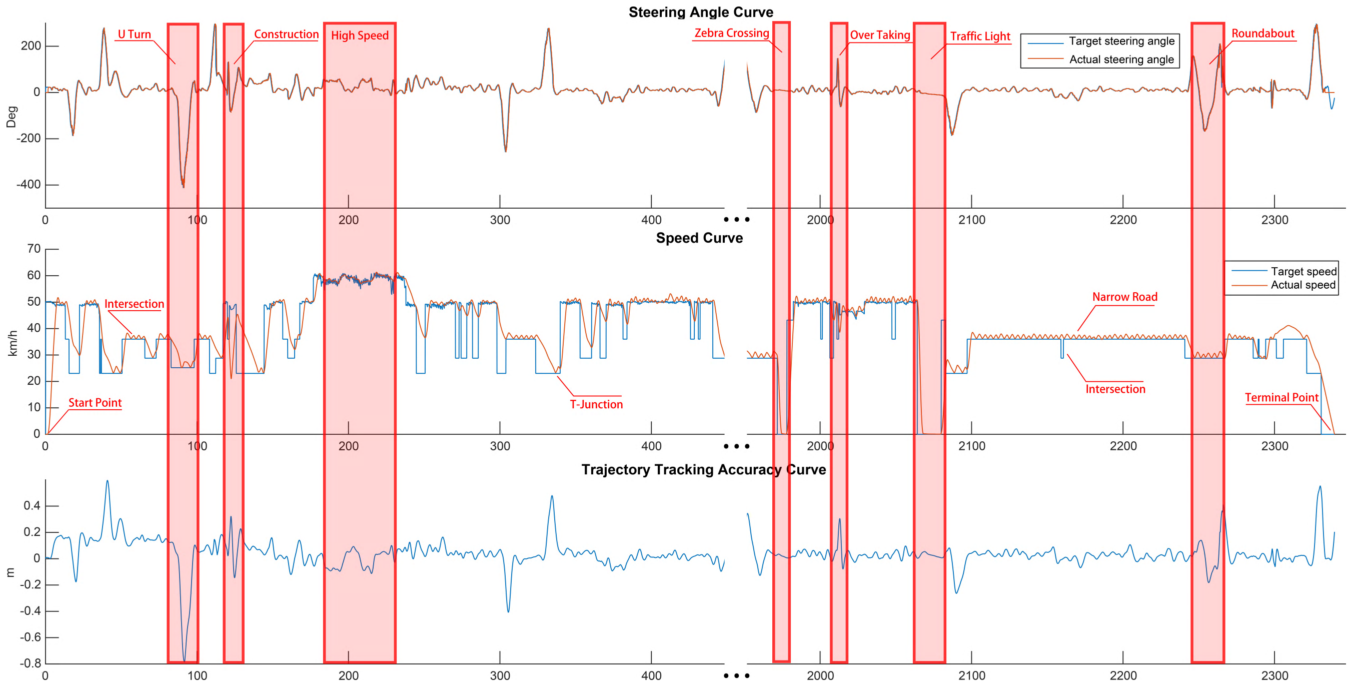


FIGURE 12. Key performance index curve with some typical scenes highlighted by red rectangles and labeled by blue words. From top to bottom: steering curve, speed curve and trajectory tracking curve.

in Table 2 are all based on object level, not pixel level, except for drivable area detection, which means that if the detection box area is truly the target object that needs to be detected, it is treated as a true positive (TP). If there is no target object that we need in the detection box, then it is a false positive (FP). If the target object does not have a box around it, then it is counted as a false negative (FN).

1) LANE DETECTION

As the most traditional and necessary perception unit, it has been strenuously studied for decades [42]–[45]. In this driving test, a famous early algorithm named GOLD [45] from Vislab is selected, which can handle most of the cases, such as solid and dashed, strong shadow and zebra crossing, among others. However, as a monocular algorithm without 3D information, it is difficult for this algorithm to distinguish white light poles from lanes. Fortunately, through the data fusion module, with the help of the SLAM map and other sensors, these false positives can be successfully removed. The single frame detection result is shown in Fig. 13 and Table 2.

