Research Article

A model-based design of an onboard stereo vision system: obstacle motion estimation for cooperative automated vehicles

Narsimlu Kemsaram[1] · **Anweshan Das**[1] · **Gijs Dubbelman**[1,2]

Received: 8 September 2021 / Accepted: 30 May 2022
Published online: 27 June 2022

Abstract

Cooperative automated vehicle software needs to use advanced software engineering design methodologies to manage the complexity of distributed software. This article presents a design model of a stereo vision system for cooperative automated vehicles based on object-oriented analysis and design methodologies and unified modeling language concepts. We use a rational unified process and the 4+1 architectural view model to design the stereo vision system. The designed model is a part of the perception system of the ego vehicle. The stereo vision system continuously perceives the traffic environment using an on-board front-facing stereo camera and sends information to the vehicle control system. The stereo vision system is developed on Ubuntu 16.04 LTS platform with the Nvidia DriveWorks framework using the C++ programming language, and it is deployed on Nvidia Drive PX2 automotive-grade embedded hardware platform. The designed and developed system is evaluated on the Carla simulation environment and the Renault Twizy cooperative automated vehicle research platform. The experimental results show that the approach is applicable, and the system is capable of running in real-time.

Keywords 4+1 architectural view model · Cooperative automated vehicle · Object-oriented analysis and design · Rational unified process · Stereo vision system · Unified modeling language

1 Introduction

Over the past several years, there is a rapid development in the field of automated vehicles to minimize traffic congestion, reduce fuel emission, and avoid human error in road accidents [1]. One of the potential solutions to these problems is platoon based cooperative automated vehicles, where multiple vehicles drive together in formation [2]. Vehicles work together in forming these platoons, coordinating with each other using a Vehicle-to-Vehicle (V2V) communication channel [3]. In such platooning of vehicles, the front lead vehicle is controlled by the driver, while the second or other vehicles behind it autonomously follow the vehicle in front of it [4, 5]. For this challenging task, the integrated Cooperative Automated Vehicles (i-CAVE) research project team has set up a Renault-Twizy platoon vehicle pair [6], consisting of one lead vehicle and one ego vehicle, is as shown in Fig. 1.

The i-CAVE is a research project that addresses the challenges of the design and development of a platoon vehicle platform [7]. In this platoon vehicle platform, all independent Electronic Control Units (ECUs) are deployed on vehicles to enforce digital control of functional aspects such as vision, radar, steering, throttle, and brakes [8]. Each

Narsimlu Kemsaram, Anweshan Das and GijsDubbelman these authors have contributed equally to this work.

✉ Narsimlu Kemsaram, n.kemsaram@tue.nl; Anweshan Das, anweshan.das@tue.nl; Gijs Dubbelman, g.dubbelman@tue.nl; gijs@aiim.ai | [1]Department of Electrical Engineering, Mobile Perception Systems Lab, Signal Processing Systems Group, Eindhoven University of Technology (TU/e), 5612 AZ Eindhoven, The Netherlands. [2]AI in Motion (AIIM), 5657 EB Eindhoven, The Netherlands.

Fig. 1 i-CAVE platooning with Renault Twizy vehicles

ECU requires interactions and communications across several ECUs. For example, the vision needs to send the perceived traffic obstacles information to the vehicle control system. To increase ECU reuse across systems, we followed the standardized interfaces such as a Controller Area Network (CAN) bus that ease communication and deployment between distributed ECUs. Since the number of independent ECUs grows in the platoon vehicle platform, there is a need to use advanced software engineering methodologies and design concepts to manage software complexity [9]. Moreover, many errors come from requirements or design flaws [10–12]. To avoid the requirement errors or design flaws, we need a good software architecture and design model before we can develop a system in real life. A good software architecture and design model ensures the system functional and non-functional requirements before developing software [13–15].

This article describes the architecture of the automated vehicle software and explains how the i-CAVE research team adopted it into the integrated cooperative automated vehicle software architecture. The main goal is to design a stereo vision system using a combination of the rational unified process and the 4+1 architectural view model. The stereo vision system is an independent ECU, which perceives the traffic environment with as much clarity as possible along with real-time processing capability. It is easy to interface with existing ECUs in cooperative automated vehicle software architecture.

The main contributions of this article are:

- In this paper we presented a designed model of an onboard stereo vision system for cooperative automated vehicles. We extend our previous work [16, 17] with a complete design of an onboard stereo vision system, explaining the design methodology and presenting all design views with their corresponding implemented modules.
- We evaluated the performance of the proposed onboard stereo vision system in a simulation environment and real-world scenarios.

This article is organized as follows: Section 2 gives an overview of the cooperative automated vehicle architecture. Section 3 explains the proposed onboard stereo vision system. Section 4 analyzes the stereo vision system requirements. Section 5 elaborates on the stereo vision system design model. Section 6 provides the auto code generation and development of the stereo vision system. Section 7 presents the experimental setup. Section 8 presents the experimental results. A discussion on the proposed stereo vision system is given in Section 9. Finally, Section 10 presents our conclusions.

2 Architecture of cooperative automated vehicles

In this section, we provide a brief description of the automated vehicle functional architecture and integrated cooperative automated vehicle functional architecture.

2.1 Automated vehicle functional architecture

The i-CAVE research project uses the multi-layered functional architecture, as proposed in [18, 19]. This architecture has multiple layers through which information flows from the left to right, from sensors to actuators, decisions at each layer level. The various layers are as follows [20]: (i) *Interface in layer* reads input from sensors (such as Stereo Camera, Radar (RAdio Detection And Ranging), GPS (Global Positioning System), IMU (Inertial Measurement Unit), V2V in) and converts the sensor input into data that can be used in the next layer, (ii) *Sensor layer* combines data from different sources into information about the vehicle and its surroundings, (iii) *Control layer* generates the appropriate autonomous signal (autonomous/cooperative/service) based on the state of the vehicle and its surroundings, (iv) *Actuator layer* selects the appropriate control signal source (such as steering, throttle, brake) for the different physical movements of the vehicle and (v) *Interface out layer* takes the input control signal from the previous layer and sends the output control signal to physical components (Vehicle, HMI (Human-Machine Interface), V2V out). The proposed stereo vision system is a part of the *Sensor layer*. Fig. 2 shows the layers and components that are used in an automated vehicle functional architecture. Refer to [18, 19], and [20], for more details.

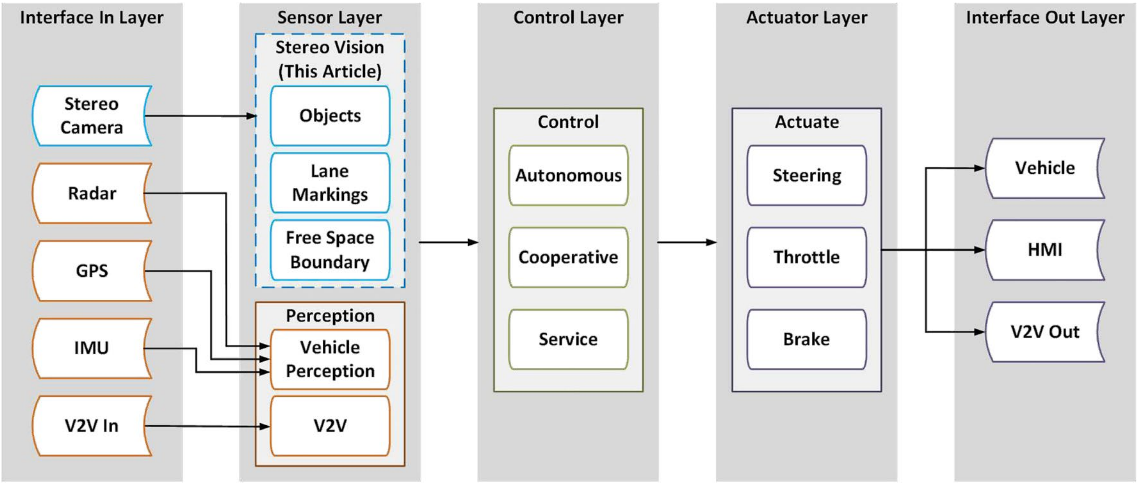


Fig. 2 Automated vehicle functional architecture

2.2 Cooperative automated vehicle functional architecture

In the vehicle platoon scenario, two or more vehicles form a vehicle train or platoon. The front-most vehicle is controlled by the driver, and all following vehicles autonomously follow the vehicle in front of them. Vehicles work together in forming these platoons, coordinating with each other using a V2V communication channel. The V2V communication channel is also used by vehicles to communicate information used to optimize the safe inter-vehicle distance within a platoon. Fig. 3 shows the functional components used when one vehicle wants to cooperate with other vehicles in a cooperative automated driving scenario. Refer to [20] for more details.

In the platoon based cooperative automated vehicle architecture, each vehicle contains the following functional components: (i) *Sense* component corresponds to the *interface in layer* and *sensor layer* of the automated vehicle functional architecture, (ii) *Control* component corresponds to *control layer* of the automated vehicle functional architecture, (iii) *Actuate* component corresponds to the *actuator layer* and the *interface out layer* of the automated vehicle functional architecture, (iv) *V2V in* component processes the incoming wireless messages of the vehicle, and (v) *V2V out* processes the outgoing wireless messages of the vehicle.

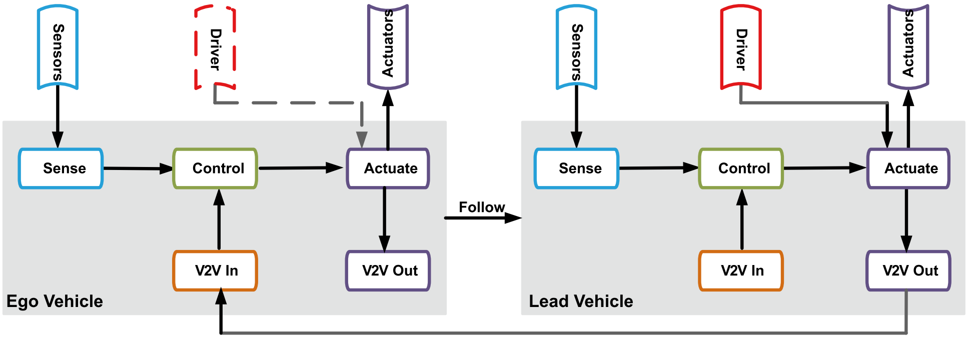
3 Proposed onboard stereo vision system for cooperative automated vehicles

In this section, we describe the proposed stereo vision system overview along with the system architecture.

