Journal of Technology and Social Science (JTSS)

Application of Model-Based Design to Autonomous Driving for Four Mecanum Wheel AGVs

Haohao Zhang[1,a,*], Kazuhiro Motegi [1,b] and Yoichi Shiraishi [1,c]

[1]Domain of Mechanical Science and Technology, Graduate School of Science and Technology, Gunma University, 29-1 Honcho, Oota, Gunma 373-0057, Japan

* Corresponding author

[a]<t192b004@gunma-u.ac.jp>, [b]<motegi@gunma-u.ac.jp>, [c]< yoichi.siraisi@gunma-u.ac.jp>

Keywords: model-based design, ROS, simulink, autonomous driving, AGV

Abstract. We describe an autonomous driving application for all-wheel-driven automatic guided vehicles (AGVs) with four mecanum wheels on the basis of a model-based design (MBD) method. The MBD method allows us to focus on control logic design by implementing graphical modeling and automatically generating program codes in a short time. Using a robot operating system (ROS) and the MBD platform tool Simulink, we can communicate with the ROS network without having to write C/C++ codes and verify them, which saves time. Herein, we first describe the hardware construction of an AGV, followed by a detailed description of the control architecture of the AGV on the basis of ROS and Simulink. We also demonstrate simultaneous localization and mapping and navigation technology in autonomous mode. The exhibition demo result of the AGV under an autonomous mode using the YOLO v3 object detection algorithm at Tokyo International Robot Exhibition 2019 shows the feasibility of the MBD method and the proposed control architecture.

1. Introduction

As one component of Industry 4.0, autonomous robots have been researched for a multitude of scenarios such as self-driving robots, controlling robots in manufacturing lines, home delivery service robots, and automatic guided vehicles (AGVs) [1]. However, current controlling methods of AGVs still rely on centralized controller systems to dispatch AGVs along predefined fixed paths marked with tapes or magnetic markers [2]. With the rapid advancement of the Internet of Things (IoT), traditional methods for controlling AGVs are facing a significant challenge, and it is becoming difficult to meet the demands of frequent and flexible change guide paths. Thus, a new autonomous driving framework for AGVs is required, enabling AGVs to be much more flexible, intelligent, and compatible with IoT devices.

The model-based design (MBD) is a method for designing control systems for embedded systems [3,4], especially for the ECU (Engine Control Unit) development in the automobile industry [5]. The development process of a control system is focused on the model design using graphical modeling and code generation capabilities. Simulink (MathWorks, Inc.) and SCADE (Ansys, Inc.) are two examples of MBD platforms. Unlike traditional code handwriting and debugging methods, MBD graphical modeling reduces the complexity of the mathematical equation representation of a system, and code generation avoids the transcription process to programming languages, which may contain errors that are difficult to detect. In this way, the task of designing a control system can be simplified to create a graphical model and validating it with appropriate tests and simulations. The remaining processes, including code generation, validation, and compilation, are automated using MBD platform software. In the development phase, C code handwriting, verification, and testing are time consuming, especially as the target system becomes more complex. These steps can be skipped with the Code Generation function in the MBD method to save time. For example, using the Simulink Coder Generation toolbox,

one can easily generate C code from models designed with Simulink blocks. Additionally, error detection, model revise and mathematical equation details are easier to handle from the high-level perspective of graphical modeling. According to [6], using the MBD method and code generation, Alstom France designed the Propulsion control system, saving 50% development time, and Honeywell Aerospace USA designed the Flight control system, saving 60% development time. Thus, Simulink is chosen as the MBD platform in this work to build a new AGV control system prototype capable of enabling vehicles to move autonomously in a short time.

Robot operating system (ROS) is free and open-source software under the BSD license. Presently, ROS has been widely used by several companies and universities [7], such as Yujin Robots, Clearpath Robotics, and JAXA. ROS comprises many open-source robotics libraries for dealing with robotics tasks, including sensor driver libraries, simultaneous localization and mapping (SLAM), robot navigation, and motion control. The basic method of communication in ROS is through messages held by topics based on the TCP/IP layer. Many small programs (called nodes) can be executed concurrently and assigned to different machines, enabling one to construct a distributed computation system [8]. Thus, we select ROS as the middleware for developing our application.

This paper is organized as follows. In Section 2.1, we describe the hardware construction of the AGV used in this research in detail. In Section 2.2, we describe the autonomous driving control framework of the AGV on the basis of ROS and Simulink. In Section 2.3, we discuss available autonomous driving technologies to be used for mobile robots such as SLAM and navigation methods. In Section 2.4, we show the results of AGV autonomous driving at Tokyo International Robot Exhibition 2019 and discuss the obtained results. In Section 2.5, we conclude the knowledge obtained through this research and present future research plans.