2) VEHICLE AND PEDESTRIAN DETECTION

Considering the balance of stability, accuracy and operational time, DPM [46] is the most suitable for detecting vehicles and pedestrians. Although the accuracy is less than that with deep learning [47]–[49], the calculation cost is suitable for our processor without GPU acceleration. Compared with driver assist systems, fully autonomous driving cannot tolerate

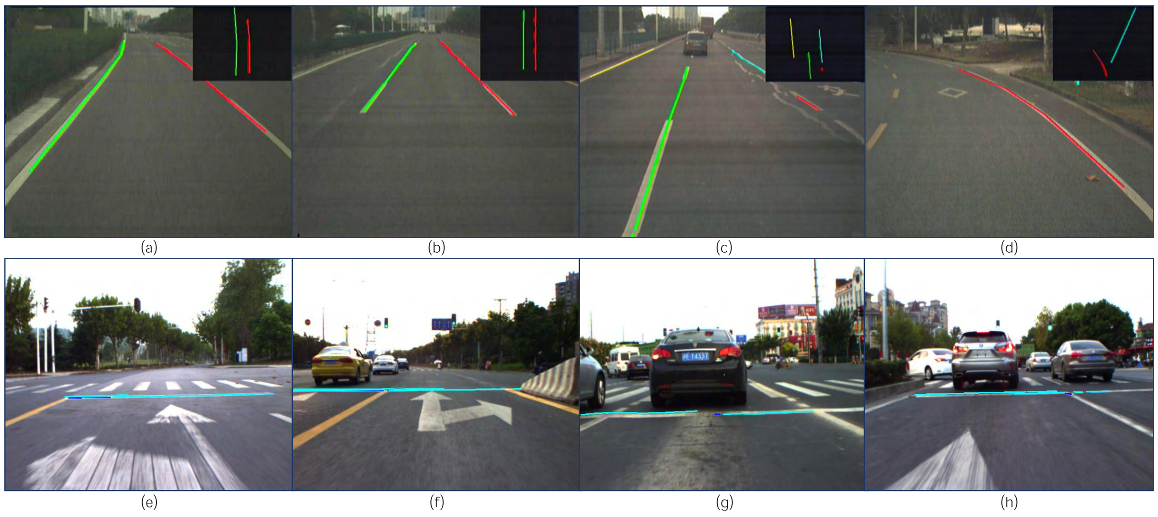


FIGURE 13. Lane detection results. (a-c) The normal lane detection results on both campus and highway roads with different colors representing different lanes. (d) False positive caused by light pole. (e-h) The stop lanes are represented by blue lines.

TABLE 2. Perception performance indices.

Lane Detection					
Ego Lane			Side Lane		
Detection rate (TP)	Error rate (FP)	Miss rate (FN)	Detection rate (TP)	Error rate (FP)	Miss rate (FN)
98.33%	1.12%	1.67%	65.45%	2.34%	34.55%

Stop Lane Detection			
Detection rate (TP)	Error rate (FP)	Miss rate (FN)	Distance accuracy (m)
95.12%	7.87%	4.88%	±0.16

Traffic Light Detection					
50 meter			100 meter		
Detection rate (TP)	Error rate (FP)	Miss rate (FN)	Detection rate (TP)	Error rate (FP)	Miss rate (FN)
94.20%	1.19%	5.80%	81.57%	3.73%	18.43%

Vehicle Detection		
Detection rate (TP)	Error rate (FP)	Miss rate (FN)
85.98%	5.11%	14.02%

Pedestrian Detection		
Detection rate (TP)	Error rate (FP)	Miss rate (FN)
77.26%	7.03%	22.74%

Drivable Area Detection						
MaxF	AP	PRE	REC	FPR	FNR	Runtime
81.41%	82.12%	72.65%	82.18%	12.62%	21.22%	0.015s

mistakes such as false negatives within a certain distance, which will be fatal during driving. Consequently, through adjusting the parameters, we decrease the false negatives to the greatest extent possible. With the help of the data fusion module, most of the false positives can be removed.

3) TRAFFIC LIGHT DETECTION

The detection result shown in Fig. 15 is derived from an improved version of [50], mainly by adding a verification part to remove false positives and improving arrow light recognition.

