Journal of Korea Robotics Society (2020) 15(4):375-384
https://doi.org/10.7746/jkros.2020.15.4.375 ISSN: 1975-6291 / eISSN: 2287-3961

이종의 제어 플랫폼들로 구성된 로봇 시스템을 ROS 기반의 시스템으로 손쉽게 통합하기 위한 소프트웨어의 개발

SW Development for Easy Integration of Robot System Composed of Heterogeneous Control Platforms into ROS-based System

강 형 석[1] · 이 동 원[2] · 신 동 헌[†]
Hyeong Seok Kang[1], Dong Won Lee[2], Dong Hun Shin[†]

Abstract: Today's robots consist of many hardware and software subsystems, depending on the functions needed for specific tasks. Integration of subsystems can require a great deal of effort, as both the communication method and protocol of the subsystem can vary. This paper proposes an expandable robotic system in which all subsystems are integrated under Robot Operation System (ROS) framework. To achieve this, the paper presents a software library, ROS_M, developed to implement the TCP/IP-based ROS communication protocol in different control environments such as MCU and RT kernel based embedded system. Then, all the subsystem including hardware can use ROS protocol consistently for communication, which makes adding new software or hardware subsystems to the robotic system easier. A latency measurement experiment reveals that the system built for loop control can be used in a soft real-time environment. Finally, an expandable mobile manipulator robot is introduced as an application of the proposed system. This robot consists of four subsystems that operate in different control environments.

Keywords: ROS, Node, Framework, Protocol, TCP/IP

1. Introduction

Robots that offer high levels of service are composed of various subsystems such as software modules, sensors and actuators. However, integrating subsystems is not easy because their communication methods and protocols tend to differ from each other. In addition, the circumstances and tasks of robots vary so considerably that the ability to add and remove subsystems to cope with these changes is required.

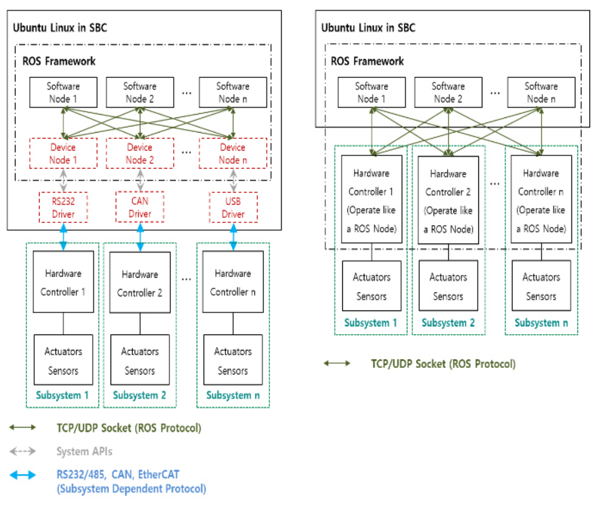
In this paper, we propose an expandable robotic system to address this laborious integration problem. The proposed system is based on the communication method of ROS[1], which is a de facto standard robotic software framework[2-6]. The ROS protocol is a TCP/IP-based communication that informs the IP address and the provided functions between nodes, which means the primitive unit of the ROS framework. However, ROS protocol is basically only available among ROS nodes which operate in ROS framework. So, there should be another method if ROS system needs to communicate with non-ROS system outside. There had been previous researches in order to interconnect ROS and non-ROS systems. Rosbridge is one of the official packages in ROS framework, which is the first solution developed to solve this interconnection problem[7]. Rosbridge is a special node operating in ROS framework which receives and sends JSON based ROS functionality messages through web socket server. It receives JSON message from non-ROS system which contains

Received : Jun. 12. 2020; Revised : Aug. 7. 2020; Accepted : Sep. 16. 2020
※ This paper was supported by the research grant of University of Seoul

