Journal of Information Processing Vol.29 227–235 (Mar. 2021)

[DOI: 10.2197/ipsjjip.29.227]

Regular Paper

CoSAM: Co-Simulation Framework for ROS-based Self-driving Systems and MATLAB/Simulink

Keita Miura[1,a)] Shota Tokunaga[2] Yuki Horita[3] Yasuhiro Oda[4] Takuya Azumi[1]

Received: June 30, 2020, Accepted: December 1, 2020

Abstract: In recent year, autonomous vehicles have been developed worldwide. ROS, which is a middleware suitable for the development of a self-driving system, is rarely used in the automotive industry. MATLAB/Simulink, which is a development software suitable for Model-based development, is usually utilized. To integrate a program created with MATLAB/Simulink into a ROS-based self-driving system, it is necessary to convert the program into C++ code and adapt to the network of the ROS-based self-driving system, which makes development inefficient. We used Autoware as ROS-based self-driving system and provided a framework which realizes co-simulation between Autoware and MATLAB/Simulink (CoSAM). CoSAM enables developers to integrate the program created with MATLAB/Simulink into the ROS-based self-driving system without converting into C++ code. Therefore, CoSAM makes the development of the self-driving system easy and efficient. Furthermore, our evaluations of the proposed framework demonstrated its practical potential.

Keywords: self-driving system, development framework, MATLAB/Simulink, robot operating system, and autoware

1. Introduction

Recently, autonomous vehicles have been developed rapidly [1]. The autonomous vehicle is composed of various parts such as a camera, LIDAR, GNSS (GPS), millimeter wave radar, and steering control device. In developing these various parts, Robot Operating System (ROS) [2] has been used. ROS is an open-source middleware running on Linux, and it provides functionalities such as inter-process communication and package management for robot applications.

One of the autonomous driving systems based on ROS is Autoware [3]. Autoware is open-source software for autonomous driving systems. Autoware has a rich set of modules for the self-driving systems, such as detection, localization, planning, and actuation, and cannot only operate the autonomous vehicle but also simulate with actual data.

However, in the automotive industry, the development has often used MATLAB®/Simulink® [4], because MATLAB/Simulink is suitable for Model-based development (MBD) [5]. MBD is a development method using Simulink models and has an advantage that simulation can be performed at the stage of writing the specifications. Therefore, MATLAB/Simulink is often used in the automotive industry. However, a program (which means MATLAB code or a Simulink model) created with MATLAB/Simulink cannot directly communicate to Autoware in the currently adopted development framework. To incorporate such programs into Autoware, it is necessary to convert the programs to C++ code. Although MATLAB/Simulink can automatically generate C++ code from MATLAB code and Simulink model, the C++ code does not correspond to Autoware (i.e., ROS). Moreover, it is possible that a program ported to Autoware will not perform as designed because the MATLAB/Simulink environment differs from that of Autoware. To solve these problems, we proposed a framework called CoSAM (Co-Simulation between Autoware and MATLAB/Simulink) that manages the programs created with MATLAB/Simulink as *nodes* that represent individual processes in ROS. This framework enables communication between MATLAB/Simulink and Autoware without converting the programs to C++ code.

To the best of our knowledge, this is the first work covering co-simulation and operation of a real vehicle using MATLAB/Simulink for the self-driving systems. The main contributions of this study are as follows:

- We provided the framework which enables communication between MATLAB/Simulink and ROS-based self-driving system, and confirmed the practicality of the framework by comparing the data communication time and processing capacity.
- We improved the development efficiency in MATLAB/Simulink based on CoSAM generating MATLAB template scripts and Simulink template models (Section 4.2), which can help a developer design *nodes* for Autoware using MATLAB/Simulink.
- We improved the usability by extending Runtime Manager, which is a graphical user interface (GUI) tool for Autoware, to enable operations for MATLAB/Simulink (Section 4.3), as well as by making available the other functionalities pro-

[1] Saitama University, Saitama 338–8570, Japan
[2] Osaka University, Suita, Osaka 565–0871, Japan
[3] Hitachi, Ltd., Chiyoda, Tokyo 100–0004, Japan
[4] Hitachi Automotive Systems, Ltd., Chiyoda, Tokyo 100–0004, Japan
[a)] k.miura.017@ms.saitama-u.ac.jp