3.1 System overview

The system being proposed is the stereo vision system and its system architecture, which is located in an on-board computer on the ego vehicle. The benefits of the stereo vision system in cooperative automated driving are: (i) perceives the traffic environment information in front of the vehicle, (ii) computes the obstacle information, and (iii) provides the obstacle information in real-time to the ego vehicle control system. The automated vehicle control system of the ego vehicle enables it to autonomously follow

Fig. 3 Cooperative automated vehicle functional architecture



the lead vehicle based on the perceived information. It visualizes the perceived traffic environment information in a meaningful way on the human-machine interface. This is accomplished through a series of modules that as explained as part of system architecture in Section 3.2. Fig. 4 shows the system overview of the proposed stereo vision system with a front-facing stereo camera and wireless communication in the platoon of vehicles.

3.2 System architecture

Raw data from the stereo camera is processed and then used to perceive the vehicle's surroundings. A diagram illustrating the system architecture of the proposed stereo vision system is shown in Fig. 5.

This process is mainly composed of the following modules, which allows a modular approach to the development of the stereo vision system:

3.2.1 Acquisition

This module acquires the left raw input image frame and the right raw input image frame simultaneously from the on-board stereo camera, loads the pre-calibrated camera rig configuration file, and processes the raw input image frame into the compatible image format of stereo vision, and obstacle motion estimation.

3.2.2 Stereo vision

This module removes the lens distortion of the input image frame and performs stereo rectification. The stereo rectification process transforms each image plane such that pairs of conjugate epipolar lines become collinear and parallel to the camera image plane on given a pair of stereo images. A similarity measure correlates matching pixels, and the disparity is the similarity distance between

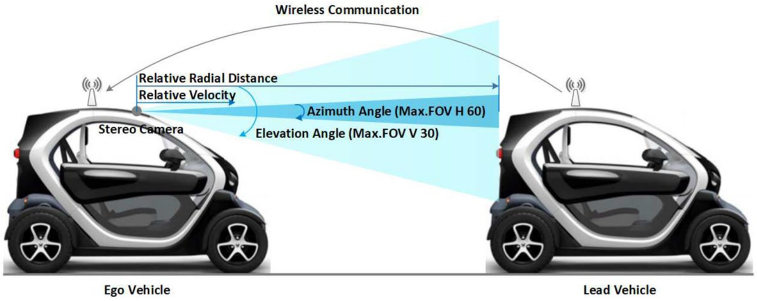


Fig. 4 System overview of the stereo vision system in vehicle platooning

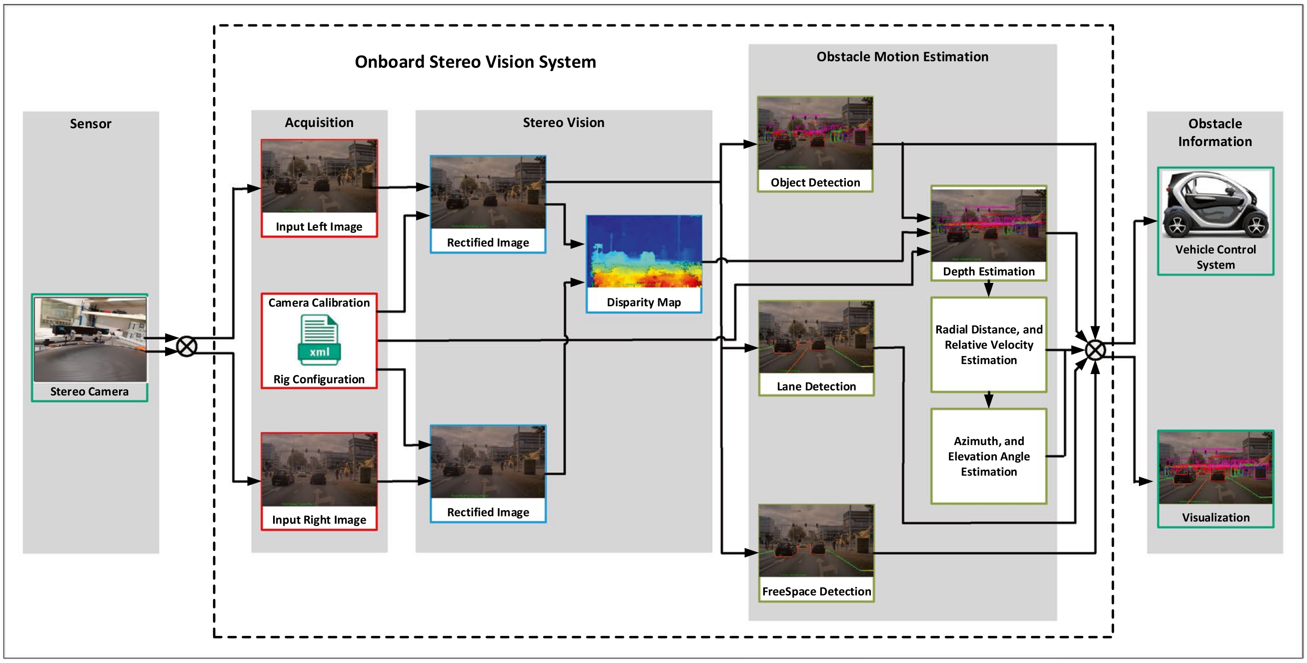


Fig. 5 System architecture of the stereo vision system

both points. The stereoscopic vision system uses a pair of cameras pointed at the same scene, and the differences between the images are used to extract the disparity map.

3.2.3 Obstacle motion estimation

This module performs object detection, depth estimation, radial distance estimation, relative velocity estimation, azimuth angle estimation, elevation angle estimation, lane detection, and free space detection to determine the vehicle's ability to move freely and safely inside the desired path:

- Object detection: The object detection algorithm detects, classifies, and tracks the different classes of objects such as cars, pedestrians, bicycles, traffic signs, and traffic lights.
- Depth estimation: The object depth estimation algorithm uses the object detection and depth-map-calculating algorithm to estimate the depth of each detected object.
- Radial distance estimation: The object distance estimation algorithm estimates the radial distance of each detected object by utilizing the computed object's depth and the corresponding object point in the left stereo camera image plane. Similarly, the radial distance estimation algorithm estimates a lead vehicle's relative radial distance with respect to the ego vehicle in a platooning scenario.
- Relative velocity: The velocity estimation algorithm estimates each detected object's velocity by utilizing the object's displacement from the previous position to the current position within the known time. Similarly, the velocity estimation algorithm estimates the relative velocity of a lead vehicle with respect to the ego vehicle in a platooning scenario.
- Azimuth and elevation angle estimation: The object azimuth and elevation angle estimation algorithm estimates each detected object's azimuth and elevation angle by utilizing the computed object's depth and radial distance. Similarly, the object azimuth and elevation angle estimation algorithm uses the lead vehicle's depth and horizontal radial distance to estimate the azimuth angle of a lead vehicle with respect to the ego vehicle in a platooning scenario. Also, it uses the lead vehicle's depth and vertical radial distance to estimate the elevation angle of a lead vehicle with respect to the ego vehicle in a platooning scenario.
- Lane detection: The lane detection algorithm recognizes the four different lane markings: left adjacent-lane, left ego-lane, right ego-lane, and right adjacent-lane when they are present on the road.
- Free space detection: The free space detection algorithm identifies the drivable collision-free space as a boundary across the road in front of the vehicle.

3.2.4 Obstacle information

This module sends the perceived obstacle information to the ego vehicle control system. The image rendering processing renders the output image frames in a meaningful way to visualize the user through the in-vehicle human-machine interface.

4 Requirements analysis

In this section, we provide research goals, system requirements, functional and non-functional requirements of the proposed stereo vision system. The stereo vision system is intended to be used within cooperative automated vehicles in a platoon driving environment. After starting the ego vehicle, the stereo vision system is activated. Its functionality provides consistent assistance to an ego vehicle under any ailment and within any road-based environment.

4.1 Research goals

The main goal is to design a stereo vision system for cooperative automated vehicles. Designing these systems is challenging as it requires advanced software engineering and design concepts to manage their complexity. We aim to reduce the gap between software engineering and cooperative automotive engineering concepts using model-based design methodologies.

4.2 System requirements

The main system requirements of the proposed stereo vision system are to perceive the traffic environment information in front of the ego vehicle and provide the processed traffic environment information to the automated vehicle control system of the ego vehicle. The automated vehicle control system of the ego vehicle can perform a command such that the ego vehicle autonomously follows the lead vehicle based on the perceived obstacle information. A display installed in the ego vehicle to provide visual feedback of the current stereo vision system functionality. These research goals and system requirements are accomplished by fulfilling various functional and non-functional requirements, are as explained below.

4.3 Functional requirements

The following are the proposed stereo vision system most relevant functional requirements: (i) *Operational*, the stereo vision system has to be functional whenever the driver activates the stereo vision system using the user interface, (ii) *Environmental perception*, the stereo vision system has to perceive the stationary and movable objects, such as the type of objects, the available traffic lanes, the drivable collision-free space boundary, including the dynamics of objects such as radial distance to objects, the relative velocity of objects, azimuth angle, and elevation angle of the objects, (iii) *Cooperation*, the stereo vision system has to support platooning by providing the perceived obstacle motion information to the ego vehicle control system.

4.4 Non-functional requirements

The following are the proposed stereo vision system most relevant non-functional requirements: (i) *Usable*, the stereo vision system has to be interoperable, reusable, and easy to integrate with other subsystems, (ii) *Reliable*, the stereo vision system has to perform in real world scenarios, (iii) *Performance*, the stereo vision system must perform with real-time execution speed, (iv) *Supportable*, the stereo vision system has to support with modular design, well-defined interface, and easy to install in a variety of cooperative automated vehicles.