4) VISION-BASED DRIVABLE AREA DETECTION

The result shown in Fig. 16 is the monocular-vision-based drivable region recognition algorithm, which will be presented in our future papers. In contrast to the deep learning methods [47], [51], the processing is real time without any GPU. Irrespective of whether the road is structured or ill-structured, curb based or non-curb based, the algorithm extracts optimal regions as road. The green region represents drivable areas where vehicles, pedestrians and other obstacles are eliminated. Similarly, this recognition result is also input into a real-time mapping module to derive a noiseless and

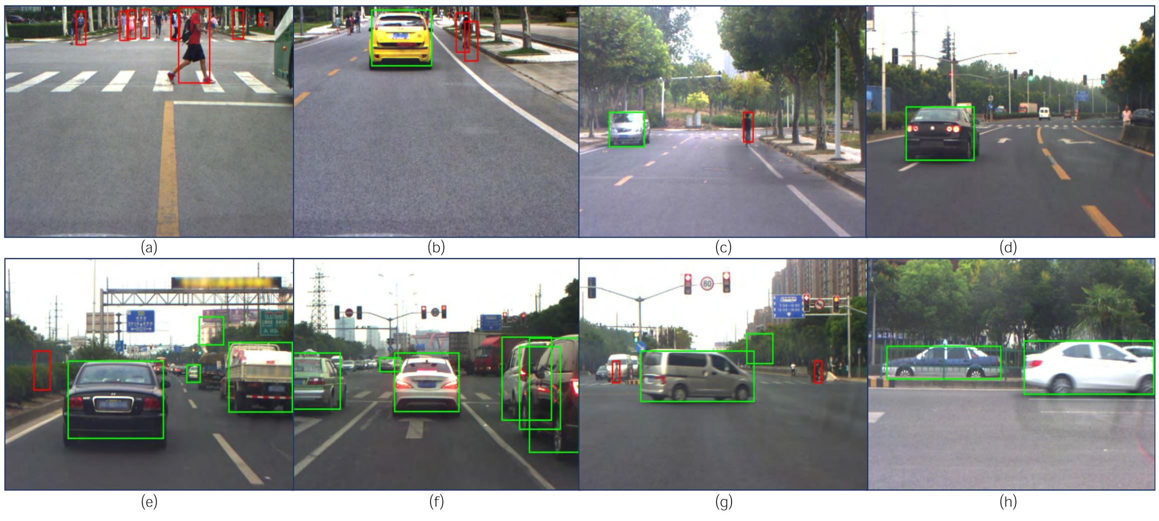


FIGURE 14. Vehicle and pedestrian detection results. (a-b) The detection results from the front camera for nearby pedestrians and vehicles. (c-h) The detection results from in-car cameras. Irrespective of the vehicle poses, the algorithm can handle it, and considering the safety, there are some false positives.

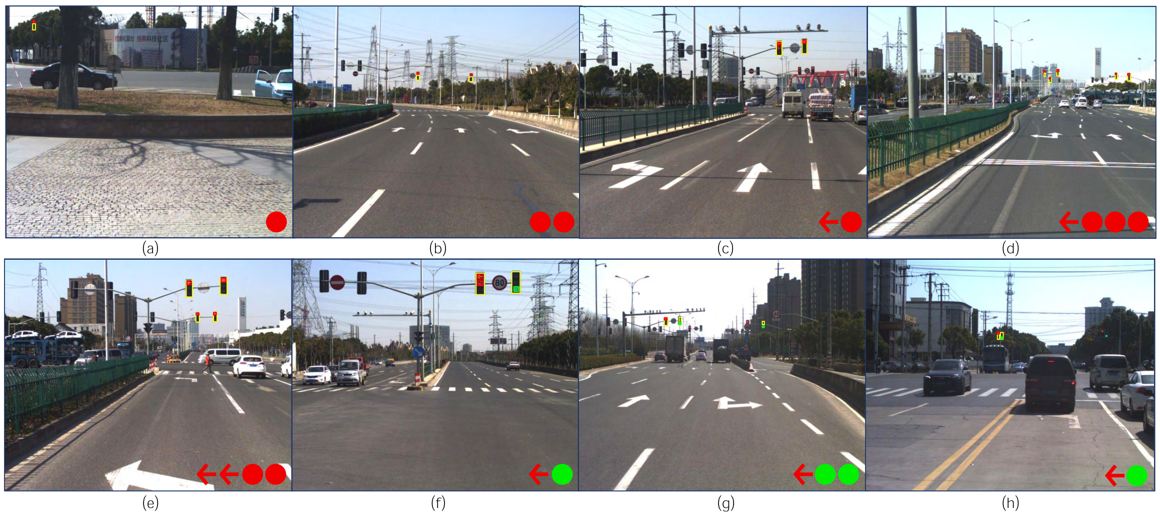


FIGURE 15. Traffic light detection results. (a-b) Traffic lights with complex background. (c-h) Detection results under different traffic scenes.

non-blind zone perception, demonstrated by a green area. With the recognition of traffic lights and stop lanes, the drivable region will increase or decrease, which finally forms the environment perception result in the data fusion module. The quantitative index to analyze the algorithm is chosen from [52].