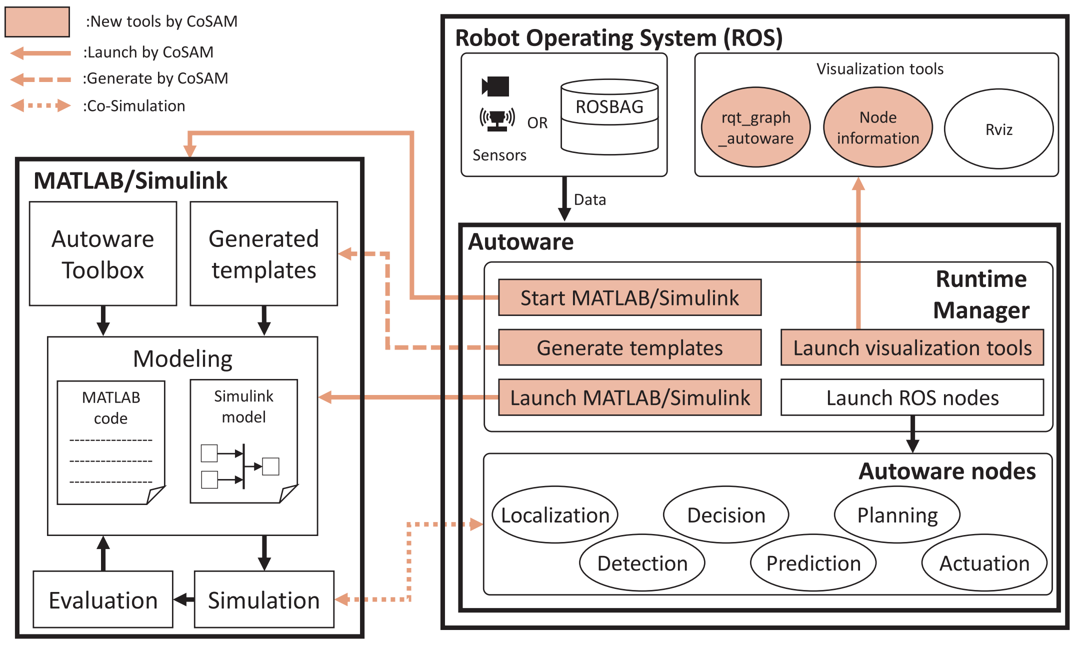

Fig. 1 System model of CoSAM.

vided by CoSAM (e.g., template generation).

2. System Model

The proposed framework (CoSAM) provides several functionalities to enable co-simulation Autoware and MATLAB/Simulink, as shown in **Fig. 1**. Autoware, based on Robot Operating System (ROS), is a popular open-source software project developed for the autonomous vehicles. The colored areas in Fig. 1 are the proposed framework. The solid arrows mean that Runtime Manager launches software, such as MATLAB/Simulink, MATLAB code, Simulink model, and visualization tools. The dashed arrow means that Runtime Manager generates templates with MATLAB/Simulink. The solid arrows and dashed arrow are running commands. The dotted two-way arrow means co-simulation between Autoware and MATLAB/Simulink by using data of the self-driving system, such as "current_pose" including the ego-vehicle's position. This section explains the background of the proposed framework, such as ROS, Autoware, and MATLAB/Simulink.

2.1 Robot Operating System

The Robot Operating System (ROS) [2] is an open-source middleware suite for robot development. ROS includes many software modules with individual capabilities (called packages), and almost all of them have been disclosed. The visualization tool, Rviz, is also a package that can display three-dimensional (3D) models, coordinate systems, and 3D point clouds.

2.1.1 Distributing Computing

In ROS, a large system is separated into small codes called *nodes*. By communicating between *nodes*, the large system is realized. Therefore, ROS can be the distributed computing and improve reusability and readability. This communication method is called *publish/subscribe communication*, and the data communicated between *nodes* is called *topics*. Here, the terms *publish* and *subscribe* refer to the sending and receiving of a *topic*, respectively. All nodes are supervised by the *ROS master*, which stores the names and types of the *topics* and the IP addresses of the *nodes*.

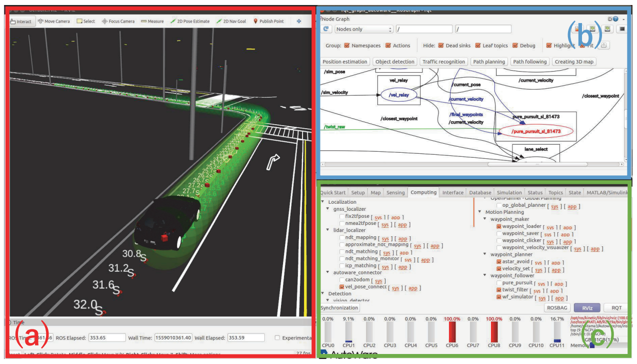

Fig. 2 Screenshot of co-simulation using CoSAM: (a) RViz displaying Autoware status, (b) the rqt_graph_autoware, and (c) the Runtime Manager for CoSAM.

2.1.2 RViz

Software can be developed efficiently by visualizing the state of the system. RViz (as shown in **Fig. 2** (a)) is a 3D visualization tool for ROS. RViz can display various robots and sensor information, and it also supports the ROS *topics* such as cameras and lasers data. When developing control programs, it is possible to confirm the operation in advance by performing simulation using RViz, and then improve the development efficiency.

2.1.3 rosbag

Bag is a file format that stores ROS Messages data. ROS has a console tool called *rosbag* that has a code API to read and write a *Bag* file in C++ or Python. By using *rosbag*, ROS Messages data on *topics* and its transmission time can be stored in a Bag file. *Rosbag* can also reproduce ROS Messages data recorded in the Bag file on *topics*. Since the ROS Messages contain the transmission time, it is possible to reproduce the same state as at