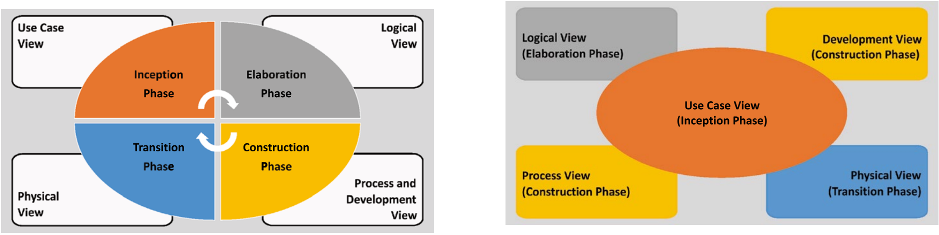
5 Model-based design of onboard stereo vision system

In this section, we describe the system design model of the stereo vision system in detail. In the software development life cycle process, the design model is an essential step before the development of software. The design models are good at checking requirements and planning efficient use of resources to achieve a successful product. The stereo vision system is designed based on the Object-Oriented Analysis and Design (OOAD) methodologies with the Unified Modeling Language (UML) concepts using the Sparx Systems Enterprise Architect tool. The UML is a language for visualizing, specifying, constructing, and documenting the artifacts of the object-oriented software-intensive system [21]. The software development life cycle process of the stereo vision system is based on a combination of the Rational Unified Process (RUP) and the 4+1 architectural view model [22], is as shown in Fig. 6. The software development life cycle process is carried out using the RUP into four phases: the inception phase, the elaboration phase, the construction phase, and the transition phase. Fig. 6a shows the phases in the RUP. In the inception phase, the focus is on requirements capture using the use case view. In the elaboration phase, the focus turns toward analysis and design using the logical view. In the construction phase, the focus is on implementation using the process view and development view. In the transition phase, the focus is on deployment using the physical view. The stereo vision system uses the iterative and incremental process using RUP throughout the software development life cycle. The 4+1 view model architecture designs the system from various perspectives, such as the use case view, the logical view, the process view, the development view, and the physical view. Fig. 6b shows the views in the 4+1 view software architecture model.

5.1 Use case view (scenarios)

The use case view includes the use cases that define the system's behavior and provide an outside view of the system [21]. With the UML, the static aspects of this use case view or scenario view are captured in the use case diagram, the dynamic aspects of this scenario view are captured in the interaction diagram, the statechart diagram, and the activity diagram. The stereo vision system has various research goals (see Section 4.1) and system requirements (see Section 4.2) that need to be satisfied to make this system successful. These goals and requirements can be translated into use cases and be seen in the use case diagram in Fig. 7.

The stereo vision system is an on-board GPU-based real-time embedded processing system. The primary function of the stereo vision system is to provide vision-based


Use Case View
Logical View
Inception Phase
Elaboration Phase
Transition Phase
Construction Phase
Physical View
Process and Development View
Logical View (Elaboration Phase)
Development View (Construction Phase)
Use Case View (Inception Phase)
Process View (Construction Phase)
Physical View (Transition Phase)

(a) RUP Phases.

(b) 4+1 Architectural Model Views.

Fig. 6 The software development life cycle process of the stereo vision system is based on a combination of the RUP and the 4+1 view architectural model

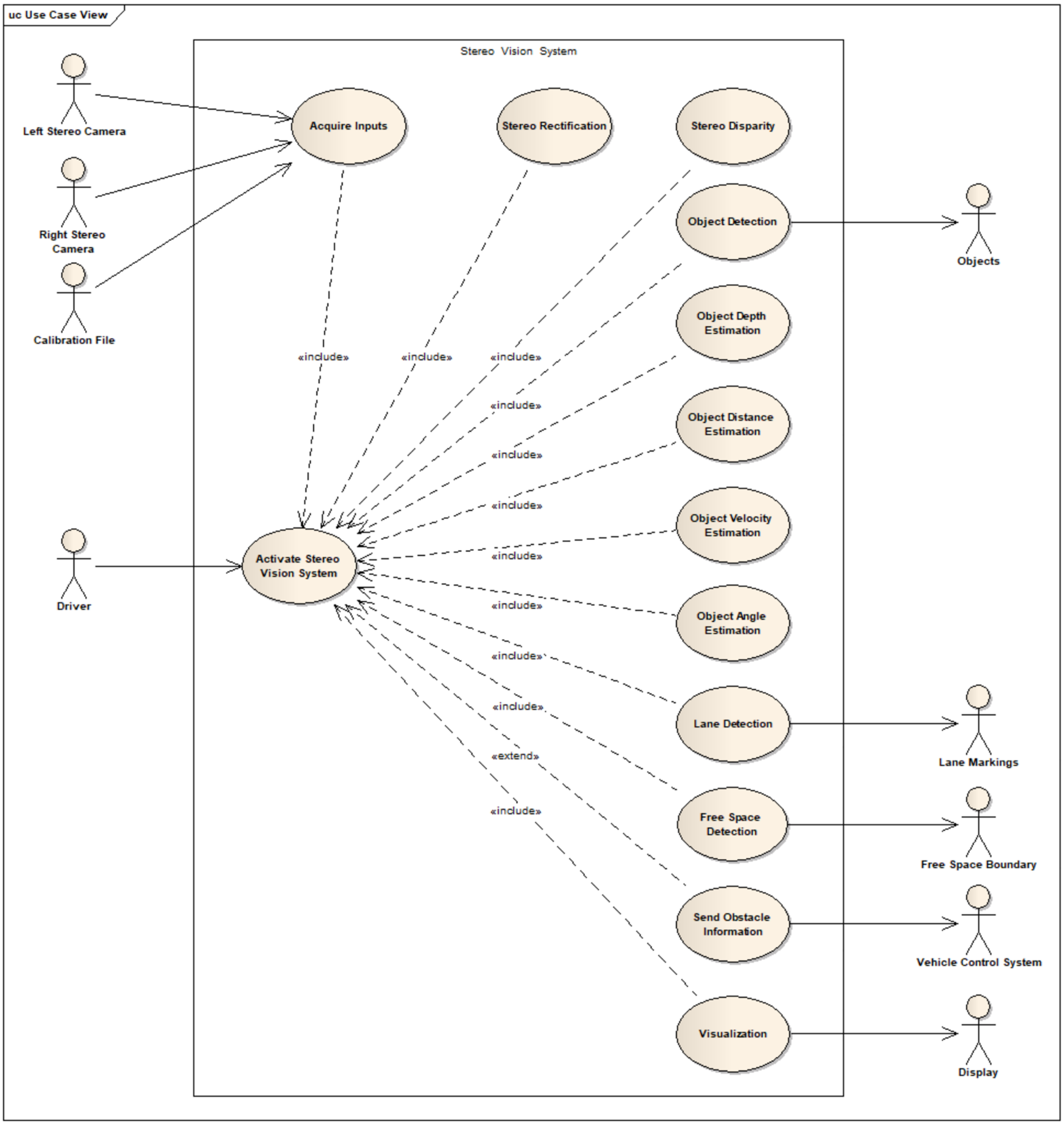


Fig. 7 Use case diagram of the stereo vision system

obstacle information during the platooning of the cooperative automated vehicle. To achieve the stereo perception, the stereo vision system interacts with the following actors: *driver*, *left stereo camera*, *right stereo camera*, *calibration file*, *objects*, *lane markings*, *free space boundary*, *vehicle control system*, and *display*. The *driver* must activate the stereo vision system via vehicle's user interface using *activate stereo vision system* use case. The stereo vision system takes the synchronized input image pair from *left stereo camera* and *right stereo camera* sensor, and the pre-calibrated stereo camera rig configuration parameters from *calibration file* using *acquire inputs* use case. It rectifies the input stereo images using *stereo rectification* use case. The left and right camera images are converted to gray-scale images, and a pyramid of Gaussian images is built up to a specified level. The *stereo disparity* use case uses the Sum of Squared Difference (SSD) pixel matching technique to find the stereo correspondence of every pixel of the left image with the right image to compute the stereo disparity map with respect to the left image. The rectified left camera image is passed as input to the *object detection*, *object depth estimation*, *object distance estimation*, *object velocity estimation*, *object angle estimation*, *lane detection*, *free space detection*, *send obstacle information*, and *visualization* use cases. The *object detection* use case detects and classifies the different class of objects such as car, pedestrian, bicycle, traffic sign, and traffic light within an image and overlays the detected objects on the output image. The *object depth estimation* use case estimates the depth of each detected object by utilizing the computed disparity map. We compute the disparity of each object by calculating an average disparity of the bounding box around its center. The depth of each detected object is then computed using the triangulation method. The *object distance estimation* use case estimates the radial distance of each detected object by utilizing the computed object's depth, and the corresponding object point in the left stereo camera image plane. Similarly, we can estimate the relative radial distance of a lead vehicle with respect to the ego vehicle in a platooning scenario. The *object velocity estimation* use case estimates the velocity of each detected object by utilizing the displacement of the object from the previous position to the current position within the known time. Similarly, we can estimate the relative velocity of a lead vehicle with respect to the ego vehicle in a platooning scenario. The *object angle estimation* use case estimates the azimuth and elevation angle of each detected object by utilizing the computed object's depth and radial distance. Similarly, we can use lead vehicle's depth and horizontal radial distance to estimate the azimuth angle of a lead vehicle with respect to the ego vehicle in a platooning scenario. Similarly, we can use lead vehicle's depth and vertical radial distance to estimate the elevation angle of a lead vehicle with respect to the ego vehicle in a platooning scenario. The *lane detection* use case recognizes the four different types of lane markings such as left adjacent-lane, left ego-lane, right ego-lane, and right adjacent-lane, when they are present on the road, within an image and overlays the recognized lane markings on the output image. The *free space detection* use case identifies the drivable collision-free space as a boundary across the image. It overlays the identified drivable area on the output image. The *send obstacle information* use case sends the perceived obstacle information to the ego *vehicle control system*. The *visualization* use case sends the output image to the *display* in a meaningful way to the ego vehicle's *driver* via an in-vehicle monitor.

5.2 Logical view (object oriented decomposition)

The logical view includes the classes, interfaces, collaborations that describe the system's solution and support the system's functional requirements (see Section 4.3) [21]. With the UML, the static aspects of this object-oriented design decomposition view are captured in the class diagram and the object diagram. The dynamic aspects of this design view are captured in the interaction diagram, the statechart diagram, and the activity diagram. Fig. 8 displays a simplified class diagram of the stereo vision system.