5) DATA FUSION MODULE

In Fig. 17, the dark blue points are the current scan results from one Ibeo LUX placed in front of the vehicle, and the light blue points are the estimated results from the past few frames introduced in Section III in detail. It is clear that the algorithm expands Ibeo data and fills up blind zones. The red points are lane detection results from the current and past few frames, which fit into the red line by the probability-based least squares module to be the final lane result. This fusion result can actually remove most of the noise caused by a single frame.

C. KEY PERFORMANCE INDEX CURVES

The data plotted in Fig. 12 are the control performance index curves of steering wheel, speed and tracking accuracy of the campus part because of the large amount of data and complexity of the open road condition. (a) Steering wheel control is fast and accurate. Unless in the circumstance of U-turn obstacle avoidance and overtaking, the steering wheel changes smoothly to ensure an optimal lateral control. (b) Although the expected speed is given as a step signal, the controller of the accelerator pedal will guarantee that the acceleration, even with jerks, changes continuously. Unfortunately, due to the

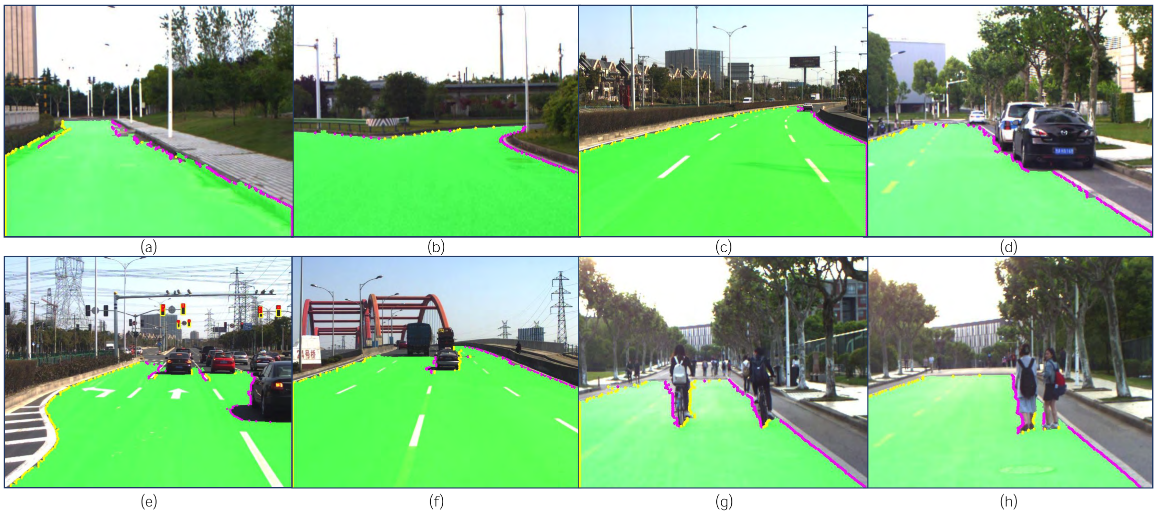


FIGURE 16. Drivable area detection results represented by green color. (a) Unmarked straight road detection on campus. (b) Unmarked branched road detection on campus. (c) Marked road detection on highway. (d) Marked road detection with parked cars on campus. (e) Marked road detection with moving vehicles on highway. (f) Marked road detection with guide lines and moving vehicles on highway. (g-h) Marked road detection with cyclists and pedestrians on campus.