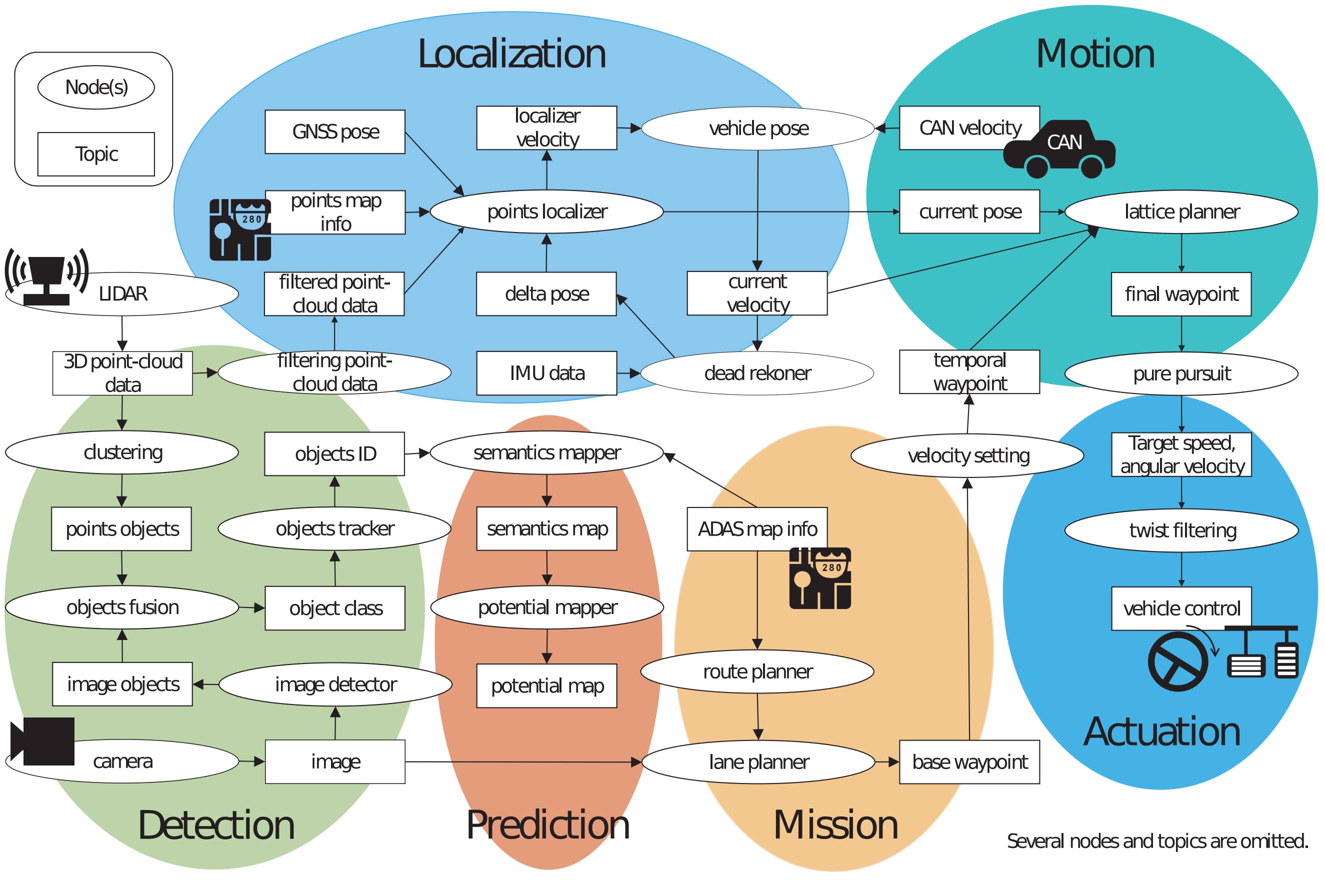

Fig. 3 Autoware node graph.

recording in program reproduction. Since the same situation can be reproduced when problems occur, *rosbag* is useful for problem analysis. It can also be used for performance comparison. In Autoware, simulation using *rosbag* is performed.

2.2 Autoware

Autoware [3], [6] is an open-source self-driving system and ROS-based software. Autoware can be used in embedded systems, such as NVIDIA DRIVE PX2 [6] and Kalray MPPA-256 [7]. Autoware is composed of a *Localization* module, *Detection* model, *Prediction* module, *Mission* module, *Motion* module, and *Actuation* module, and cannot only operate the autonomous vehicle but also simulate with actual data (which is *rosbag* data). **Fig. 3** is the node graph of Autoware. The *Localization* module is to estimate position. The *Detection* module receives sensor data such as image data and LiDAR scanner data, and detects surrounding objects. The *Prediction* module predicts the moving of the detected objects. The *Mission* module plans a trajectory from the place of departure to the destination. The *Motion* module calculates next velocity and angular of the ego-vehicle to move along the trajectory of the *Mission* module. The *Actuation* module operates the ego-vehicle using data from the *Motion* modules.

2.3 MATLAB/Simulink

MATLAB/Simulink [4] is composed of MATLAB and Simulink. MATLAB is development software specialized for matrix calculations and vector operations. MATLAB contains many libraries and toolboxes, and is widely used in diverse fields. Simulink is MATLAB-based designing software enabling graphical visualization. Simulink represents the processing units as connecting input and output of blocks by lines. Since Simulink can display programs graphically, it is useful for co-development and MBD. Simulink programs only the block placements for different parameters: knowledge of the detailed block processing is not needed. This programming method is called *Block Diagram* and the programs created with this method are called *models*. Simulink can change *Block Diagram* models into C or C++ code.

2.3.1 Robotics System Toolbox

Robotics System Toolbox [8], [9] is a system toolbox provided by MathWorks [4]. Robotics System Toolbox provides blocks which act as interface between ROS and MATLAB/Simulink. By creating models using these blocks, the model can communicate to ROS systems as a *node*. Therefore, the model can be co-simulation without converting models to C++ code.

2.3.2 Autoware Toolbox

Autoware Toolbox [10], [11] provides a part of the self-driving systems with MATLAB code and Simulink models using Robotics System Toolbox. Autoware Toolbox is composed of 11 MATLAB code and 7 Simulink models. Because the models and code in Autoware Toolbox are developed imitating Autoware nodes, they are able to communicate with ROS. By referring to Autoware Toolbox, developers can design the code and models using Robotics System Toolbox, which is an effective development approach. CoSAM which is provided by this paper makes the approach more efficient.

3. Design and Implementation

The functionalities provided by CoSAM facilitate the integrated development of Autoware and MATLAB/Simulink. The key functionalities are as follows (Fig. 1):

- They can generate MATLAB template scripts and Simulink template models, and provide visualization tools to aid the template generation (Section 3.1);
- They enable MATLAB/Simulink to operate on Runtime Manager, to display *node* information, and to make use of the other provided functionalities (Section 3.2).