1. Principle Researcher, MINTROBOT Co., Ltd., Seoul, Korea (justinhskang@gmail.com)
2. Researcher, MINTROBOT Co., Ltd., Seoul, Korea (dongwon0409@naver.com)
† Professor, Corresponding author: Mechanical and Information Engineering, University of Seoul, Seoul, Korea (shin@uos.ac.kr)

functional information of ROS framework. Then it translates the received message into ROS protocol and sends it to ROS nodes. Thus, it operates as a bridge to interconnect ROS system and non-ROS system. The system that communicates with Rosbridge naturally needs to prepare a JSON parser and websocket protocol which are overheads. And because of these shortcomings, the use of Rosbridge is very limited. The most generally use cases of Rosbridge are a communication bridge between ROS system and web browsers which can visualize the specific information in the ROS system or interface the intension of users to ROS system[8-10]. It can hardly be used for control purpose which must reduce overheads as much as possible for its fast response.

The system that controls actuators and sensors normally operate under MCU like firmware environment. In the past, implementing TCP/IP communication in a restricted environment such as an MCU platform was not easy with respect to technology and cost because an MCU was insufficient for processing transmitted data packets concurrently at high frequencies. Thus, Simpler communication methods such as RS232/485 then became widely used to link a top-level control system and subsystems[11-13]. However, since the technological advances have led many available solutions, TCP/IP communication can now be easily implemented even in an MCU platform. Thus, communicating with an ROS framework by using an ROS communication method even in an MCU environment is possible. [Fig. 1] shows the difference between the integration method of subsystems that use the conventional serial communication protocol and the system integration method



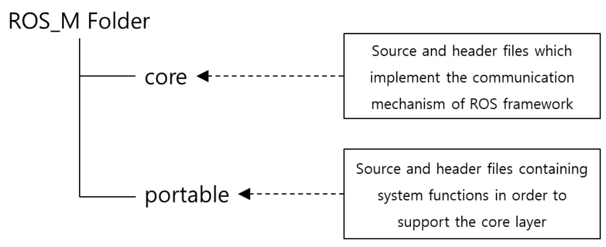
[Fig. 1] Difference between two robotic system structures in the ROS framework (left: Integrated system with serial protocols; right: Integrated system with TCP/IP based ROS protocol)

using the TCP/IP-based ROS protocol proposed in this paper. Since the original ROS framework is installed only on Linux-based systems, the structure of the system shown in [Fig. 1(b)] requires the ROS protocol stack that is available on MCU platforms, and several studies have been conducted to address this issue. Migliavacca developed µROSNode, a software library that provides the core functions of the communication mechanism of ROS and enables hardware to communicate with the node in the ROS system without having to install an ROS framework[14]. Also, there exists several implementation studies by using developed µROSNode[15-17]. However, because only one hardware node of the Cortex-M4 MCU/DSP-based platform was implemented in the aforementioned study, additional effort is required to apply this library to other platforms. Above all, even though the specific use cases of µROSNode were controllers, the previous researches didn't show the feasibilities in order to use the developed method for a control system, such as latency performance to determine control loop period. Recently, Quigley introduced a small embedded system for ROS2, which has since been developed as a next generation of ROS[18]. His embedded system uses a UDP/RTPS-based communication protocol suitable for the system architecture of an ROS2. However, because the newly distributed architecture of the ROS2 is quite different from the existing master/slave architecture of the ROS, the robot systems based on ROS could have difficulty migrating to this new ROS2 environment.