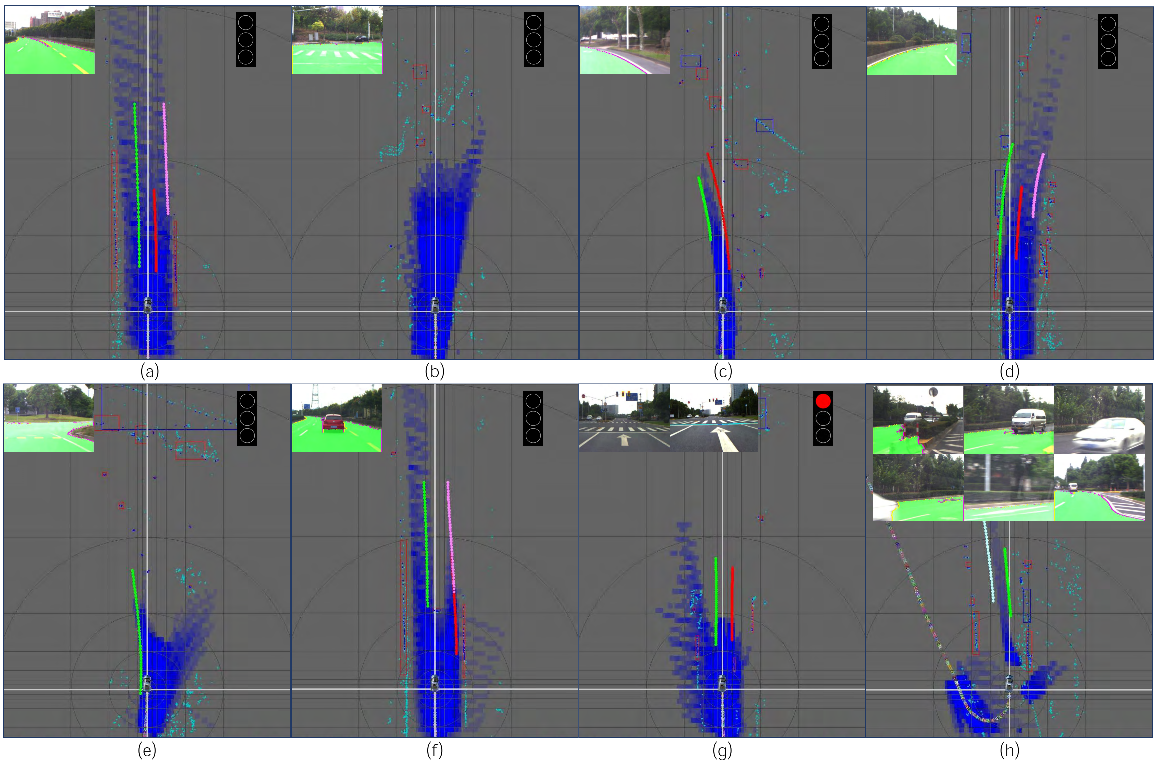


FIGURE 17. Real-time mapping and data fusion results. Light blue points represent Ibeo mapping points. Colored lines represent lane detection results with mapping system. Traffic detection results are shown on the top right if available. The blue area is the drivable area, which is represented by a lateral 0.5 m and longitudinal 1.0 m grid map. The brighter the color is, the greater the probability that the area is trusted as a road area. (a) Marked highway road. (b) T-junction road. (c-d) Left and right turns. (e) Branched road. (f) Road with a moving front car to shorten the drivable area. (g) Crossing road with red light. (h) U-turn point with moving vehicles and guide lines.

mechanical deviation of our autopilot, the real-time speed has an approximate 1 kph control deviation, particularly under the condition of even-speed driving, which can be avoided with a controlled-by-wire chassis. (c) In our architecture, the control module receives trajectories from the upper level. Thanks to the local planning loop mentioned in Section III (B), the real-time trajectories are continuous and meet the vehicle dynamics model. The tracking accuracy is mostly less than 10 centimeters and no more than 20 centimeters during the driving test.

VII. CONCLUSION AND FUTURE WORK

After five years of overall testing of Tongji's autonomous vehicle, the system is proven to be stable and efficient. Although low-cost sensors are our interest, the hardware system appears to be adequate for daily autonomous driving on structured and half-structured roads. In particular, compared with ROS, the software performance is satisfactory. In addition, part of the system has been tested on Roewe E50 of SAIC Motor, which is one of the largest automotive manufacturers in China. However, there is still much work to do in the near future. The test cases are all performed under the $\times 86$ architecture, which does not appear to be the best choice for a vehicle platform. Consequently, considerable emphasis has recently been placed on embedded conversion of the system. Recently, end-to-end deep learning autonomous driving algorithms have been developed by NVIDIA and Comma.ai, which may be the new trend in the near future. Therefore, considerable emphasis will be placed on this topic to make the vehicle drive more like humans drive.

ACKNOWLEDGMENT

The authors would like to thank the Associate Editor and anonymous reviewers for their constructive comments that have improved the presentation of this paper.

REFERENCES

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

[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, 2008.