In this section, we describe the design and implementation of each of these functionalities, and use cases of the proposed framework are shown.

3.1 Template Generation

When designing MATLAB code and Simulink models as a *node* co-simulating with Autoware, the code and models must require essential information, such as a *node* name, the *topics* to publish/subscribe, and the *message* type of each *topic*. These information can be obtained by surveying the source code of Autoware and executing ROS commands. However, the need for such analyses places a burden on developers, especially for those who are unfamiliar with ROS. Therefore, we provided the functionalities that enable to generate MATLAB template scripts and Simulink template models that include these necessary information, as the templates help developers design *nodes* in MATLAB/Simulink. Moreover, we provided two visualization tools to help the template generation. One is the rqt_graph_autoware plugin (Fig. 2 (b)). In addition to the functionalities of rqt_graph [2], rqt_graph_autoware can render *node* dependency for the Autoware applications. The other tool displays a list of the running *nodes* and provides information on any *node* selected from the list.

As noted, before the template of the desired *node* is generated, it is necessary to obtain the *node* information; therefore, we created a *.yaml* file containing information pertaining to all Autoware *nodes*. The *.yaml* file also includes the *topic* information (e.g., topic name, message structure) of each node. The *topic* information is required to configure Robotics System Toolbox$^{\text{TM}}$ [8]. Because automatically configuring the information to Robotics System Toolbox, the proposed framework can generate templates without any configuration by developers. Therefore developers can create *nodes* for Autoware in MATLAB/Simulink using the generated template.

The original rqt_graph can only display the node graph of running nodes. The rqt_graph_autoware, an extended version of original rqt_graph provided by the proposed framework, can display the node list in each module. The rqt_graph_autoware can help developers to look for appropriate nodes. To implement the rqt_graph_autoware plugin, we created *.dot* files that render *node* dependency graphs for each Autoware's application. Moreover, to create the GUI for rqt_graph_autoware, we added buttons to rqt_graph using Qt designer, which is a Qt tool for designing a GUI. The buttons were configured to open each *.dot* file, and rqt_graph_autoware is drawn by clicking on these buttons. Therefore, this enables developers to display the relation of the *nodes* included in each Autoware's application as shown in Fig. 3.

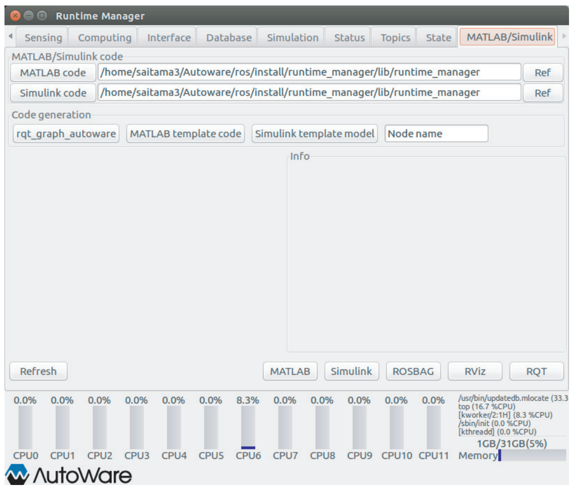

Fig. 4 Runtime Manager of the proposed framework.

To display *node* information, we used the rosnode command-line tool [2]. This tool includes commands that fetch the *node* information, including *rosnode list* and *rosnode info node_name*. The *rosnode list* command displays a list of running *nodes*, whereas *rosnode info node_name* displays information about the *topics* to be published/subscribed by the *node*. Displaying the results of these commands in Runtime Manager renders the *node* information easily comprehensible. Section 3.2 describes the method for displaying these results in Runtime Manager.

This functionality aims to developers of the self-driving systems as the Autoware nodes in Fig. 3. By using this functionality, the code and blocks provided by Robotics System Toolbox are set in MATLAB and Simulink, respectively. Therefore, the developers can use the functionality without looking for the essential information, such as the topic name and type in the node.

Autoware has many nodes as shown in Fig. 3. The template generation and rqt_graph_autoware have corresponded to all nodes in Autoware.

3.2 Runtime Manager for CoSAM

Since Autoware and MATLAB/Simulink are operated with different GUI tools, this is troublesome for developers who want to use the two pieces of software simultaneously. Therefore, we extended the Autoware's GUI tool (i.e., Runtime Manager Fig. 2 (c)) to allow use of MATLAB/Simulink and the functionalities provided in CoSAM (**Fig. 4**). Because the original Runtime Manager can not directly operate MATLAB/Simulink, the difference of GUI tools between Autoware and MATLAB/Simulink was a cause of trouble. By adding the GUI for MATLAB/Simulink to Autoware's GUI tool, developers can operate both software on the same GUI tool. These GUIs enabled the following functionalities:

- Starting MATLAB, Simulink, and rqt_graph_autoware;
- Executing MATLAB scripts and Simulink models;
- Generating MATLAB template scripts and Simulink template models;
- Displaying *node* information.

This unification of operation method simplifies the MATLAB/Simulink operation and utilization of the provided functionalities.

Runtime Manager was designed using the wxPython toolkit [12]. Therefore, we designed the GUIs for the added functionalities using wxGlade, and outputted its designs as wxPython. The GUIs involve buttons and panels that execute each functionality.