2.1 AGV Hardware Construction

Fig. 1 shows the layout and main components of the AGV developed in this research. The width and length are 0.66 and 0.79 m, respectively. We use the all-wheel driven type with mecanum wheels as a driving system, which can realize vehicle movement in any direction and rotation at the center of the vehicle. The movement part of the AGV consists of four mecanum wheels (OD: 150 mm) units and four motors. Each mecanum wheel is driven by one 24 V AC servo motor with an assembled gearbox (Reduction Ratio 1:50) independently (FHA-14C-50-E200-CEK. Harmonic Drive Systems Inc). Each AC servo motor can achieve Max 15.5 Nm @ Max 120 r/min. Every motor is equipped with one 16-bit encoder and is driven by one dependent AC servo driver (HA-680-6-24. Harmonic Drive Systems Inc). Four drivers receive commands and send feedback to the four-axis controller board (NPMC6045A-4104B), and the controller board communicates with one LF64 MCU board via PC/104 bus, as shown in Fig. 1.

Regarding sensors assembled into the AGV, one stereo camera (Intel Real Sense D435i [9]) is installed on the front of the vehicle's body, which collects RGB images and depth (called point cloud) information simultaneously. A six-axis inertial measurement unit sensor is internally equipped with Intel Real Sense. Consequently, the measurements of the accelerator and gyroscope for the x, y, and z axes are available. Additionally, four ultrasonic sensors are installed in the front of the vehicle. They aid in the detection of obstacles when the external light is dim or at a blind corner. Considering physical protection for the AGV, two touch sensors are integrated into the vehicle's front frame to prevent collision.

We use two computers as the brain part of the AGV for this framework: one is Intel NUC7, and another is NVIDIA Jetson Xavier. Both are running on Linux 18.04 LTS and ROS Melodic. The following are the main functions of Intel NUC7: (1) collection and management of sensor data such as RGB-D sensor and ultrasonic sensor data, (2) SLAM, (3) navigation functions (global and local planner), (4) sending commands to LF64 MCU board through USB serial communication, (5) sending RGB image topics to NVIDIA Jetson Xavier and receiving object detection results, and (6) receiving

commands from the joystick in manual mode via Bluetooth. NVIDIA Jetson Xavier [10] currently performs only object detection because of its powerful GPU capabilities (512 CUDA cores inside).