This study introduces ROS_M library that we developed to realize the system structure shown in [Fig. 1(b)]. ROS_M is a software library toolkit that helps implement computer program that behaves like a ROS node without the installation of the original ROS framework. It can communicate with other ROS nodes in the system by providing functions to construct topic and service transmission mechanisms for the ROS framework within very fast response enough to construct closed control loop. Furthermore, ROS_M has improved in terms of portability as compared to previous studies. Because ROS_M is designed by considering porting to a wider variety of environments, it can be easily ported to an RT-kernel-based embedded system or even a Windows-based general system. Therefore, not only MCU-like platforms, motion controllers that require real-time capabilities or tablet PCs used for playback of content can be recognized as ROS nodes and integrated into an ROS framework. We first present the structure of our ROS_M library and implementation details related to the manner in which the ROS_M library communicates with other ROS nodes. We then show the results of a latency measurement experiment to prove that how much feasible the developed ROS_M library is for control system, in which we also considered the effect on using ethernet switches required to expand the subsystem, a topic that has not been reviewed in previous studies. Finally, an expandable mobile manipulator robot which has several subsystems operating under different platform respectively is introduced as an implementation. The experimental robot was composed of four subsystems, namely, those for the head, arm, body, and mobility, each with a different control platform. Each subsystem communicates each other by using only ROS protocol consistently and enough fast to make closed control loop among subsystems in 1-kHz control period.

2. ROS_M

ROS_M[1)] is a software library that implements topic and service communication mechanisms of ROS framework. Because it is written in only ANSI C language without any support utilities such as a XML parser or STL in C++, it is sufficiently light to be ported in any platform. ROS_M enables implementing a computer program that can operate like an ROS node, thus allowing for communication with other ROS nodes over networks. Therefore, the ROS_M library is extremely useful for developing distributed subsystems in an ROS framework (including networks) if the subsystems cannot install standard ROS packages due to system limitations. [Fig. 2] presents the structure of our ROS_M library. The ROS_M library consists of two layers: core and portable. The core layer implements the communication mechanism of the ROS framework. It implements the TCP/IP server capability, which is required for communication with the ROS Master and other ROS nodes in the network. In addition, the data structures of the topic/service and their transmission mechanisms are implemented. Users do not need to modify the core layer, even though they must port the ROS_M library into other platforms. This is because the core layer references only the functions in the portable layer. The portable layer implements the system functions such as threading to support the functions in the core layer. The essential features of the ROS_M library are as follows.


ROS_M Folder
core
portable
Source and header files which implement the communication mechanism of ROS framework
Source and header files containing system functions in order to support the core layer

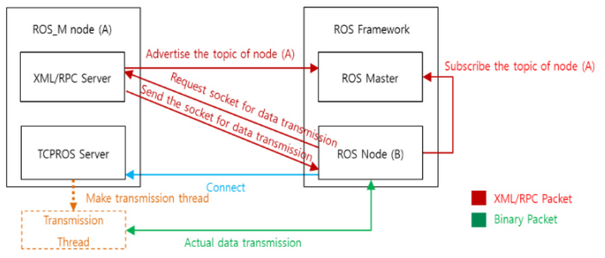
[Fig. 2] ROS_M Library structure

1) The source codes of ROS_M library are available from https://github.com/mintrobot/ROS_M.

2.1 Master and Slave APIs handling

The ROS Master of the ROS framework manages node information such as the node name, provided topic and service, IP address, port number, and others. It collects information about nodes when the nodes are initialized. Then, it provides the proper information to the other nodes by using the collected information. The ROS framework uses XML/RCP packets for these communication functions, which are known as *ROS Master* and *Slave APIs*.

These functions are necessary to make a P2P connection between nodes for the purpose of data transmission. The ROS framework uses a binary packet for data transmission because it is faster than the text-based XML/RCP packet. If a node receives the proper information from the ROS Master for data transmission, the socket connection between nodes is created and the actual data transmission, such as a publish/subscribe topic or request/response service, is started. To satisfy this mechanism, ROS_M provides two servers: XML/RCP for the *ROS Master* and *Slave APIs*, and TCPROS for creating a P2P connection between nodes. [Fig. 3] shows the data transmission between the ROS and ROS_M nodes, which is the computer program that operates like an ROS node by the ROS_M library. The XML/RPC server in the ROS_M node (A) sends and receives the XML/RPC packets to and from the ROS Master in the ROS framework. Then, the ROS node (B) receives the proper information from the