We next changed the execution code of the Runtime Manager to configure them for GUI functionalities. The execution code imports modules, includes the code generated by wxGlade, and loads the *.yaml* files. In the execution code, loading *.yaml* files initiates functions that align simple operations to specify buttons. Therefore, by creating a *.yaml* file for MATLAB/Simulink, we configured the initiation of MATLAB, Simulink, and rqt_graph_autoware to each button.

To enable to execute MATLAB scripts and Simulink models on Runtime Manager, we provided multiple GUIs with the following configurations:

- A button to open a dialog for file selection;
- A panel to display the absolute path of the selected file; and
- A button to execute the file displayed on the panel.

This execution button was configured to run if the selected file was a MATLAB/Simulink file (i.e., a .m or .slx file).

To generate MATLAB template scripts and Simulink template models, we designed the GUIs (like Fig. 2 (c)), such as a panel to input the *node* name and buttons to run the execution code that generates the template of the input *node*.

For the *node* information display, we designed two panels, with the first panel displaying the output of the executing a *rosnode list* command. When a *node* is selected from the list, the second panel displays the output of *rosnode info the_selected_node_name* command, which eliminates the need to enter the rosnode command.

3.3 Use Case

The flow of CoSAM is three steps as shown in **Fig. 5**. At first, "pure_pursuit_sl.slx" file which is referred in Fig. 5 (a) is a Simulink model provided by Autoware Toolbox. The Simulink model is launched by pushing the "simulink code" in Runtime Manager (Fig. 5 (a)). Next, Autoware nodes are launched by Runtime Manager. Finally, running the Simulink model can perform co-simulation between Autoware and MATLAB/Simulink. Thus, CoSAM enables developers to operate MATLAB/Simulink and Autoware with the same GUI tool. CoSAM can reduce the trouble caused by them and improve the usability.

The demonstration video as shown in Fig. 5 can be viewed at the following hyperlink: https://youtu.be/NU3ujOiBrqI. In this video, one of the *nodes* in the planning module is executed by MATLAB/Simulink. This simulation facilitates an operational check of MATLAB/Simulink *nodes*.

4. Evaluations

This study aims to improve the development efficiency. We demonstrated this improvement and evaluated the practicality, efficiency, and usability of CoSAM. To evaluate the practicality, we compared the communication time between ROS *nodes* and between a MATLAB/Simulink *node* and ROS *nodes*. Additionally, we performed co-simulation and operation of an autonomous vehicle to show the practicality of the proposed framework. We investigated the design efficiency by measuring the generated MATLAB/Simulink template. To evaluate the usability, we compared the development environments with Autoware, Robotics System Toolbox, and CoSAM. These evaluations demonstrated that CoSAM improved the development efficiency. **Table 1** summarizes the software and hardware environments used in the experiments. We updated the version of ROS, MATLAB/Simulink and OS. These are the latest version in 2019.

4.1 Practicality

CoSAM realized the communication between *nodes* designed using MATLAB/Simulink and Autoware *nodes* to improve the development efficiency. However, it is necessary to consider the practicality of the *nodes* created with MATLAB/Simulink comparing to ROS *nodes*. Therefore, to evaluate the practicality, ROS and MATLAB/Simulink were compared as follows:

(1) according to the relationship between the communication time and the data size when a *message* is sent via ROS and via MATLAB/Simulink, respectively; and
(2) according to the processing capacity when the same type of method was used.

As shown in **Fig. 6**, the communication time was defined as the elapsed time from *Node* 1 published the *message* to *Node* 3 which subscribed the *message* via *Node* 2. We compared the processing

Table 1 Evaluation environment.

CPU	**Model number**	Intel Core i7-6700K
	Cores	12
	Threads	8
	Frequency	4.00 GHz
Memory		32 GB
ROS		kinetic
MATLAB/Simulink		R2019a
OS		Ubuntu 16.04 LIT

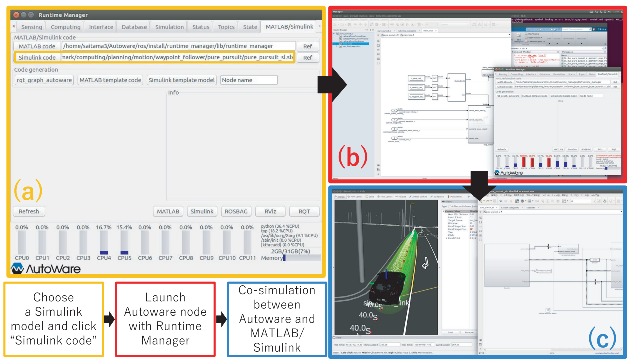
(a)
(b)
(c)
Choose a Simulink model and click "Simulink code"
Launch Autoware node with Runtime Manager
Co-simulation between Autoware and MATLAB/ Simulink
AutoWare

Fig. 5 Use case of Runtime Manager for CoSAM.

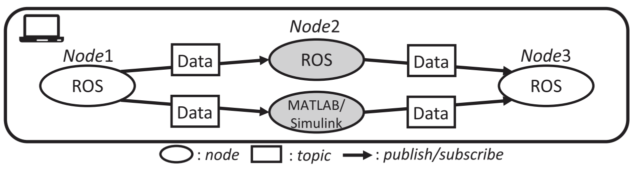

Fig. 6 Measurement of the communication time.

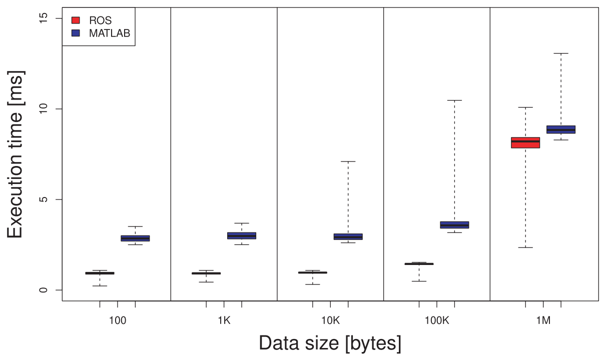

Fig. 7 The communication time according to the size of the *message* data.