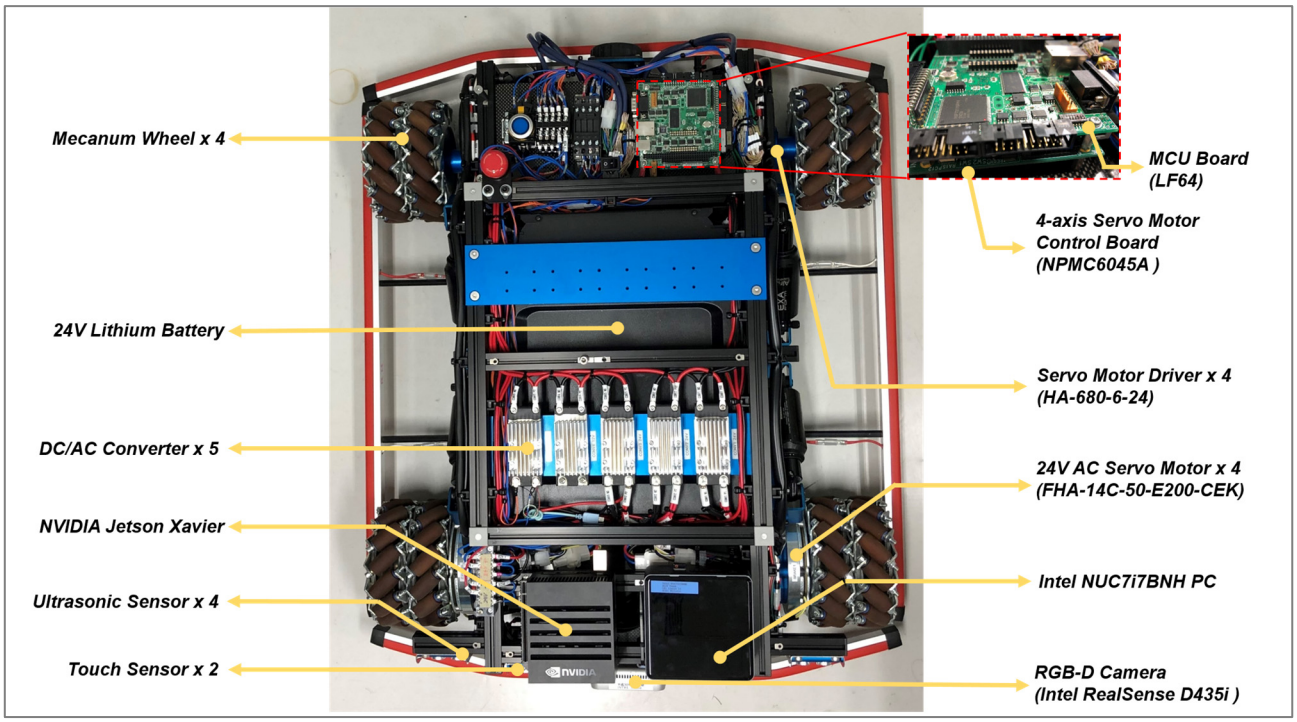

Fig. 1. AGV Layout

2.2 AGV Driving Control System

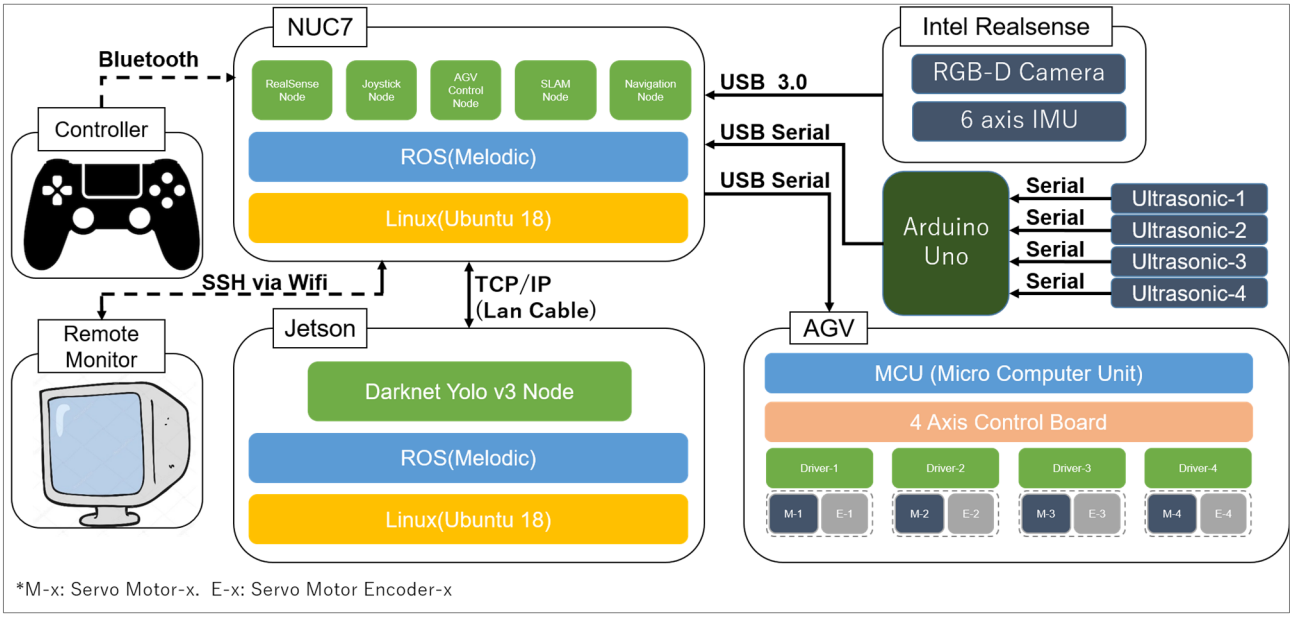

Fig. 2. Framework of AGV Based on ROS

Fig. 2 shows the hardware-based control system of the AGV. Here, NUC7 is used as the hub and core part of the entire system, which is also the ROS master. It enables the transfer of vehicle

movement commands from a controller in manual mode or a motion planer in autonomous mode through USB serial communication. Four ultrasonic sensors are connected to one Arduino Uno board and communicate with the ROS master through a USB. Arduino Uno calculates the distance using the information collected by each ultrasonic, then packs the results into ROS messages, and sends these messages to the ROS master in a topic format. RGB-D sensor data from Intel Realsense, including RGB images, point clouds, and infrared ray images, will also be sent to the ROS master. RGD images will be transferred from NUC7 (the ROS master) to Xavier for object detection, and point cloud data will be transformed into laser scan data for creating 2D maps. In manual mode, a human operator can control the vehicle's movements with a SONY PS4 controller that is connected to NUC7 via Bluetooth. In most cases, the human operator is required to monitor statuses, such as the camera's working status and network communication status, for the entire AGV system. In this research, Secure Shell is used as the remote control terminal.