[3] J. Leonard *et al.*, "A perception-driven autonomous urban vehicle," *J. Field Robot.*, vol. 25, no. 10, pp. 727–774, 2008.

[4] I. Miller *et al.*, "Team Cornell's Skynet: Robust perception and planning in an urban environment," *J. Field Robot.*, vol. 25, no. 8, pp. 493–527, 2008.

[5] A. Bacha *et al.*, "Odin: Team VictorTango's entry in the DARPA urban challenge," *J. Field Robot.*, vol. 25, no. 8, pp. 467–492, 2008.

[6] J. Xin, C. Wang, Z. Zhang, and N. Zheng, "China future challenge: Beyond the intelligent vehicle," *IEEE Intell. Transp. Syst. Soc. Newslett.*, vol. 16, no. 2, pp. 8–10, Apr. 2014.

[7] L. Xu, Y. Wang, H. Sun, J. Xin, and N. Zheng, "Design and implementation of driving control system for autonomous vehicle," in *Proc. IEEE 17th Int. Conf. Intell. Transp. Syst. (ITSC)*, Oct. 2014, pp. 22–28.

[8] L. Xu, Y. Wang, H. Sun, J. Xin, and N. Zheng, "Integrated longitudinal and lateral control for Kuafu-II autonomous vehicle," *IEEE Trans. Intell. Transp. Syst.*, vol. 17, no. 7, pp. 2032–2041, Jul. 2016.

[9] T. Xia, M. Yang, R. Yang, and C. Wang, "CyberC3: A prototype cybernetic transportation system for urban applications," *IEEE Trans. Intell. Transp. Syst.*, vol. 11, no. 1, pp. 142–152, Mar. 2010.

[10] W. Yao, Z. Deng, and L. Zhou, "Road curb detection using 3D lidar and integral laser points for intelligent vehicles," in *Proc. Joint 6th Int. Conf. Soft Comput. Intell. Syst. (SCIS), 13th Int. Symp. Adv. Intell. Syst. (ISIS)*, Nov. 2012, pp. 100–105.

[11] D. Liu, X. An, Z. Sun, and H. He, "Active safety in autonomous land vehicle," in *Proc. Workshop Power Electron. Intell. Transp. Syst. (PEITS)*, Aug. 2008, pp. 476–480.

[12] X. Li, Z. Sun, D. Cao, D. Liu, and H. He, "Development of a new integrated local trajectory planning and tracking control framework for autonomous ground vehicles," *Mech. Syst. Signal Process.*, vol. 87, pp. 118–137, Mar. 2017.

[13] J. Ziegler *et al.*, "Making Bertha drive—An autonomous journey on a historic route," *IEEE Intell. Transp. Syst. Mag.*, vol. 6, no. 2, pp. 8–20, Apr. 2014.

[14] J. Levinson *et al.*, "Towards fully autonomous driving: Systems and algorithms," in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Jun. 2011, pp. 163–168.

[15] C. Berger and B. Rumpe. (2014). "Autonomous driving-5 years after the urban challenge: The anticipatory vehicle as a cyber-physical system." [Online]. Available: https://arxiv.org/abs/1409.0413

[16] J. Kim, R. R. Rajkumar, and M. Jochim, "Towards dependable autonomous driving vehicles: A system-level approach," *ACM SIGBED Rev.*, vol. 10, no. 1, pp. 29–32, 2013.

[17] J. Wei, J. M. Snider, J. Kim, J. M. Dolan, R. Rajkumar, and B. Litkouhi, "Towards a viable autonomous driving research platform," in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Jun. 2013, pp. 763–770.

[18] M. R. Endsley, "Autonomous driving systems: A preliminary naturalistic study of the tesla model s," *J. Cognit. Eng. Decision Making*, vol. 11, no. 3, pp. 225–238, 2017.

[19] P. Lin, W. Ren, and H. Gao, "Distributed velocity-constrained consensus of discrete-time multi-agent systems with nonconvex constraints, switching topologies, and delays," *IEEE Trans. Autom. Control*, vol. 62, no. 11, pp. 5788–5794, Nov. 2017.

[20] J. Qin, C. Yu, and H. Gao, "Collective behavior for group of generic linear agents interacting under arbitrary network topology," *IEEE Trans. Control Netw. Syst.*, vol. 2, no. 3, pp. 288–297, Sep. 2015.