This diagram offers a high-level depiction of the key elements of the system, which include the core classes, the important methods that will be included in those classes, and the relationships between each of the classes. The main control class of this system is called *StereoVisionNetApp* and it is associated with the class *StereoRectifierApp*, *StereoDisparityApp*, *DriveNetApp*, *LaneNetApp*, and *FreeSpaceNetApp*, which contains the current type of objects, lane markings, free space boundary, objects distance, objects velocity, objects azimuth angle, and objects elevation angle. The abstract class *DriveWorksApp* encapsulates the common properties of all the above classes. The class *DriveNetApp* can belong to a detected objects such as a car, person, bicycle, traffic sign, and road sign. The class *LaneNetApp* can belong to a traffic lane markings if they are present on the road. Each road contains four types of traffic lanes: left adjacent-lane, left ego-lane, right ego-lane, and right adjacent-lane. The class *FreeSpaceNetApp* can belong to a free space boundary. The free space boundary separates the obstacle from drivable collision-free space. Each pixel on the boundary is associated with one of the four obstacles: vehicle, pedestrian, curb, and others.

5.3 Process view (process decomposition)

The process view includes the processes that describe the system's concurrency and address the system's performance, scalability, and throughput [21]. With the UML, the static aspects of this process decomposition view are captured in the class diagram and the object diagram. The dynamic aspects of this process view are captured in the interaction diagram, the statechart diagram, and the activity diagram with active classes that represent these processes. State machine diagrams depict the discrete behavior of a part of a designed system through finite state transitions. The *stereo vision system's* state machine diagram is shown in Fig. 9, for all key classes that participate in the system design model (view of participating classes), as discussed in Section 5.2.

Depending on the requirements discussed in Section 4, the stereo vision system is capable of operating in any of the following states: *initialize state*, *process state*, *render state*, *reset state*, and *release state*. After power on, the stereo vision system enters the *initialize state*. In an *initialize state*, the stereo vision system takes the various input parameters from the stereo camera calibration file, and it initializes the parameters of stereo perception, object detection, lane detection, and free space detection. After completion of the *initialize state*, the stereo vision system performs *process state* functions. In this state, the system performs stereo perception, object detection, depth estimation, distance

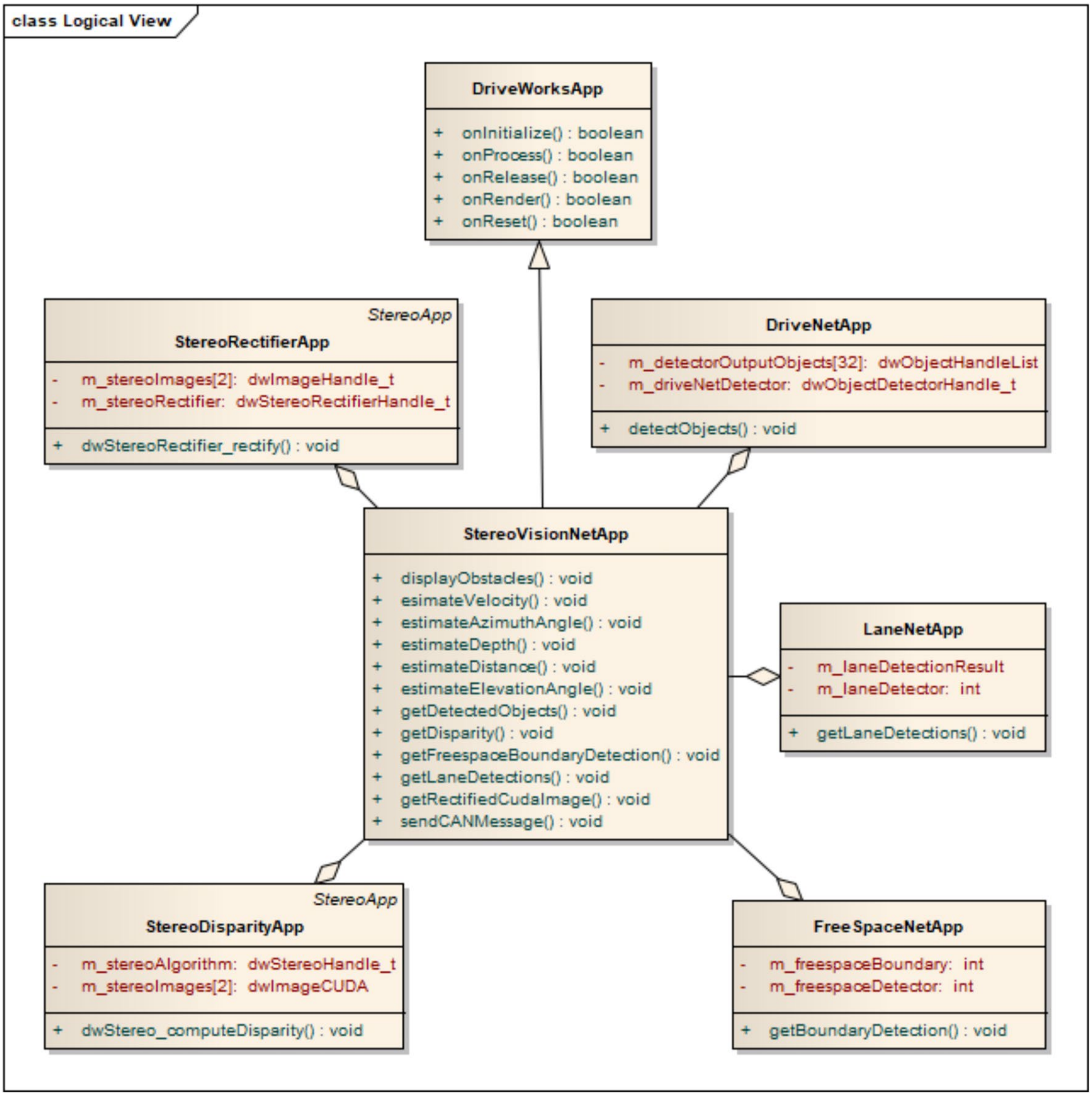

class Logical View
DriveWorksApp
+ onInitialize() : boolean
+ onProcess() : boolean
+ onRelease() : boolean
+ onRender() : boolean
+ onReset() : boolean
StereoApp
StereoRectifierApp
- m_stereoImages[2]: dwImageHandle_t
- m_stereoRectifier: dwStereoRectifierHandle_t
+ dwStereoRectifier_rectify() : void
DriveNetApp
- m_detectorOutputObjects[32]: dwObjectHandleList
- m_driveNetDetector: dwObjectDetectorHandle_t
+ detectObjects() : void
StereoVisionNetApp
+ displayObstacles() : void
+ esimateVelocity() : void
+ estimateAzimuthAngle() : void
+ estimateDepth() : void
+ estimateDistance() : void
+ estimateElevationAngle() : void
+ getDetectedObjects() : void
+ getDisparity() : void
+ getFreespaceBoundaryDetection() : void
+ getLaneDetections() : void
+ getRectifiedCudaImage() : void
+ sendCANMessage() : void
LaneNetApp
- m_laneDetectionResult
- m_laneDetector: int
+ getLaneDetections() : void
StereoApp
StereoDisparityApp
- m_stereoAlgorithm: dwStereoHandle_t
- m_stereoImages[2]: dwImageCUDA
+ dwStereo_computeDisparity() : void
FreeSpaceNetApp
- m_freespaceBoundary: int
- m_freespaceDetector: int
+ getBoundaryDetection() : void

Fig. 8 Class diagram of the stereo vision system

estimation, velocity estimation, view angle estimation, lane detection, and free space detection. The system transmits the results of the *process state* to the vehicle control system. After completion of the *process state*, the stereo vision system performs *render state* functions. In this state, the system displays the *process state* results in a meaningful way to the user via the in-vehicle monitor. In the *reset state*, the stereo vision system re-initializes the various parameters explained in the *initialize state* in case of a reset command or system failure. In the *release state*, the stereo vision system free up used memory of parameters as explained in the *process state*.

5.4 Development view (subsystem decomposition)

The development view or implementation view of a system encompasses the components and files that are used to assemble and release of the physical system, and it addresses the configuration management of the system. With the UML, the static aspects of this development view or decomposition view are captured in the component diagram, the dynamic aspects of this decomposition view are captured in the interaction diagram, the statechart diagram, and the activity diagram [21]. A component is a physical module of code. Components can include both source code files, header