2.2.1 AGV control node by implementing Simulink and Stateflow.

Simulink is a commercial GUI programming tool developed by MathWorks, Inc., which includes a variety of toolboxes, and in each toolbox, there are several high-level blocks such as PID and State Estimator. Simulink is widely used as an MBD platform by mobile vehicle manufacturers such as Toyota, Honda, and Subaru. In this paper, we use the ROS toolbox in Simulink to communicate with the ROS network by subscribing to a sensor and publishing AGV motion control topics. Simulink's modeled blocks' simplicity, dependability, and convenience allow developers to concentrate on their primary tasks. Stateflow is another tool embedded in Simulink used to model reactive systems using state machines and flow charts. Especially, when designing complex logic models, Stateflow is much easier to build, understand, and debug than traditional code-style programs such as C. In this paper, we combine Simulink and Stateflow to construct the AGV autonomous driving control node. Fig. 3 shows the Simulink model.

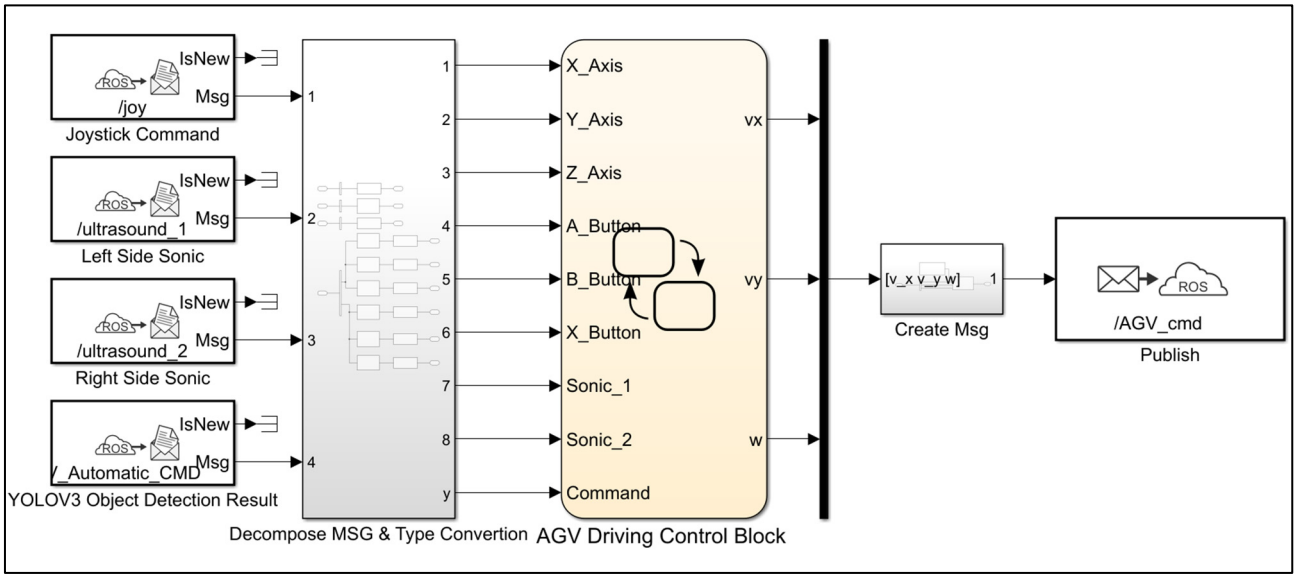

Fig. 3. AGV Control Node Simulink Model

The AGV control Simulink node can be divided into five parts from left to right:

1) The sensor data receiving part: this is used to receive ROS topic messages such as Joystick and ultrasonic sensor signals by subscribing to related ROS topics.

2) Decomposition of message data and type conversion: this is used to extract necessary data and convert data types for the next process. For example, in the Joystick command, there are more than 15 buttons or axes exist; thus, we must extract the ones we want to use. Regarding data type conversion, the extracted data are of the float type, but Stateflow handles data as the double type in default, so we must convert the data type from float to double using the double block.
3) Logic Block for the Main AGV Driving Control Block: this was created by Stateflow as the core logic part, including AGV body orientation calibration, emergency mode, and recovery mode. Section 2.2.2 will provide a more detailed description.
4) Creation of ROS message: this is used to create ROS messages by combining one empty message with geometry_msgs/Twist type and vx, vy, and w.
5) Packing and publication of AGV control messages: this is used to publish AGV_cmd topic into the ROS network with the msg created in 4.

2.2.2 AGV autonomous driving control logic by implementing Stateflow.

Simulink and Stateflow are common methods used for implementing the logic control of MBD. Simulink is available through relational operator or logic operator. However, Stateflow is more suitable for complex logic control since it is equipped with state transition diagrams (STDs), state transition tables (STTs), and truth tables. An STD is a type of diagram (as shown in Fig. 4) that comprises *state*, *transition*, and *action*, enabling the behavior of systems to be described dynamically. An STT can be used to express sequential modal logic (as shown in Table 1 for the logic transition of control modes). Instead of drawing a diagram, an STT can be used to express state, transition, and modal logic. Users can monitor the current state of a system and visually analyze logic errors. Refer to [11] for detailed usage information about Stateflow.