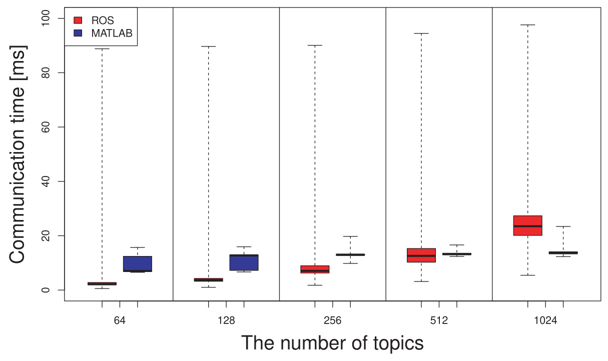

Fig. 9 The communication time according to the size of the *message* data.

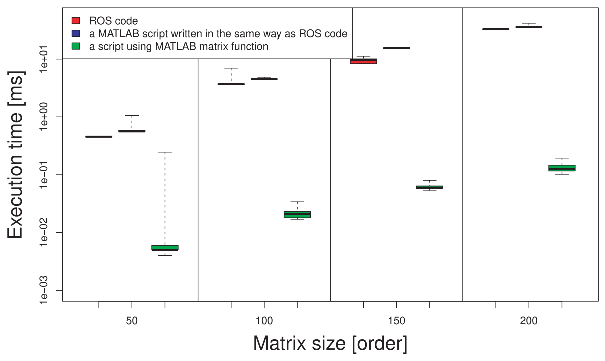

Fig. 8 The average processing time according to each matrix size.

performance by the result using the same machine and running 1,000 iterations (*Node* 1 published the *message* at 10 Hz).

We measured the communication time between ROS and MATLAB/Simulink when the *message* data size on each *topic* was configured to 100, 1 K, 10 K, 100 K, and 1 M bytes. **Fig. 7** shows the communication time via ROS and MATLAB/Simulink summarized against each data size. Both the ROS and MATLAB/Simulink communication time increased along with data size, although the data transfer by MATLAB/Simulink had an overhead exceeding that of ROS. However, the MATLAB/Simulink communication time did not exceed the Autoware maximum of 32 Hz. Human drivers generally take more than 100 ms to understand traffic conditions and make motion decisions [13]. The communication time is significantly lower than 100 ms, therefore the development of the self-driving system will not be affected.

To evaluate the processing capacity, we calculated square matrices in the order of 50, 100, 150, and 200 with ROS and MATLAB/Simulink, and measured the execution time. The time required to process each matrix size was measured and we evaluated the performance of the functions provided by MATLAB/Simulink. Therefore, we measured the MATLAB/Simulink processing by using two MATLAB scripts: one written in the same way as the ROS code, and the other using MATLAB matrix functions. **Fig. 8** shows the processing time at each matrix size. When running for the MATLAB script written in the same way as the ROS code, the processing time of ROS and MATLAB/Simulink was approximately the same. However, when the MATLAB script used the matrix functions, its processing time was significantly shorter than that of the other two methods, because the processing was executed on multiple cores with multiple threads, even when this was unspecified. Comparison of the processing time with the communication time revealed that the script using the matrix functions was again significantly faster, thereby confirming that applications of the functions provided by MATLAB/Simulink code enabled the handling of processes with more complexity (e.g., image processing), even when accounting for the communication time. Therefore, as shown in the videos in Section 3.3, the practicality of CoSAM is demonstrated.

Moreover, we measured the execution time when communicating multiple topics for the evaluation of the practicality. The topic type is "geometry_msgs/TwistStamped" which is actually used in Autoware. This experiment gradually increased the number of topics and checked whether the difference in performance between ROS and MATLAB/Simulink appears. As shown in **Fig. 9**, MATLAB/Simulink was found to be stable while ROS had variability. The number of topics published and subscribed in Autoware is no more than 1,000. Therefore, the result shows the proposed framework can be used for the development of the self-driving systems.

4.2 Efficiency

To improve the design efficiency, we provided functionality to generate both the MATLAB template scripts and Simulink template models. These templates assist developers to design *nodes* for Autoware in MATLAB/Simulink. In order to quantitatively evaluate the efficiency, not the communication time, we counted the number of lines and blocks of the automatically generated template code and models. Because the generated code and blocks do not need to be written by the developers, a large amount of the code and blocks will be more efficient.

Table 2 shows the amount of the template generated by a MATLAB template script. The MATLAB template script defines the essential information, as mentioned in Section 3.1, and creates callback functions used when a *topic* is subscribed. For example, the lane_stop *node* required for planning has one *publisher* and five *subscribers*. One line is generated to define a *node*, a *subscriber*, and a *publisher*, and two lines are generated to define the callback function. Therefore, in total, 17 lines are generated for the MATLAB template script for the lane_stop *node*.

When creating a Simulink model, it is necessary to place and configure the Simulink blocks, to define the model name, and to connect the blocks. **Table 3** shows the number of Simulink blocks placed and the settings created by a Simulink template

Table 2 Task reduction using MATLAB template scripts.

	MATLAB template scripts
Generated lines	(1) + α(2) + β((3) + 2(4))

(1): Defining *node*
(2): Defining *publisher*
(3): Defining *subscriber*
(4): Defining callback function
α: The number of *publishers*
β: The number of *subscribers*

Table 3 Task reduction using Simulink template models.