[Fig. 3] Example of data transmission between ROS_M and ROS nodes

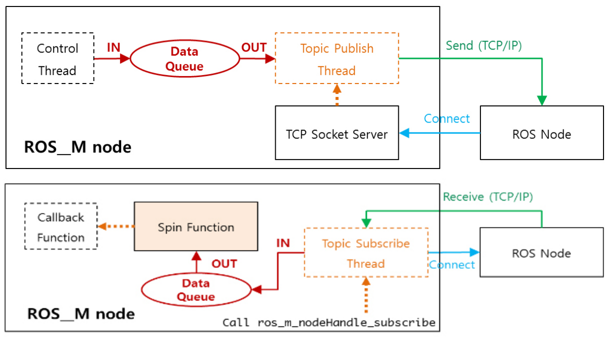
ROS Master to connect to the ROS_M node (A). It then connects to the TCPROS server in the ROS_M node (A) to make the P2P connection for data transmission.

2.2 Publish/Subscribe and Request/Response mechanism

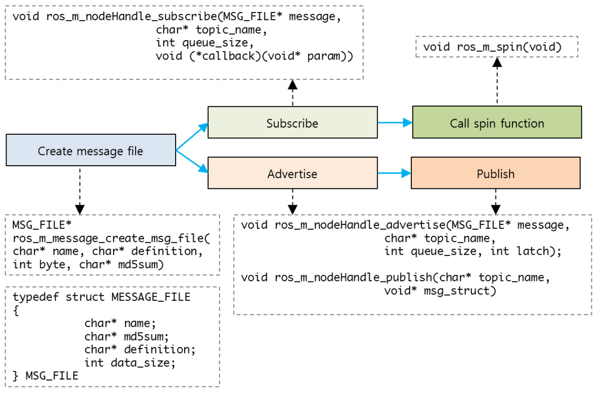
Publish/subscribe topic and request/response service are the main transmission mechanisms of the ROS framework. Publish/subscribe topic mechanism is used for sharing specific variables among nodes. The shared variable is called a topic. A node in the ROS system can share the value of some variables owned itself by sending the value to the other nodes who are interested in. This is the publish topic mechanism. However, the node sends the value of the variable only if when it has received the request of sending from the other nodes. This is the subscribe topic mechanism.

On the other hand, request/response service mechanism is used for executing specific functions owned in other nodes. The executed function is called a service. A node can execute some function implemented in the other nodes by sending execution request packet. This is the request service mechanism. Also, the node can receive the result of the executed function by waiting receive result packet. This is the response service mechanism. Those mechanisms are also implemented in the ROS_M library, applying the same principle as that of the ROS framework but using a different approach for added portability. [Fig. 4] shows how the ROS_M library handles topic transmission. First, the message file structure, which describes the structural information of the transmission data, must be instantiated. This information is described in the *.msg file in the original ROS framework. Then, it advertises the topics that it publishes to the ROS Master in the other platform. Therefore, the ROS nodes in other platforms can connect to the ROS_M node in order to subscribe the topic that the ROS_M node publishes. Once the ROS node connects to the ROS_M node, the thread for publishing the topic is generated and the actual data transmission proceeds. In addition, the ROS_M node can connect to the ROS node in order to subscribe the topic that the ROS node publishes. In the same manner as with the ROS node, the thread for subscribing a topic is generated after the ROS_M nodes connect to the ROS node. Then, the data transmission proceeds. The thread receives the data from the ROS node and pushes the received data into the data queue. Then, if the ROS_M node calls the spin function, that function pulls the data from the data queue and executes the callback function with the acquired data. [Fig. 5] represents the detailed process of the topic transmission in the ROS_M library.