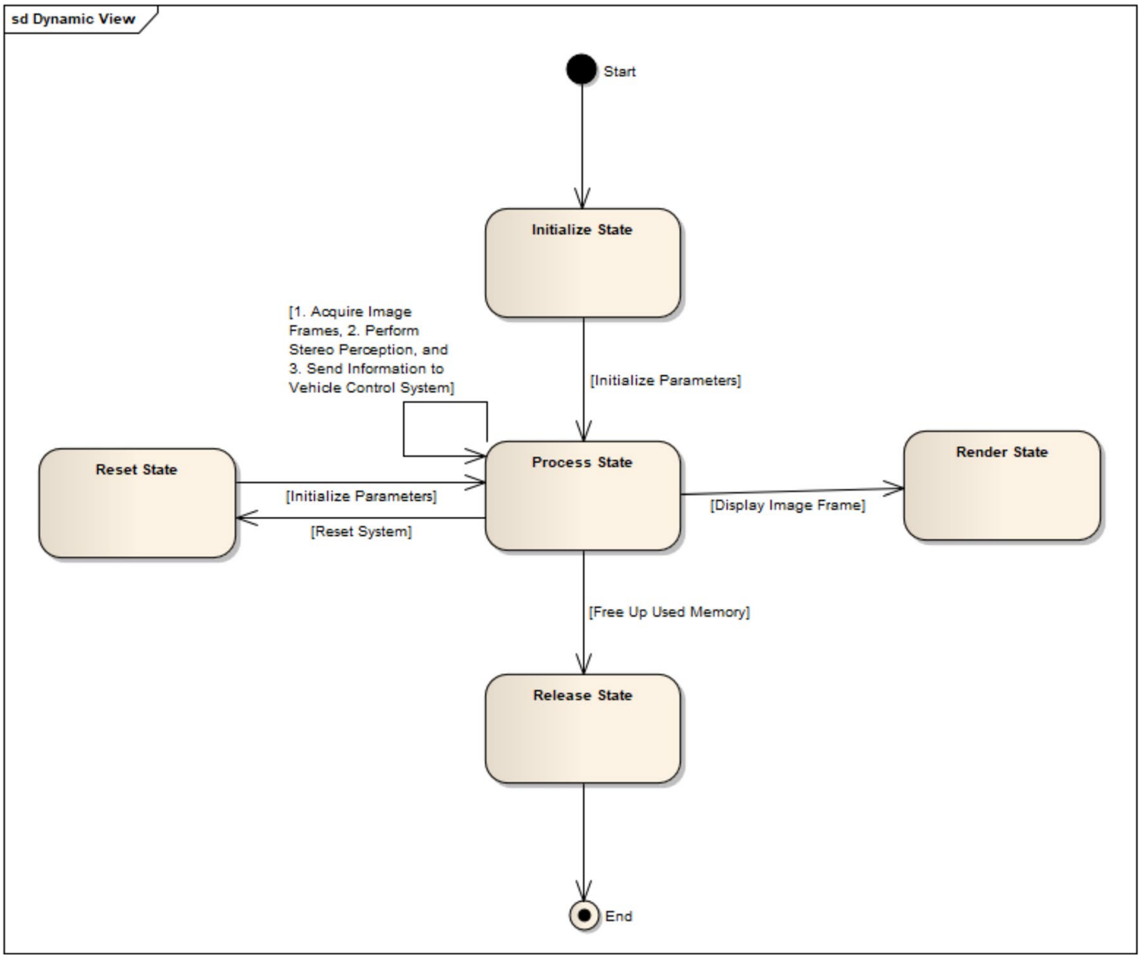


Fig. 9 State diagram of the stereo vision system

files, libraries, and run time files. For example, if we are using C++, each source .cpp, and header .hpp files are separate components. The component diagram displays the components in the system and the dependencies between them. The component diagram of the stereo vision system is shown in Fig. 10.

The component diagram of the stereo vision system consists of a *StereoVisionNet* package component which includes the stereo vision software-related header files, a *StereoVisionNet* component which includes the stereo vision software-related source code files, and their dependencies components are the *DNN* package which includes the Deep Neural Network (DNN) related files, the *Framework* package which includes the DriveWorks framework related files, and the *Stereo* package which includes stereo rectification and stereo disparity related files.

5.5 Physical view (mapping software to hardware)

The physical view (also known as deployment view) includes the nodes representing the hardware configuration on which the system runs [21]. It addresses the distribution, the delivery, and the installation of the parts that make up the physical system. With the UML, the static aspects of this physical view are captured in the deployment diagram, the dynamic aspects of this physical view are captured in the interaction diagram, the statechart diagram, and the activity diagram. The deployment diagram of the stereo vision system is shown in Fig. 11.

The deployment diagram maps the software components to hardware nodes in the system and the dependencies between them. The stereo vision system physical environment consists of a stereo camera (*left stereo camera* and *right stereo camera*) which acquires stereo images, a *vehicle battery* which provides power supply, a *Desktop PC* (Host)

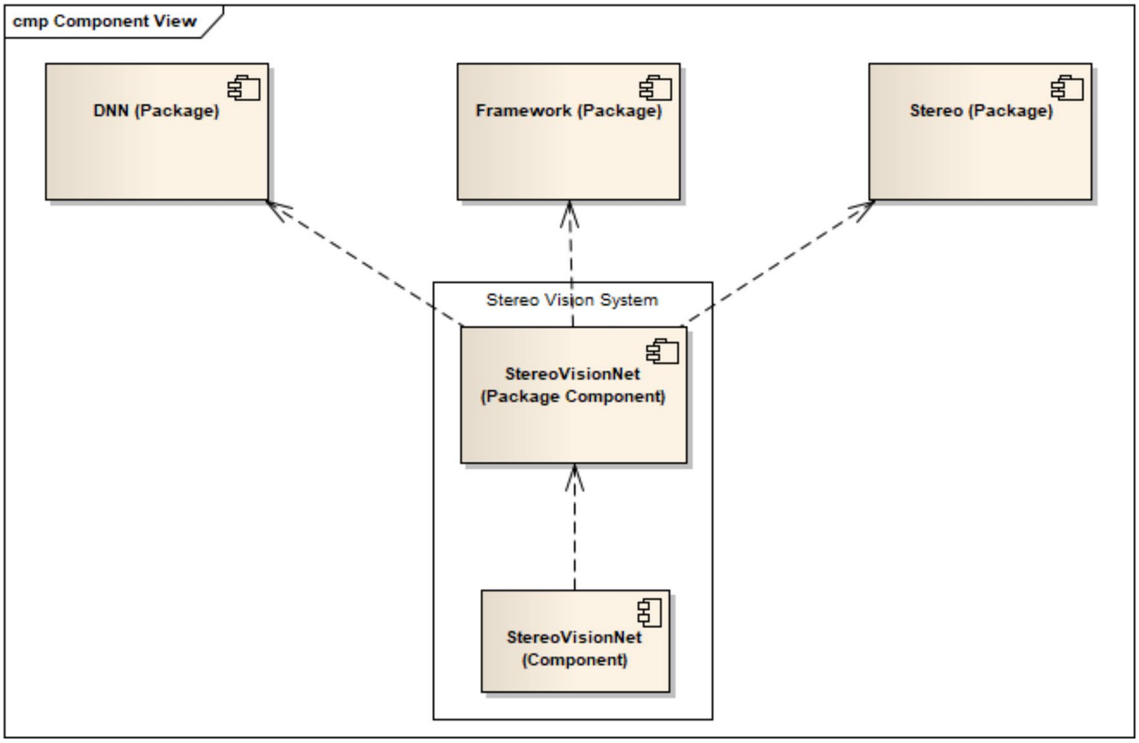


Fig. 10 Component diagram of the stereo vision system

which uses for stereo vision software development and then deploys the developed stereo vision software on to Nvidia Drive PX2, *Nvidia Drive PX2 (Target)* unit, a *Real-Time PC* which contains the vehicle control system Matlab Simulink model, and a *monitor* which visualizes the stereo vision software results in the vehicle's display.

6 Auto code generation and development of stereo vision software

The Sparx Systems Enterprise Architect tool collects information from the logical view (discussed in Section 5.2), the process view (discussed in Section 5.3), and the development view (discussed in Section 5.4) of the model and generates a skeletal code. The logic is written for each skeletal code generated and compiled with a C++ compiler. The development platform of the stereo vision software is Ubuntu 16.04 LTS using C++ language with Nvidia DriveWorks 1.2, CUDA 9.2, and CuDNN 7.4.1 library. Refer to [17] for detailed information of developed stereo vision system methods.

7 Experimental setup

In this section, we describe our proposed stereo vision system experimental setup in detail. Experimental tests have been carried out with a Renault Twizy cooperative automated vehicle research platform, is as shown in Fig. 12.

The lead vehicle and ego vehicle were equipped with GPS and V2V sensors and communicated over a wireless communication network, exchanging information about their position and speed. The ego vehicle, Renault Twizy, is equipped with a Gigabit Multimedia Serial Link (GMSL) stereo camera, Nvidia Drive PX2 hardware platform, with deployed stereo vision software.

7.1 Hardware platform

We used a custom-built stereo camera, which is composed of two identical Sekonix AR0231 GMSL Automotive Cameras, 1928x1208 resolution, 60 *degrees* Horizontal Field Of View (HFOV), 30 *degrees* Vertical Field Of View (VFOV), focal length 5.8 mm, with a baseline of 30 cm [17]. The stereo camera is firmly fixed on the ego vehicle roof using a rigid mounting bar. The power system uses a DC-DC converter module to convert the input from an ego vehicle battery 24V DC to a stable 12V DC to power the Drive PX2 unit. The Drive PX2 hardware platform is mounted on the ego vehicle roof using a rigid mounting box. We used the Ubuntu 16.04 LTS based Desktop PC (Host) with Intel Core-i7 7700k, Nvidia Titan Xp GPU, and 32GB RAM as a development platform for the proposed stereo vision software. We deployed our proposed stereo vision software from

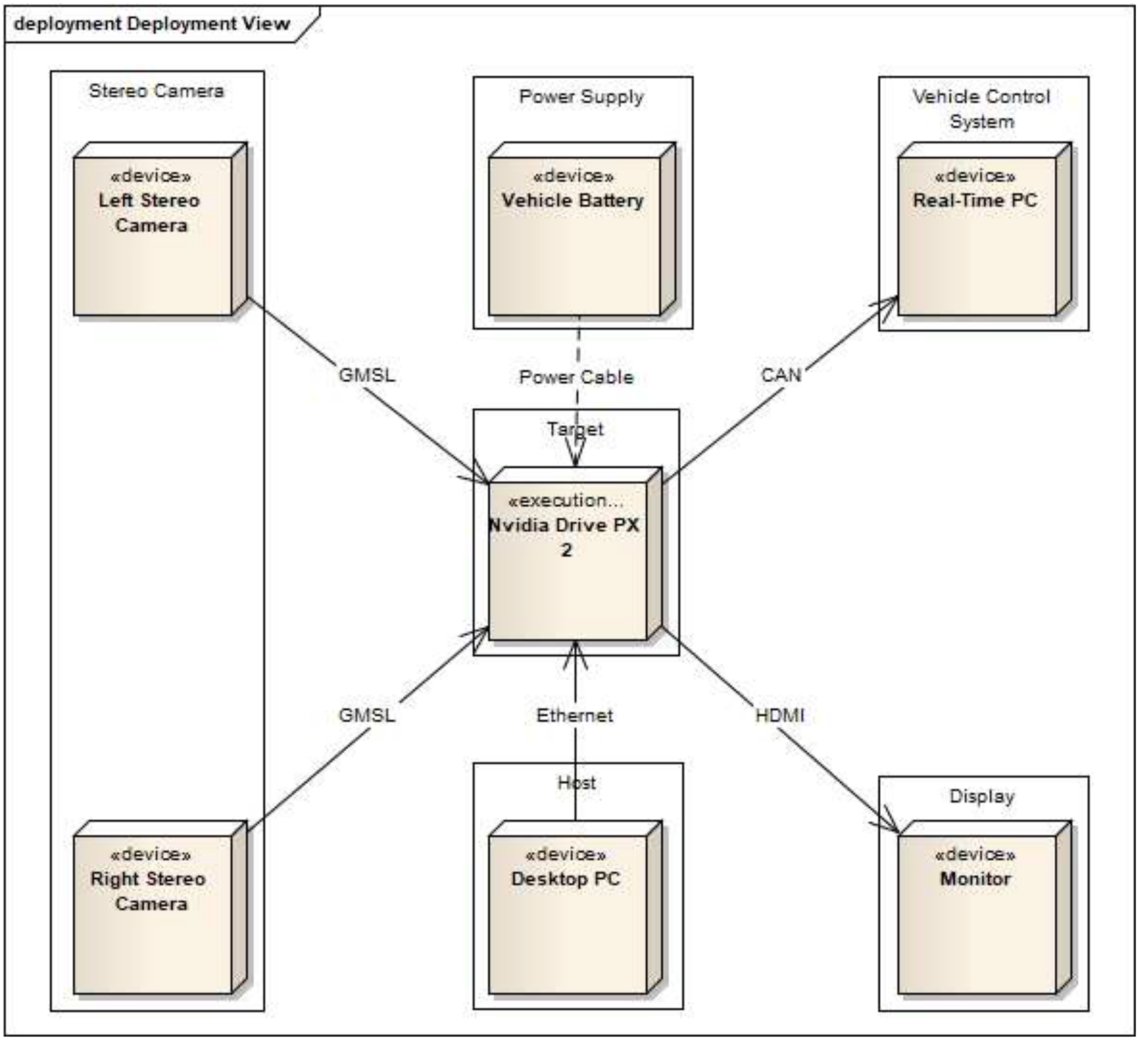


Fig. 11 Deployment diagram of the stereo vision system

the Desktop PC (Host) onto the Drive PX2 AutoChauffeur (Target) hardware platform via Ethernet cable [23]. The hardware interface of the proposed stereo vision system is depicted in Fig. 13.