The video [12] confirms the AGV walking situation captured at Tokyo International Robot Exhibition 2019.

Table 1 State transition table for AGV operation mode change

State	Label		Item	Contents
Manual	T01		Transition condition	Press Button B
		A01	Condition action	-
		→	Destination state	Automatic
	T02		Transition condition	Press Button A
		A02	Condition action	-
		→	Destination state	Emergency
Emergency	T03		Transition condition	Press Button X
		A03	Condition action	-
		→	Destination state	Manual
	T04		Transition condition	Press Button B
		A04	Condition action	-
		→	Destination state	Automatic
Automatic	T05		Transition condition	Press Button X
		A05	Condition action	-
		→	Destination state	Manual
	T06		Transition condition	Press Button A
		A06	Condition action	-
		→	Destination state	Emergency

Fig. 4 shows the state diagram of AGV logic control based on Table 1. In this paper, Manual, Emergency, and Automatic are three parent states, which also represent the three control modes, as stated in the Introduction section. The default state is the manual mode, which enables a user to control the AGV using a joystick. When a user presses buttons A and B separately, the emergency and automatic states are switched. In the automatic state, the AGV operates as follows:

1) The AGV goes into *init* state in default.
2) When the AGV receives an executable command from the YOLO v3 node, it will go into *MoveForward* state after a 3 s temporal logic.
3) In the *MoveForeward* state, the AGV will move forward while calibrating its head angle using data from both the left and right ultrasonic sensors simultaneously. This function is described during the action section. To prevent the AGV from colliding with the wall, a state transition condition of $\frac{Sonic_1+Sonic_2}{2} <= 35$ [cm] is used.
4) On the basis of the detected command, the AGV will execute three different actions, *CCW_90deg*, *CW_90deg*, and *CCW_90deg*, corresponding to the trump card numbers 1, 2, and 3 individually.
5) After executing the rotation command at 4, the AGV's head may not be perpendicular to the wall in the front of it because of external noise such as ground flatness and the coefficient of friction. To avoid collision with objects while moving along a planned path, the AGV's head direction will be calibrated in the state *Posture_Correction*.
6) When the state *Posture_Correction* satisfies the transition condition of $abs(Sonic_1 - Sonic_2) < 0.07$[cm], the state returns to *init*, and the loop continues until the A or B buttons are pressed.