[21] R.-H. Zhang, Z.-C. He, H.-W. Wang, F. You, and K.-N. Li, "Study on self-tuning tyre friction control for developing main-servo loop integrated chassis control system," *IEEE Access*, vol. 5, pp. 6649–6660, 2017.

[22] Z. He, L. Zheng, L. Song, and N. Zhu, "A jam-absorption driving strategy for mitigating traffic oscillations," *IEEE Trans. Intell. Transp. Syst.*, vol. 18, no. 4, pp. 802–813, Apr. 2017.

[23] L. Lu, J. Wang, Z. He, and C.-Y. Chan, "Real-time estimation of freeway travel time with recurrent congestion based on sparse detector data," *IET Intell. Transp. Syst.*, vol. 12, no. 1, pp. 2–11, Feb. 2018.

[24] Z. He, L. Zheng, L. Lu, and W. Guan, "Erasing lane changes from roads: A design of future road intersections," *IEEE Trans. Intell. Veh.*, to be published.

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

[26] W. Xu, H. A. Omar, W. Zhuang, and X. S. Shen, "Delay analysis of in-vehicle Internet access via on-road WiFi access points," *IEEE Access*, vol. 5, pp. 2736–2746, 2017.

[27] M. R. Jabbarpour, H. Zarrabi, J. J. Jung, and P. Kim, "A green ant-based method for path planning of unmanned ground vehicles," *IEEE Access*, vol. 5, pp. 1820–1832, 2017.

[28] Y. He, D. Sun, M. Zhao, and S. Cheng, "Cooperative driving and lane changing modeling for connected vehicles in the vicinity of traffic signals: A cyber-physical perspective," *IEEE Access*, vol. 6, pp. 13891–13897, 2018.

[29] X. Li, S. Wu, J. Han, and W. Wang, "Fast location algorithm based on an extended symmetry nested sensor model in an intelligent transportation system," *IEEE Access*, to be published.

[30] D. Drake, S. Koziol, and E. Chabot, "Mobile robot path planning with a moving goal," *IEEE Access*, vol. 6, pp. 12800–12814, 2018.

[31] P. Friudenberg and S. Koziol, "Mobile robot rendezvous using potential fields combined with parallel navigation," *IEEE Access*, vol. 6, pp. 16948–16957, 2018.

[32] S. Behere and M. Törngren, "A functional architecture for autonomous driving," in *Proc. 1st Int. Workshop Autom. Softw. Archit.*, 2015, pp. 3–10.

[33] S. Liu, J. Tang, C. Wang, Q. Wang, and J.-L. Gaudiot, "A unified cloud platform for autonomous driving," *Computer*, vol. 50, no. 12, pp. 42–49, 2017.

[34] K. Jo, J. Kim, D. 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.

[35] K. Jo, J. Kim, D. Kim, C. Jang, and M. Sunwoo, "Development of autonomous car—Part II: A case study on the implementation of an autonomous driving system based on distributed architecture," *IEEE Trans. Ind. Electron.*, vol. 62, no. 8, pp. 5119–5132, Dec. 2015.

[36] C. Berger. (2014). "From a competition for self-driving miniature cars to a standardized experimental platform: Concept, models, architecture, and evaluation." [Online]. Available: https://arxiv.org/abs/1406.7768

[37] M. Quigley *et al.*, "ROS: An open-source robot operating system," in *Proc. ICRA Workshop Open Source Softw.*, 2009, vol. 3. nos. 3–2, p. 5.

[38] E. Santana and G. Hotz. (2016). "Learning a driving simulator." [Online]. Available: https://arxiv.org/abs/1608.01230