	Simulink template models
Simulink blocks	α((1) + (2) + (3)) + β((4) + (5) + (6))
Settings	(i) + (α + β)((ii) + (iii) + (iv) + 2(v))

(1): Placing *Publisher*	(i): Defining model name
(2): Placing *Message*	(ii): Setting *message* name
(3): Placing Bus Assignment	(iii): Setting *topic* name
(4): Placing *Subscriber*	(iv): Configuring *topic* source
(5): Placing Bus Selector	(v): Connecting blocks
(6): Placing Terminal	α: The number of *publishers*
	β: The number of *subscribers*

Table 4 Functionalities available with Autoware, Robotics System Toolbox, and CoSAM.

	Autoware [3]	**Robotics** [8] **System Toolbox**	**CoSAM**
Operating Autoware	✓		✓
Operating MATLAB/Simulink		✓	✓
Communicating between Autoware and MATLAB/Simulink		✓	✓
Drawing node dependency	✓		✓
Generating MATLAB/Simulink			✓
Displaying node information			✓

model. CoSAM defines the model name and places the essential Simulink blocks, thereby creating a model for Autoware. Additionally, the Simulink blocks are configured and connected together. For example, when the Simulink template model of lane_stop *node* is generated, 18 Simulink blocks are placed and 31 settings are configured in total.

If the functionality allowing the MATLAB/Simulink templates to be generated is not provided, the developer must examine the *node* information and define it in a MATLAB script or a Simulink model. By contrast, when the templates are used, this becomes unnecessary; therefore, this improves design efficiency.

The template created with CoSAM is used by Autoware Toolbox [10], [11]. Autoware Toolbox is an open-source self-driving system using the template. The developer who does not know how to use the template can refer to Autoware Toolbox.

4.3 Usability

CoSAM realized the operation of MATLAB/Simulink in Autoware and provided functionalities to improve the usability. We compared with each functionality of Autoware, Robotics System Toolbox, and CoSAM, as summarized in **Table 4**.

Autoware cannot operate MATLAB/Simulink, and Robotics System Toolbox cannot operate Autoware. CoSAM can operate the functionalities required to operate MATLAB/Simulink in Runtime Manager for CoSAM, such as starting MATLAB/Simulink or executing MATLAB scripts and Simulink models. Therefore, CoSAM can operate both of the systems. Communication between Autoware and MATLAB/Simulink is possible in Robotics System Toolbox and CoSAM. Moreover, CoSAM enables to visualize *node* dependency using the rqt_graph_autoware plugin created by extending the rqt_graph available in Autoware. In addition to these features, CoSAM can generate the MATLAB/Simulink templates and display the *node* information. Because increasing the number of the available functionalities, the usability is also enhanced, which in turn improves the development efficiency.

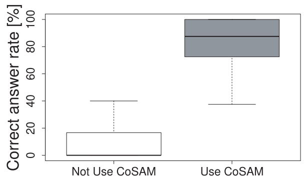

Fig. 10 The correct answer rate.

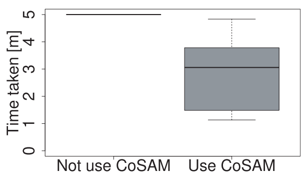

Fig. 11 The time taken for the experiment.

Moreover, we evaluated the usability of the proposed framework by having eight people use the rqt_graph_autoware. The usability test let the users enumerate the publish and subscribe topics of a particular node within five minutes which is the time limit and measured the time and correct answer rate. The time and correct answer rate represent the usability quantitatively because reducing the time means that working processes are decreased and increasing the correct answer rate means the high coverage. Eight people who normally use ROS tried the usability test. The usability test was conducted two times because of comparing the time and coverage between using the proposed framework and not using it. As the result in **Fig. 10** shows, if not using the rqt_graph_autoware, the users could not find the node and topics, because they have no knowledge of the structure of Autoware. By using the rqt_graph_autoware, all users could find more topics than when not using it. As the result in **Fig. 11** shows, if not using the rqt_graph_autoware, all users took five minutes. Because the users needed to use commands to find the node and topics if not using the rqt_graph_autoware, the result of the experiment was five minutes for all users. However, by using the rqt_graph_autoware, the time taken for the experiment decreased for all users because the tool leads to a quick understanding of the node relation graph. Therefore, the proposed framework can enhance the usability.

5. Related Work

To the best of our knowledge, this is the first work covering co-simulation and operation of a real vehicle using MATLAB/Simulink for the self-driving systems. This section introduces similar frameworks and compares our work and them.

MontiSim [14]: This framework tests the behavior of self-driving systems using a model environment that simplifies the real-world environment. The simulator supports both high-level simulations (performed in large environments, such as urban areas) and low-level simulations (detailed testing of individual components). Furthermore, map data are generated from OpenStreetMap, with road signs and traffic lights generated for each intersection on the map. Unlike this framework, our framework has a framework that promotes the development of autonomous driving systems.

AsFault [15]: This is a tool that automatically creates virtual test environments for lane-keeping systems. Compared with randomly created road networks, AsFault creates road networks occurring lane departures twice as many as random networks, thereby creating difficult networks. Unlike this framework, our framework is used for not only lane-keeping systems but all self-driving systems.

AutonoVi-Sim [16]: AutonoVi-Sim improves learning by autonomous driving algorithms and creates autonomous driving tests and environmental elements, including changes according to time and weather. This simulator allows the study of machine learning. Our framework is not aimed at the machine learning. Our framework focuses at a development approach that combines MATLAB/Simulink and ROS.