7.2 Software platform

The presented stereo vision software is part of a perception software of cooperative automated vehicles, which can be used to perceive the environment in real-time. The on-board stereo vision software acquires the input images from the stereo camera via the GMSL cable, detects the various obstacles on the road, and provides their information to the vehicle control system with respect to the ego vehicle via CAN bus. All other perception, planning, and control modules are performed by a real-time computer (Matlab Simulink model), which directly communicates with the actuators over the vehicle CAN bus. We used a small monitor that visualizes the stereo vision software results in the vehicle display via the High-Definition Multimedia Interface (HDMI) cable.

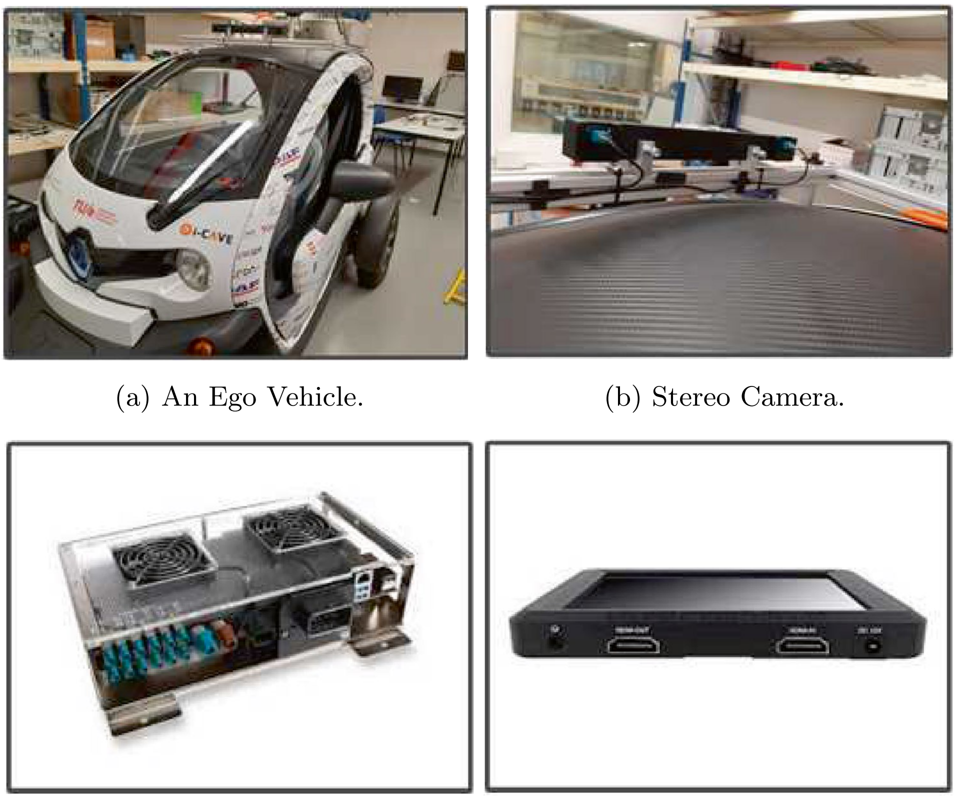
8 Experimental results

This section discusses our proposed stereo vision system experimental evaluation results with simulation and research vehicles.

8.1 Simulation

Initially, using Carla simulator [24] we have rendered a video sequence. During the rendering, we set up the virtual stereo camera to match the geometry of the real stereo camera (i.e., the same baseline, field of view and image

Fig. 12 A Twizy cooperative automated vehicle research platform: **a** An Ego vehicle equipped with **b** GMSL Stereo Camera (on the car roof using a rigid mounting bar), **c** Nvidia Drive PX2 (on the car roof using a rigid mounting box), and **d** HDMI Computer Monitor (at the back side of vehicle front seat)



(a) An Ego Vehicle.

(b) Stereo Camera.

(c) Nvidia Drive PX2.

(d) HDMI Display.

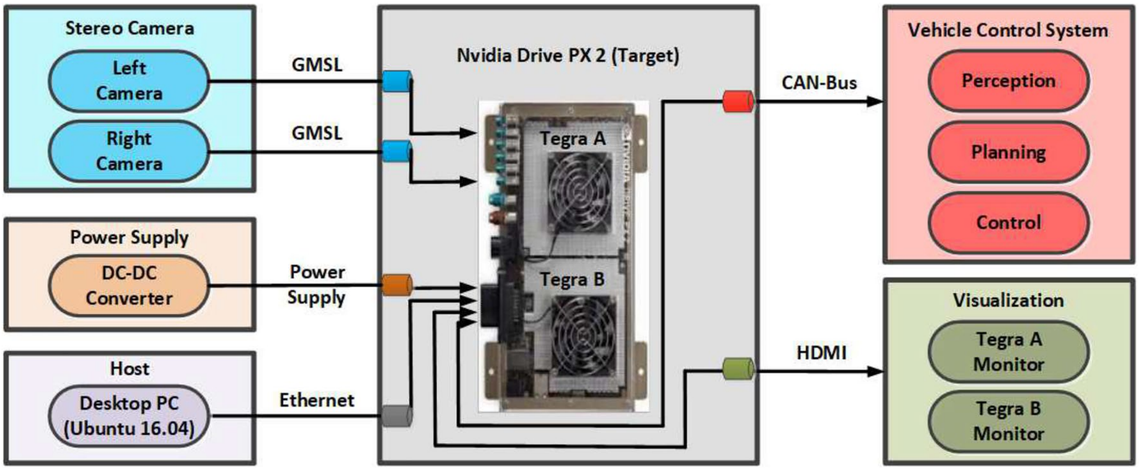
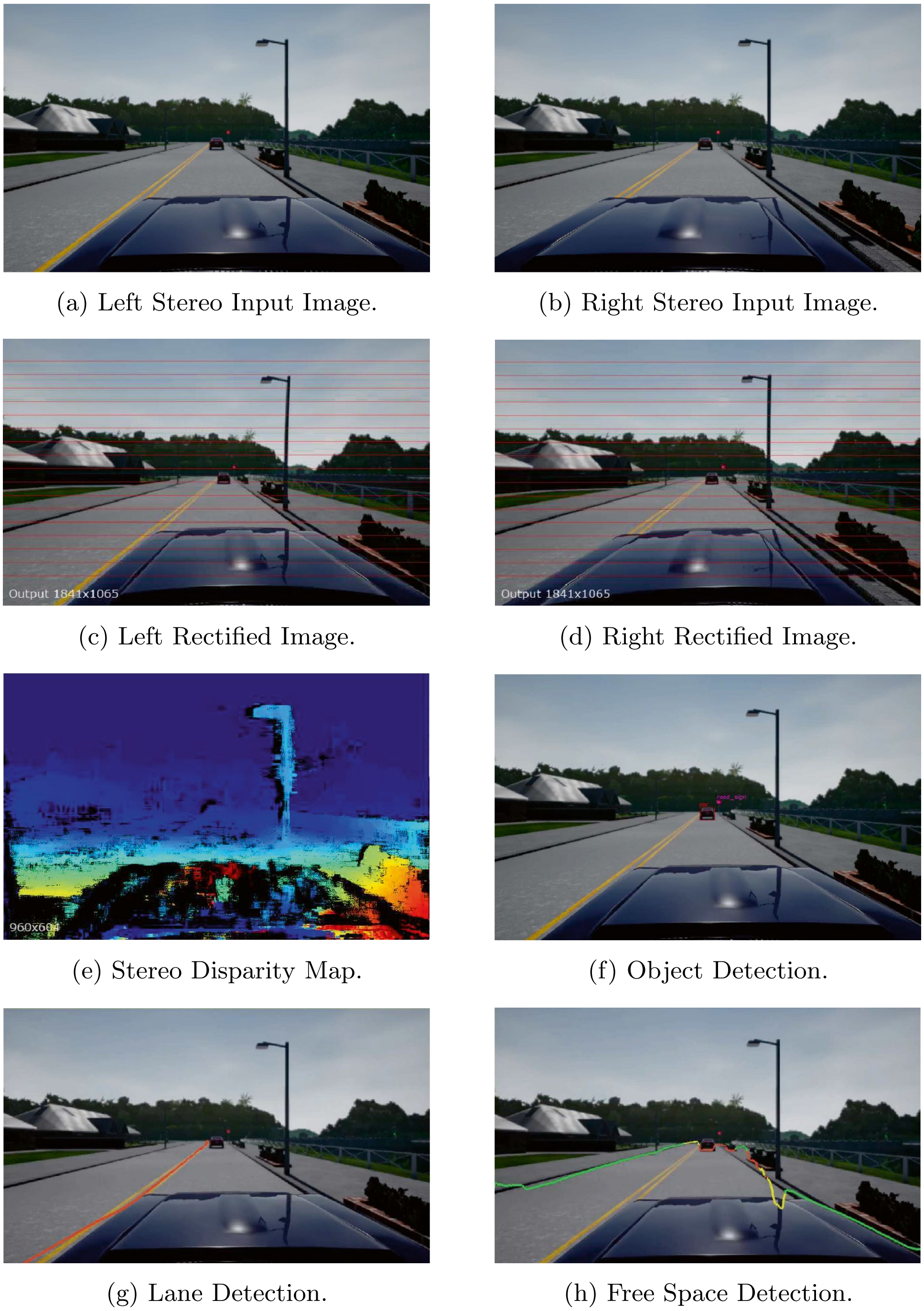


Fig. 13 Hardware interface diagram of the stereo vision system

resolution). The results of the proposed stereo vision software with a Carla simulator on an Ubuntu 16.04 LTS are shown in Fig. 14.