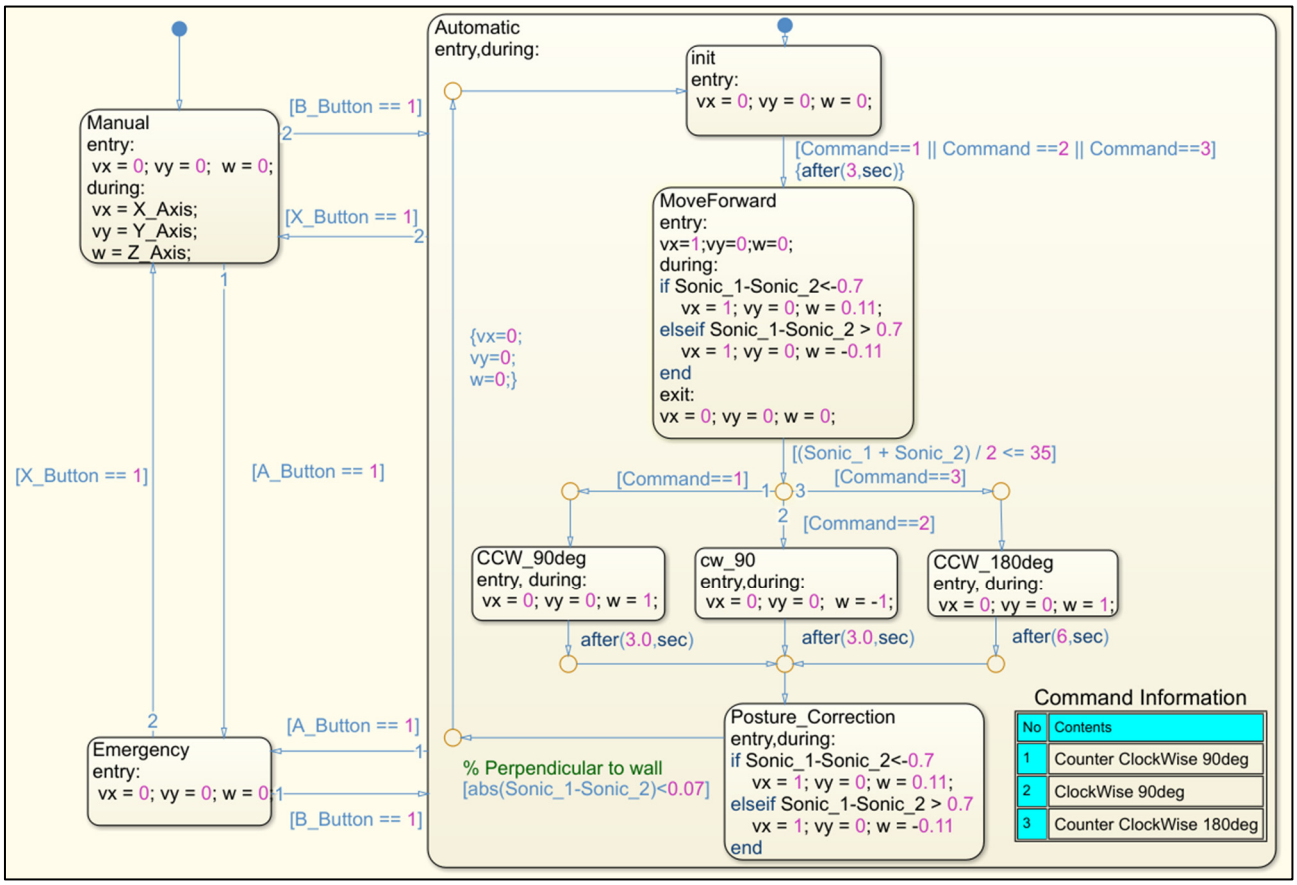

Fig. 4. State Diagram of AGV Logic Control

2.3 Autonomous Driving Technology

SLAM, object detection, and path planning are key technologies to realize autonomous driving for mobile robots. These technologies are used in this research and are described below in detail. SLAM is a branch of mobile robotics that studies how to create a map and localize the robot's position. The

main idea of the SLAM technique is to let a robot move and build a map of its surroundings simultaneously when it is placed in an unknown environment. There are several methods of SLAM available in ROS, including "gmapping" and RTAB. In this research, we use the gmapping method to generate a 2D map, which was developed on the basis of the Rao-Blackwellized particle filter and introduced by Murphy and Doucet in detail [13, 14].

The ability of a robot to recognize its surroundings is far more important for improving its ability to deal with external obstacles such as moving humans and moving vehicles, word comprehension, and reading traffic signals. To this end, we use an open-source object detection method called YOLO v3. In this paper, we incorporate YOLO v3 into ROS and perform object detection tasks on Xavier via the distributed system of ROS. Detailed information regarding YOLO v3 can be obtained in [15].

A robot understands only map creation and robot localization, and it has no idea how to get from a starting point to a target or how to navigate. In this study, the global planner node designs the necessary path for a robot. The global planner is based on the A* search algorithm. In most cases, a robot must also detect and avoid obstacles, and this can be achieved by the local planner. We use the package "move_base" to achieve autonomous navigation. The move_base package integrates the global planner and the local planner on the basis of the dynamic window approach [16].

2.4 Test Results and Discussion

To meet various and flexible requests during the plant and debug phases, we propose three control modes: manual, teaching, and autonomous. The manual mode is analogous to a human operator operating a vehicle. The human operator holds a controller and a monitor, which are used to capture the front scene through or behind the vehicle. In the teaching mode, the AGV moves autonomously along a path preprogrammed in the manual mode, which is achieved using the "rosbag" function in ROS. For the autonomous mode, after a human operator has set the start and goal points, the vehicle automatically plans the shortest path and detects and avoids static or dynamic obstacles. This mode integrates several packages such as "GMAPPING," "MAP SERVER," "AMCL," "MOVE_BASE," "JOY," "REALSENSE," "ROSSERIAL_ARDUINO," "RVIZ," and "DARKNET_ROS."

Presently, we have realized the vehicle's movement in the manual and teaching modes. This demonstrates that ROS is a useful and convenient operating system to manage sensor drivers, network communication, data logging, and visualization. Fig. 5 shows the architecture of nodes and topics of the teaching mode. Additionally, test results at the manual and teaching modes are available on [17].

For the autonomous mode, object detection has been realized, and SLAM and navigation systems are under the debugging phase. The image recognition model is used in this study is trained from scratch. The dataset used for training comprises 90 actual images. Five classes must be recognized: "go," "back," "left," "right," and "stop." After training, we use a new image to test the performance of the model. Fig. 6 shows object detection results. Except for the back class whose precision is 74%, the precision for others is 100%. Although we achieve 100% precision for these samples, we discover that the robustness of the current model is insufficient, as it occasionally misjudges objects such as whiteboards. This could be because the number of images in the current dataset, 90, is insufficient for training compared with, for example, the well-known MNIST dataset, which comprises 70,000 images with 10 classes.