[39] A. Belbachir, J.-C. Smal, and J.-M. Blosseville, "A robotic platform to evalute autonomous driving systems," in *Proc. 15th Int. IEEE Conf. Intell. Transp. Syst. (ITSC)*, Sep. 2012, pp. 1874–1879.
[40] X. Li, Z. Sun, D. Cao, Z. He, and Q. Zhu, "Real-time trajectory planning for autonomous urban driving: Framework, algorithms, and verifications," *IEEE/ASME Trans. Mechatronics*, vol. 21, no. 2, pp. 740–753, Apr. 2016.
[41] S. E. Li, F. Gao, D. Cao, and K. Li, "Multiple-model switching control of vehicle longitudinal dynamics for platoon-level automation," *IEEE Trans. Veh. Technol.*, vol. 65, no. 6, pp. 4480–4492, Jun. 2016.
[42] Y. Wang, E. K. Teoh, and D. Shen, "Lane detection and tracking using B-snake," *Image Vis. Comput.*, vol. 22, no. 4, pp. 269–280, Apr. 2004.
[43] Q. Li, N. Zheng, and H. Cheng, "Springrobot: A prototype autonomous vehicle and its algorithms for lane detection," *IEEE Trans. Intell. Transp. Syst.*, vol. 5, no. 4, pp. 300–308, Dec. 2004.
[44] Z. Kim, "Robust lane detection and tracking in challenging scenarios," *IEEE Trans. Intell. Transp. Syst.*, vol. 9, no. 1, pp. 16–26, Mar. 2008.
[45] M. Bertozzi *et al.*, "GOLD: A framework for developing intelligent-vehicle vision applications," *IEEE Intell. Syst.*, vol. 23, no. 1, pp. 69–71, Jan./Feb. 2008.
[46] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 9, pp. 1627–1645, Sep. 2010.
[47] C. Farabet, C. Couprie, L. Najman, and Y. LeCun, "Learning hierarchical features for scene labeling," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 8, pp. 1915–1929, Aug. 2013.
[48] Z. Cai, Q. Fan, R. S. Feris, and N. Vasconcelos, "A unified multi-scale deep convolutional neural network for fast object detection," in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 354–370.
[49] Y. Xiang, W. Choi, Y. Lin, and S. Savarese, "Subcategory-aware convolutional neural networks for object proposals and detection," in *Proc. IEEE Winter Conf. Appl. Comput. Vis. (WACV)*, Mar. 2017, pp. 924–933.
[50] W. Zong and Q. Chen, "Traffic light detection based on multi-feature segmentation and online selecting scheme," in *Proc. IEEE Int. Conf. Syst., Man (SMC)*, Oct. 2014, pp. 204–209.
[51] R. Mohan. (2014). "Deep deconvolutional networks for scene parsing." [Online]. Available: https://arxiv.org/abs/1411.4101
[52] J. Fritsch, T. Kuhnl, and A. Geiger, "A new performance measure and evaluation benchmark for road detection algorithms," in *Proc. 16th Int. IEEE Conf. Intell. Transp. Syst. (ITSC)*, Oct. 2013, pp. 1693–1700.

WENHAO ZONG received the B.S. degree in automation from Tongji University, Shanghai, China, where he is currently pursuing the Ph.D. degree in control theory and control engineering. He has been the Team Leader of the Tongji Autonomous Vehicle Group for six years. His current research interests include intelligent system architecture, computer vision, and pattern recognition.

CHANGZHU ZHANG received the Ph.D. degree in mechatronics engineering from the City University of Hong Kong, Hong Kong, in 2012. From 2013 to 2014, he was an Associate Research Fellow with the Institute for Advanced Study, Tongji University, Shanghai, China, where he is currently an Associate Professor with the School of Informatics and Electronics Engineering. His current research interests include intelligent control, networked control systems, signal processing, and vehicle control.

ZHUPING WANG received the B.Eng. and M.Eng. degrees from the Department of Automatic Control, Northwestern Polytechnic University, China, in 1994 and 1997, respectively, and the Ph.D. degree from the National University of Singapore, Singapore, in 2003. She is currently a Professor with the College of Electronics and Information Engineering, Tongji University, Shanghai, China. Her research interests include the intelligent control of robotic systems and nonholonomic control systems.

JIN ZHU received the B.S. and master's degree in automation from Xi'an Jiaotong University, Shanxi, China, in 1982 and 1984, respectively. He is currently an Associate Professor with the College of Electronics and Information Engineering, Tongji University, Shanghai, China. His research interests include control theory and robotic system.

QIJUN CHEN (M'14–SM'14) received the B.S. degree from the Huazhong University of Science and Technology, Wuhan, China, in 1987, the M.S. degree from Xi'an Jiaotong University, Xi'an, China, in 1990, and the Ph.D. degree from Tongji University, Shanghai, China, in 1999. He was a Guest Professor with the University of Hagen, Hagen, Germany, in 2002, and a Visiting Professor with the University of California at Berkeley, Berkeley, CA, USA, in 2008. He is currently a Full Professor with the College of Electronics and Information Engineering, Tongji University. He has published over 100 papers in journals and conference proceedings. His research interests include robotics control, the environmental perception and understanding of mobile robots, and bio-inspired control.