The implementation of a request/response service mechanism is similar to a publish/subscribe topic mechanism. [Fig. 6] shows how the ROS_M library handles a service transmission. First, the ROS_M node instantiates the service file by using the structure type, which is defined by using the *.srv file in the original ROS framework. It advertises the generated service information to the ROS Master in other platforms. The ROS nodes in other platforms connect to the ROS_M node to request the service advertised. Then, the service response thread is generated and calls the actual service function using the received parameters from the ROS node. Finally, the service response thread receives



[Fig. 5] Details of transmission (up: publish; down: subscribe)



```
void ros_m_nodeHandle_subscribe(MSG_FILE* message,
                      char* topic_name,
                      int queue_size,
                      void (*callback)(void* param))
```

```
void ros_m_spin(void)
```

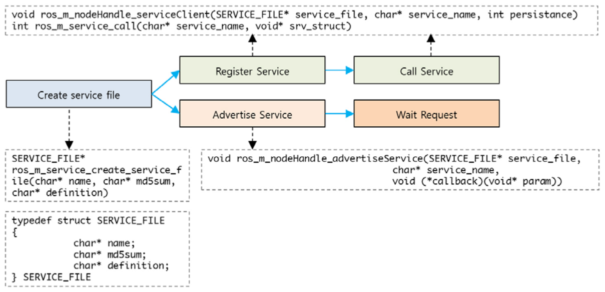
```
MSG_FILE*
ros_m_message_create_msg_file(
char* name, char* definition,
int byte, char* md5sum)
```

```
typedef struct MESSAGE_FILE
{
           char* name;
           char* md5sum;
           char* definition;
           int data_size;
} MSG_FILE
```

```
void ros_m_nodeHandle_advertise(MSG_FILE* message,
                      char* topic_name,
                      int queue_size, int latch);

void ros_m_nodeHandle_publish(char* topic_name,
                      void* msg_struct)
```

[Fig. 4] Publish/Subscribe topics in the ROS_M library



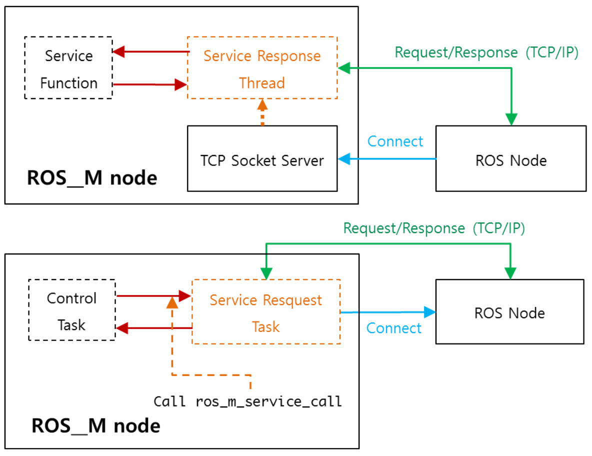
```
void ros_m_nodeHandle_serviceClient(SERVICE_FILE* service_file, char* service_name, int persistance)
int ros_m_service_call(char* service_name, void* srv_struct)
```

```
SERVICE_FILE*
ros_m_service_create_service_f
ile(char* name, char* md5sum,
char* definition)
```

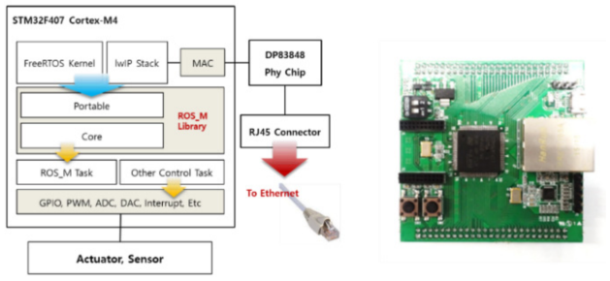
```
typedef struct SERVICE_FILE
{
          char* name;
          char* md5sum;
          char* definition;
} SERVICE_FILE
```

```
void ros_m_nodeHandle_advertiseService(SERVICE_FILE* service_file,
                          char* service_name,
                          void (*callback)(void* param))
```