Regarding object detection, NUC7 sends image topic/image to Xavier via a LAN cable; the node/darknet_ros detects the object in the sent topic and publishes the /detection_image detection results into ROS for use by other nodes such as control and rviz. According to test results, the time lag is 0.5 s and the detection speed is 5.2 fps. We have not yet determined how time lag and detection speed affect the AGV control system and safety, and we believe that it is necessary to do so in future works.

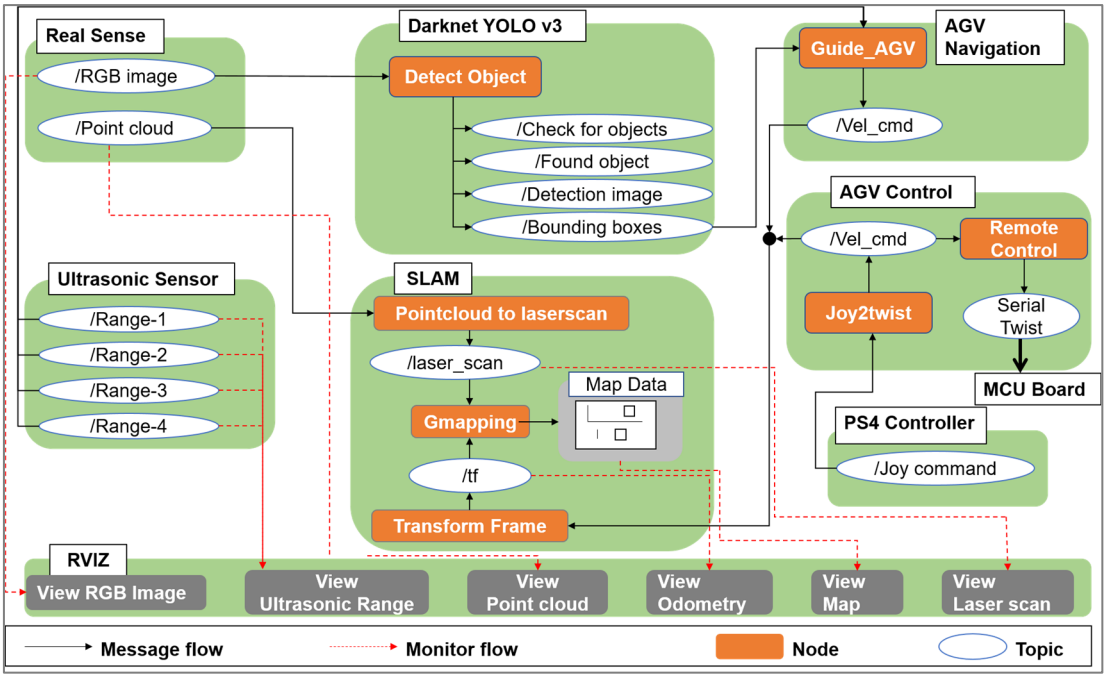

Fig. 5. ROS Node and Topic Architecture in Teaching Mode

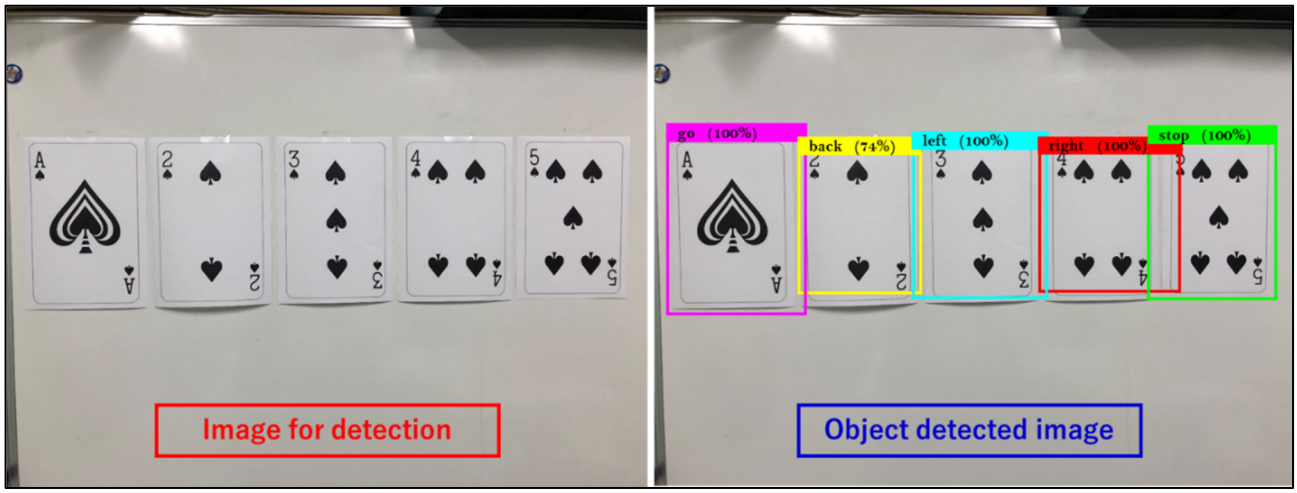

Fig. 6. Objection Detection Result by using Pretrained Model

3. Conclusions

In 3 months, we successfully built an autonomous driving framework for an AGV using the MBD method on the basis of ROS and Simulink and confirmed that ROS is an efficient integrated operating system that can deal with sensor drivers, sensor message transformation, visualization of results, and task distribution. The ROS toolbox in Simulink has proven to be user-friendly, simple, and convenient.

Stateflow, as an embedded logic control tool in Simulink, enables us to perform the following: 1) easily construct the AGV control logic using an STD and STT and 2) monitor the current status of the system and visually analyze logic errors.

On the basis of ROS, we controlled and tested the AGV in the manual and teaching modes. The Darknet YOLO v3 algorithm was incorporated into ROS and used to realize objection detection via the distributed system. Presently, work on AGV self-position prediction using the Kalman filter or particle filter; integration of SLAM and navigation node, including automatic path generation and local

and global planners; and automatic C/C++ code generation from the Simulink Model are ongoing and should be completed soon to realize a higher autonomous mode.

Acknowledgments

We gratefully acknowledge technical supports about AGV vehicle from Ueno Technica Corporation Japan.

References

[1] S. I. Tay, T. C. Lee, N. A. A. Hamid and A. N. A. Ahmad, "Industry 4.0", ***Journal of Advanced Research in Dynamical and Control Systems***, Vol.10, No.14-Special Issue, pp.1379-1387, 2018.