CARLA [17]: CARLA is an open-source simulator focused on city driving by allowing establishment of sensor parameters and test environments, such as weather and time. Additionally, CARLA can create datasets for machine learning and imitation learning. Our framework is different from this research because our work is not to create a simulator.

6. Conclusion

In this paper, we described the development of an integrated development framework for Autoware with MATLAB/Simulink (CoSAM) that facilitated communication between Autoware and MATLAB/Simulink. We evaluated the data communication time and processing capacity of MATLAB/Simulink, and confirmed the practicality of the method by using both co-simulations and experiments using an autonomous vehicle. CoSAM facilitated the generation of MATLAB/Simulink templates that can help developers create models using MATLAB/Simulink for Autoware, thereby improving the design efficiency. The functionalities added to CoSAM allow Runtime Manager to operate MATLAB/Simulink and various functionalities, further improving the usability. Furthermore, we make this framework correspond to Autoware Toolbox, which improves the efficiently and usability. For future work, the proposed framework has currently only corresponded to Autoware, because we extended Runtime Manager of Autoware GUI. The template generation and rqt_graph_autoware can be improved to correspond to other self-driving systems. It would be better to change to a framework that can be adapted to other self-driving systems.

Acknowledgments This work was partially supported by Mr. Tohru Kikawada and JST PRESTO, Japan (grant No. JPMJPR1751).

References

[1] Berger, C. and Rumpe, B.: Autonomous driving – 5 years after the urban challenge: The anticipatory vehicle as a cyber-physical system, *Computing Research Repository* (2014).

[2] ROS.org, available from ⟨http://www.ros.org⟩.

[3] Autoware/github.com, available from ⟨http://github.com/CPFL/Autoware⟩.

[4] MATLAB/Simulink, available from ⟨http:///www.mathworks.com⟩.

[5] Honda, K., Kojima, S., Fujimoto, H., Edahiro, M. and Azumi, T.: Mapping method of matlab/simulink model for embedded many-core platform, *Proc. PDP* (2020).

[6] Kato, S., Tokunaga, S., Maruyama, Y., Maeda, S., Hirabayashi, M., Kitsukawa, Y., Monrroy, A., Ando, T., Fujii, Y. and Azumi, T.: Autoware on board: Enabling autonomous vehicles with embedded systems, *Proc. ICCPS* (2018).

[7] Maruyama, Y., Kato, S. and Azumi, T.: Exploring scalable data allocation and parallel computing on NoC-based embedded many cores, *Proc. ICCD* (2017).

[8] Robotics System Toolbox, available from ⟨https://mathworks.com/products/robotics.html⟩.

[9] Saito, Y., Azumi, T., Kato, S. and Nishio, N.: Priority and synchronization support for ROS, *Proc. ICCPS, Networks, and Applications* (2016).

[10] Tokunaga, S., Ota, N., Tange, Y., Miura, K. and Azumi, T.: MATLAB/Simulink Benchmark Suite for ROS-based Self-driving System: demo abstract, *Proc. ICCPS* (2019).

[11] Miura, K., Tokunaga, S., Ota, N., Tange, Y. and Azumi, T.: Autoware Toolbox: MATLAB/Simulink Benchmark Suite for ROS-based Self-driving Software Platform, *Proc. RSP* (2019).

[12] wxpython.org, available from ⟨http://wxpython.org⟩.

[13] Lin, S.-C., Zhang, Y., Hsu, C.-H., Skach, M., Haque, M.E., Tang, L. and Mars, J.: The architectural implications of autonomous driving: Constraints and acceleration, *Proc. ASPLOS* (2018).

[14] Filippo, G., Evgeny, K., Roth, A., Bernhard, R. and von Wenckstern, M.: Simulation framework for executing component and connector models of self-driving vehicles, *Proc. International Conference on MODELS* (2017).

[15] Gambi, A., Mueller, M. and Fraser, G.: Automatically testing self-driving cars with search-based procedural content generation, *Proc. ISSTA* (2019).

[16] Andrew, B., Sahil, N., Lucas, P., Barber, D. and Dinesh, M.: Autonovi-sim: Autonomous vehicle simulation platform with weather, sensing, and traffic control, *Proc. IEEE/CVF Conference on CVPRW* (2017).

[17] Dosovitskiy, A., Ros, G., Codevilla, F., López, A. and Koltun1, V.: Carla: An open urban driving simulator, *Proc. CoRL* (2017).

Keita Miura is a master student of Graduate School of Science and Engineering, Saitama University. He received his B.E. degree from Saitama University in 2019. His research interests in self-driving systems.

Shota Tokunaga received his M.E. degree from Osaka University in 2019. His research interests in self-driving systems.

Yuki Horita received his M.S. degree from the University of Tokyo in 2006. He then joined the Research and Development Group of Hitachi Ltd. His research interests include automated vehicles, cooperative ITS, embedded systems and parallel processing.

Yasuhiro Oda received his B.E. degree from Yokohama National University in 2002. Currently he is in the Embedded Systems Engineering Group of Hitachi Industry & Control Solutions, Ltd. His research interests include automated vehicles, embedded systems.

Takuya Azumi is an Associate Professor at the Graduate School of Science and Engineering, Saitama University. He received his Ph.D. degree from the Graduate School of Information Science, Nagoya University. From 2008 to 2010, he was under the research fellowship for young scientists for Japan Society for the Promotion of Science. From 2010 to 2014, he was an Assistant Professor at the College of Information Science and Engineering, Ritsumeikan University. From 2014 to 2018, he was an Assistant Professor at the Graduate School of Engineering Science, Osaka University. His research interests include real-time operating systems and component-based development. He is a member of IEEE, ACM, IEICE, and JSSST.