The acquired left, and right stereo input images are shown in Fig. 14a and 14b. The rectified images by the stereo rectification algorithm are shown in Fig. 14c and 14d. A disparity map calculated by the stereo disparity algorithm is shown in Fig. 14e, with a reference image taken from the left camera. The output of the object detection algorithm is in Fig. 14f. The output of the lane detection algorithm is in Fig. 14g. The output of the free space detection algorithm is in Fig. 14h.

Fig. 14 Experimental results of stereo vision with simulation



(a) Left Stereo Input Image.

(b) Right Stereo Input Image.

(c) Left Rectified Image.

(d) Right Rectified Image.

(e) Stereo Disparity Map.

(f) Object Detection.

(g) Lane Detection.

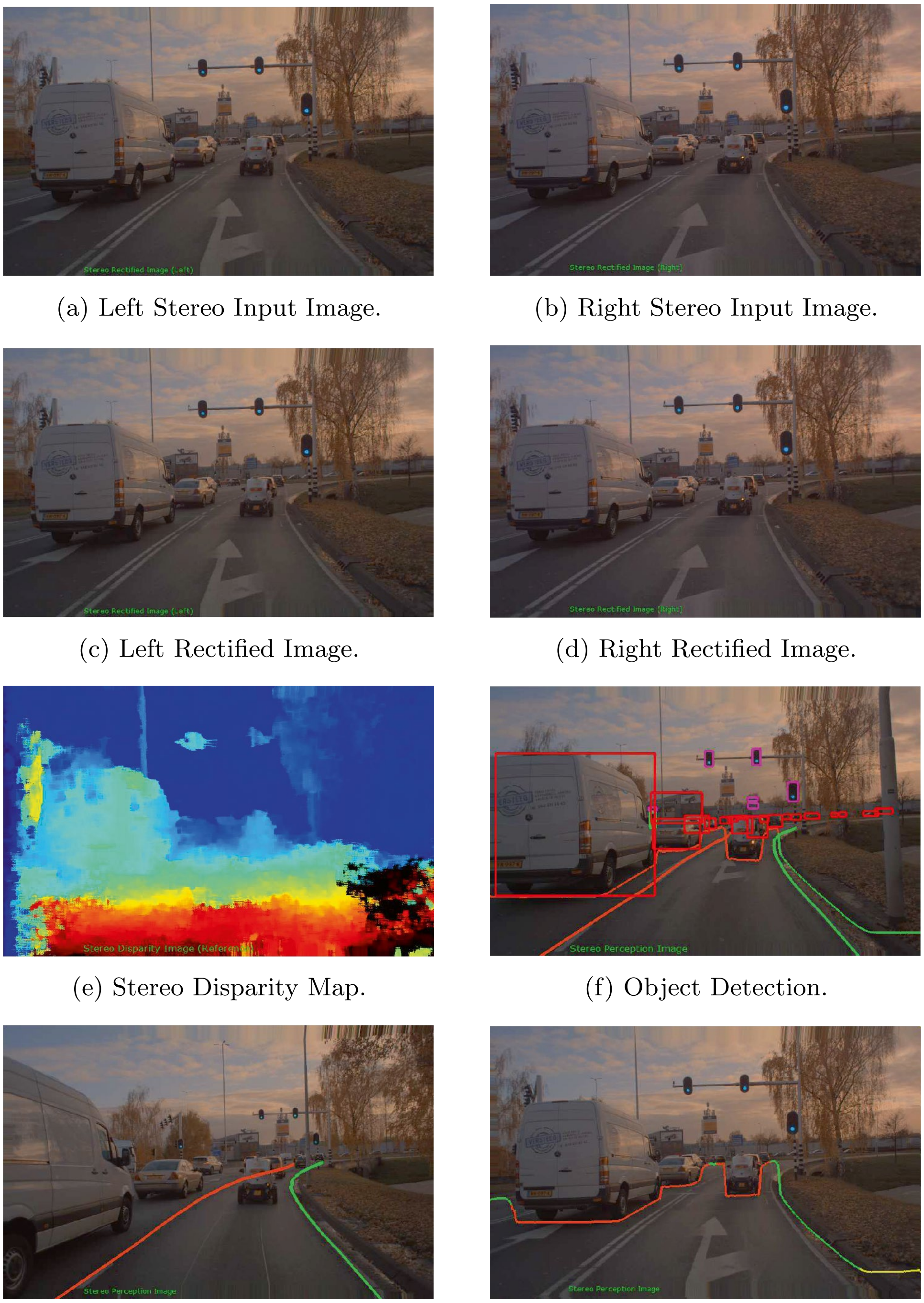
(h) Free Space Detection.

8.2 Research vehicle

We deployed proposed stereo vision software on a Renault Twizy cooperative automated research vehicle. Drive through the Eindhoven University of Technology (TU/e) campus surroundings, several stops at traffic lights, the results of the proposed stereo vision software with a research vehicle on an Nvidia Drive PX2 are shown in Fig. 15.

We acquired left, and right stereo input images using a synchronized GMSL stereo camera on Nvidia Drive PX2 platform are shown in Fig. 15a and 15b. The rectified images by stereo rectification algorithm are shown in Fig. 15c and 15d. A disparity map calculated by the stereo disparity algorithm is shown in Fig. 15e, with a reference image taken from left camera. We can see the output of object detection algorithm is in Fig. 15f. We can see the output of lane detection algorithm is in Fig. 15g. We can see the output of free space detection algorithm is in Fig. 15h. Our final output of a stereo vision software is displayed in Fig. 16.

Fig. 15 Experimental results of stereo vision with research vehicle


Stereo Rectified Image (Left)
Stereo Rectified Image (Right)
Stereo Disparity Image (Reference)
Stereo Perception Image

(a) Left Stereo Input Image.

(b) Right Stereo Input Image.

(c) Left Rectified Image.

(d) Right Rectified Image.

(e) Stereo Disparity Map.

(f) Object Detection.

(g) Lane Detection.

(h) Free Space Detection.

The detected objects on the road the object's information are displayed as bounding boxes with its class label. The color of the bounding boxes represents the classes that it detects are as follows: red for a car, green for a person, blue for a bicycle, magenta for a road sign, and orange for a traffic sign. The recognized lane lines on the road, along with their classification, are displayed as lane markings. The color of the lane markings represents the lane types are as follows: yellow for left adjacent-lane, red for left ego-lane, green for right ego-lane, and blue for the right adjacent-lane. The identified drivable free space on the road and obstacles classification are displayed as free space boundary points. The boundary separates the obstacle from open road space. Each pixel on the boundary is associated with one of the four semantic labels: red for a vehicle, blue for a pedestrian, green for a curb and yellow other.

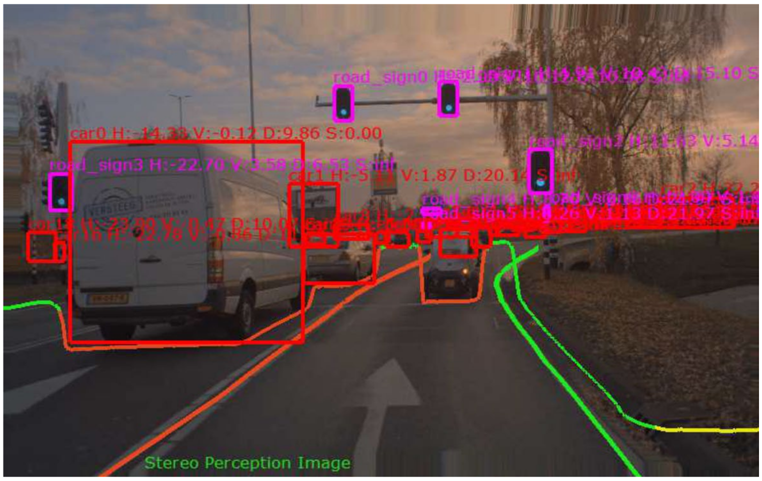

road_sign0
car0 H:-14.33 V:-0.12 D:9.86 S:0.00
road_sign3 H:-22.70 V:3.58 D:6.53 S:inf
car1 H:-5.11 V:1.87 D:20.14 S:inf
road_sign5 H:8.26 V:1.13 D:21.97 S:inf
Stereo Perception Image

Fig. 16 Result of proposed stereo vision software

Table 1 Processing time (in milliseconds)

Platform	Deep Neural Network Perception			Proposed System
	Object Detection	Lane Detection	Free Space Detection	Stereo Vision System
Simulation	11	03	01	23
Vehicle	34	06	04	138

8.3 Performance evaluation

We compare the processing time of the stereo vision system with the Nvidia DNN perception: object detection, lane detection, and free space detection, simulation on the Ubuntu 16.04 (x86_64 architecture) and research vehicle with Drive PX2 (aarch64 architecture) platform, are shown in Table 1 [17].

The processing time of the stereo vision system is 23 ms (43 Hz) on a Desktop PC with Intel Core i7 CPU (Ubuntu 16.04 LTS), Nvidia Titan Xp GPU card, and 138 ms (7.2 Hz) on the Drive PX2 platform, which is suitable for various low-speed cooperative automated vehicles.

The proposed system was presented as a live demonstration at the Intelligent Transport Systems European Congress during 3-6 June 2019 in the Automotive Campus in Helmond, Netherlands, and in the i-CAVE yearly event from 31 October to 1 November 2019 at the Van der Valk Hotel, Leeuw Trainings Center in Eindhoven, Netherlands.