[2] R. Walenta, T. Schellekens, A. Ferrein and S. Schiffer, "A Decentralised System Approach for Controlling AGVs with ROS", ***Proceedings of 2017 IEEE AFRICON*** (Aachen, Germany) September 2017.

[3] ***Model-Based Design for Embedded Systems***, Nicolescu, G., & Mosterman, P. J., CRC Press (Boca Raton, Florida, USA), 2018.

[4] MathWorks, Model-Based Design: https://www.mathworks.com/help/simulink/gs/model-based-design.html

[5] Broy, M., Kirstan, S., Krcmar, H., & Schätz, B., "What is the benefit of a model-based design of embedded software systems in the car industry?" in ***Emerging Technologies for the Evolution and Maintenance of Software Models***, IGI Global (Hershey, Pennsylvania, USA), pp. 343-369, 2012.

[6] MathWorks, Production Code Generation and Verification Using Simulink and Embedded Coder: https://www.youtube.com/watch?v=cuZj_hvVo_4&t=2211s

[7] H. Zhang, K. Watanabe, K. Motegi and Y. Shiraishi, "ROS based framework for autonomous driving of AGVs", ***Proceedings of ICMEMIS 2019*** (Kiryu, Japan) December 2019.

[8] Documentation - ROS Wiki: http://wiki.ros.org/

[9] Depth Camera D435i – Intel® RealSense™ Depth and Tracking Cameras: https://www.intelrealsense.com/depth-camera-d435i/

[10] AI-Powered Autonomous Machines at Scale | NVIDIA Jetson AGX Xavier: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-agx-xavier/

[11] Stateflow: https://jp.mathworks.com/help/stateflow/index.html?s_tid=CRUX_lftnav

[12] AGV autonomous driving with YOLO V3 Object Detection technology: https://www.youtube.com/watch?v=SsRXfIZGVo8

[13] K. P. Murphy, "Bayesian Map Learning in Dynamic Environments", ***Proceedings of the Neural Information Processing Systems:*** NIPS1999, (Denver, CO, USA) December 1999.

[14] A. Doucet, N. D. Freitas, K. P. Murphy and S. J. Russell, "Rao-Blackwellised Particle Filtering for Dynamic Bayesian Networks", ***Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*** (Stanford, CA, USA) July 2000.

[15] Redmon, J., & Farhadi, A. (2018). Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767. Redmon, J., & Farhadi, A., "YOLOv3: An Incremental Improvement", ***ArXiv***, abs/1804.02767, 2018.

[16] D. Fox, W. Burgard and S. Thrun, "The dynamic window approach to collision avoidance", ***IEEE Robotics & Automation Magazine***, Vol.4, No.1, pp.23-33, 1997.

[17] Autonomous AGV driving at manual and teaching mode: https://youtu.be/nK8DDsg-yv8