9 Discussions

The stereo vision system's design model discussed here is based on OOAD methodologies with UML concepts. Based on the use of multiple and concurrent views, the 4+1 view model of software architecture is used for describing the architecture of the proposed stereo system. These views are used to explain the system from the perspective of various stakeholders, such as end-users, developers, system engineers, and project managers. A RUP provides the 4+1 views to translate progressively the user needs into a final feasible solution. It takes into account the different views of the system development process life cycle such as the use case view, logical view, process view, development view, and physical view. The presented software development process with the 4+1 view architectural model, however, is not fully auto code generated from the designed system model. The authors implemented the prototype of the designed model based on the auto code generated skeletal code. The authors then integrated the developed stereo vision system with the perception system of cooperative automated vehicles and conducted experiments with the simulator and cooperative automated research platform. Thus, the presented methodology reduces the gap between software engineering methods and cooperative automotive engineering concepts.

10 Conclusions

This article presents a model-based design methodology to develop an onboard stereo vision system for cooperative automated vehicles. The OOAD methodologies with UML concepts simplify the software development process, while software design methodologies make the stereo vision system more modular. The designed stereo vision system model focuses on software engineering design methodologies and integrates stereo camera sensor and perception technology, which is usually seen as an engineering concept-the designed solution fulfils the research goals, system requirements, functional and non-functional requirements of the stereo vision system. The model-driven methods are derived from the designed model, which is developed and integrated into an independent stereo vision system. It can estimate the obstacle motion. The developed system runs on the Carla simulation environment and on the Renault Twizy cooperative automated vehicle research platform in real-time.

Acknowledgements This research work is part of the integrated Cooperative Automated Vehicles (i-CAVE) research programme within the sensing, mapping and localization project (project number 10024085). This i-CAVE programme is funded by NWO (Netherlands Organisation for Scientific Research).

Author Contributions All authors contributed equally to this work.

Funding This research work is funded by NWO (Netherlands Organisation for Scientific Research).

Data availability Not applicable.

Code availability Not applicable.

Declarations

Conflict of interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

Ethical approval Not applicable.

Consent to participate Not applicable.

Consent for publication All authors have agreed to publish this article.

References

1. Union E (2020) Smart, Green and Integrated Transport, Horizon 2020. Online, https://ec.europa.eu/programmes/horizon2020/en/h2020-section/smart-green-and-integrated-transport
2. Bergenhem C, Shladover S, Coelingh E, Englund C, Tsugawa S (2012) Overview of platooning systems. In: Proceedings of the 19th ITS World Congress, Oct 22-26, Vienna, Austria
3. Ploeg J (2014) Analysis and design of controllers for cooperative and automated driving
4. Van der Sande T, Nijmeijer H (2017) From cooperative to autonomous vehicles. In: Sensing and Control for Autonomous Vehicles, pp. 435–452. Springer, Dordrecht
5. Ploeg J, de Haan R (2019) Cooperative automated driving: from platooning to maneuvering. In: 5th International Conference on Vehicle Technology and Intelligent Transport Systems, VEHITS, pp. 5–10 (2019). SCITEPRESS-Science and Technology Publications, Lda
6. i-CAVE: Integrated cooperative automated vehicles project. Online, https://i-cave.nl/ (2019)
7. i-CAVE: i-Cave participation ITS European Congress 2019. Online, https://i-cave.nl/i-cave-deelname-its-european-congress-2019/ (2019)
8. Bertoluzzo M, Bolognesi P, Bruno O, Buja G, Landi A, Zuccollo A (2004) Drive-by-wire systems for ground vehicles. In: 2004 IEEE International Symposium on Industrial Electronics, vol. 1, pp. 711–716. IEEE
9. Reschka A, Böhmer J.R, Gacnik J, Köster F, Wille J.M, Maurer M (2011) Development of software for open autonomous automotive systems in the stadtpilot-project (2011)
10. Dajsuren Y, Loupias G (2019) Safety analysis method for cooperative driving systems. In: 2019 IEEE International Conference on Software Architecture (ICSA), pp. 181–190 . IEEE
11. Furda A, Vlacic L (2010) An object-oriented design of a world model for autonomous city vehicles. In: 2010 IEEE intelligent vehicles symposium, pp 1054–1059 . IEEE
12. Papp Z, Brown C, Bartels C (2008) World modeling for cooperative intelligent vehicles. In: 2008 IEEE intelligent vehicles symposium, pp 1050–1055 . IEEE
13. Garlan D (2000) Software architecture: a roadmap. In: Proceedings of the conference on the future of software engineering, pp 91–101
14. Jo K, Kim J, Kim D, Jang C, Sunwoo M (2014) Development of autonomous car-part i: distributed system architecture and development process. IEEE Trans Indus Electron 61(12):7131–7140
15. Jo K, Kim J, Kim D, Jang C, Sunwoo M (2015) 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 62(8):5119–5132
16. Kemsaram N, Das A, Dubbelman G (2020) A stereo perception framework for autonomous vehicles. In: 2020 IEEE 91st vehicular technology conference (VTC2020-Spring), pp 1–6 . IEEE
17. Kemsaram N, Das A, Dubbelman G (2020) Architecture design and development of an on-board stereo vision system for cooperative automated vehicles. In: Proceedings of 23rd International Conference on Intelligent Transportation Systems 2020 (ITSC 2020). IEEE
18. Serban A.C, Poll E, Visser J (2018) A standard driven software architecture for fully autonomous vehicles. In: 2018 IEEE International Conference on Software Architecture Companion (ICSA-C), pp 120–127 . IEEE
19. Serban A, Poll E, Visser J (2020) A standard driven software architecture for fully autonomous vehicles. J Automot Softw Eng 1:20–33. https://doi.org/10.2991/jase.d.200212.001
20. Rood N (2019) Functional safety analysis and safety pattern application on i-cave
21. Jacobson L, Booch JRG (2021) The unified modeling language reference manual
22. Kruchten P (2004) The Rational Unified Process: an Introduction. Addison-Wesley Professional, USA
23. Drive: Autonomous Vehicle Development Platforms. Online, https://developer.nvidia.com/drive/ (2019)

24. Dosovitskiy A, Ros G, Codevilla F, Lopez A, Koltun V (2017) CARLA: an open urban driving simulator. In: Proceedings of the 1st annual conference on robot learning, pp 1–16

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Narsimlu Kemsaram is a Postdoctoral Researcher with the Mobile Perception Systems Lab and Signal Processing Systems Group at the Eindhoven University of Technology (TU/e), Eindhoven, Netherlands. He is involved in the research project "integrated Cooperative Automated Vehicles (i-CAVE): Sensing, Mapping, and Localization", funded by the Netherlands Organisation for Scientific Research. He received his PhD degree in Computer Science and Engineering from the Jawaharlal Nehru Technological University, Hyderabad, India. During his PhD, he worked as an Assistant Researcher with the Aerospace Information Technology, Department of Computer Science, Division of Space Technology at the Julius Maximilian University (JMU), Würzburg, Germany, on the research project "An Autonomous Sensor Network for the Detection and Observation of Meteors (ASMET)", funded by the European Regional Development Fund. He received his Master of Business Administration degree in Aerospace (Aerospace MBA) from the Toulouse Business School, Toulouse, France, and from the Indian Institute of Management, Bangalore, India, including international modules in the Toulouse (France), Bangalore (India), Brussels (Belgium), and Seattle (USA). He has worked in various aerospace and defense industries. He has published various scientific papers and articles in international conferences and journals. His key research areas are artificial intelligence, computer vision, and deep neural networks with real-world applications for autonomous vehicles, cooperative autonomous vehicles, multi-robot systems, multi-UAV systems, and multirover systems.

Anweshan Das is a full-time PhD Researcher with the Mobile Perception Systems Lab and Signal Processing Systems Group at the Eindhoven University of Technology (TU/e), Eindhoven, Netherlands. He is involved in the research projects "integrated Cooperative Automated Vehicles (i- CAVE): Sensing, Mapping, and Localization", "PRYSTINE: PRogrammable sYSTems for INtelligence in AutomobilEs", and "ViDAS: Vision inspired Driver Assistance Systems", funded by the Netherlands Organisation for Scientific Research. He is also involved in H2020 projects like Inlane, Cloud-LSVA and Magpie. He received his Master of Science degree in Automation and Robotics from the Technical University Dortmund (TU Dortmund), Dortmund, Germany. He received his Bachelor of Technology degree in Mechanical Engineering from the Sikkim Manipal Institute of Technology (SMIT), Sikkim, India. He has published various scientific papers and articles in international conferences and journals. His areas of interest are autonomous robots and vehicles. His research aims to improve vehicle localization systems (both absolute and map-based) and 3D mapmaking techniques using graphbased optimization.

Prof. Dr. Gijs Dubbelman is the CTO of AI in Motion (AIIM) and an Assistant Professor with the Video Coding and Architecture (VCA) group at Eindhoven University of Technology (TU/e), Eindhoven, Netherlands. He heads the Mobile Perception Systems (MPS) research cluster, which focuses on signal processing technologies that allow mobile sensor platforms to perceive the world around them. His areas of expertise include computer vision and multiple-view geometry, and robotics. He has been working on topics such as large-scale visual odometry, Simultaneous Localization And Mapping (SLAM), bundle adjustment, and on-line camera re-calibration. He also has a background in computer graphics, software engineering, and project management. His key research areas are computer vision, pattern recognition, robotics, and sensor fusion. The important application domain of MPS is automotive and transportation. His line of research focuses on 3-D computer vision systems for autonomous robots and vehicles. He has designed and developed state-of-the-art computer vision algorithms for obstacle detection, ego-motion estimation, and simultaneous localization and mapping. In addition, he contributed the COP-SLAM algorithm for real-time embedded visual- SLAM to the open-source scientific project OpenSLAM. He obtained his Bachelor of Science degree in Information and Communication Technology and his Master of Science degree cum laude in Artificial intelligence from the University of Amsterdam. In 2011 he obtained his PhD from the same university on the topic of intrinsic statistical techniques for robust pose estimation. He performed his PhD research on robust estimation of motion parameters from image data using intrinsic statistics in close cooperation with the Intelligent Imaging department of the national organization of applied scientific research of the Netherlands (TNO), which funded the research. In 2011 and 2012, He was a member of the internationally renowned Field Robotics Center of Carnegie Mellon's Robotics Institute. He worked in close collaboration with the National Robotics Engineering Center (NREC) and with CMU's Qatar campus.