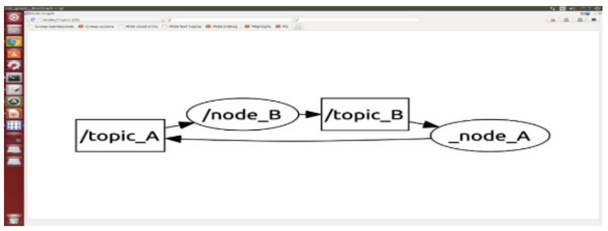
[Fig. 6] Request/Response services in ROS_M library



[Fig. 7] Details of service transmission (up: response; down: request)



[Fig. 8] Implemented ROS_M node in an MCU-based platform



[Fig. 9] Cooperation of ROS and ROS_M nodes

the result from the service function, then sends the result to the ROS nodes. Meanwhile, ROS_M node generates the service request thread when it needs to request the service from the ROS node in other platforms. The service request thread connects to the ROS node and sends the request message with parameters.

Finally, the service request thread receives the result of the service from the ROS node and returns it to the control task. The service request/response threads sustain the connection if the persistence option is enabled. This option skips the handshake process in order to prepare the actual transmission by sending and receiving XML/RPC packets. [Fig. 7] illustrates the detailed process of the service transmission in the ROS_M library.

3. Latency Measurement Experiment

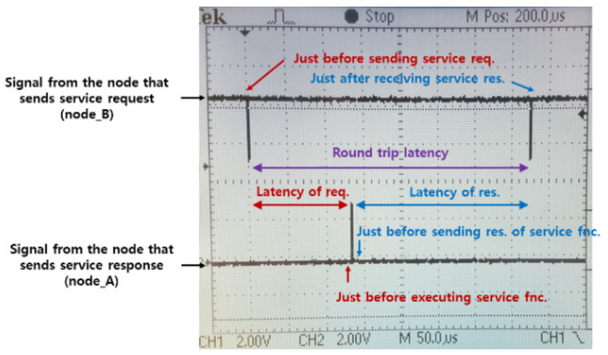
3.1 Experimental System

Latency is a critical feature in the control system of a robot that requires real-time capability and fast response. Because an ROS framework uses TCP/IP communication for distributed processing, an ethernet switch is required to connect subsystems to the network. Therefore, the latency among the nodes including the ethernet switches is crucial in constructing a trustable control system. In this experiment, we generate results based on a latency measurement test in a network environment that includes ethernet switches and ROS_M nodes based on an MCU/DSP platform, which is the most restrictive environment in which ROS_M nodes can operate.

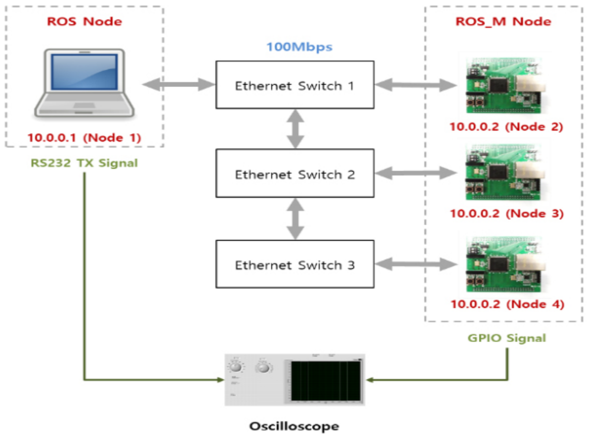
[Fig. 8] represents the ROS_M node implemented on an MCU/DSP platform, which assumes a controller of a subsystem. STM32F407 which has the Cortex-M4 core is used with a DP83848 ethernet PHY chip. Also, the FreeRTOS real-time kernel and the lwIP TCP/IP stack are ported to support the portable layer. The implemented ROS_M node connects to the ROS Master operating on the other platform and can communicate with other ROS nodes in the ROS framework. [Fig. 9] illustrates the two interfaced nodes in the ROS framework. Node_A is the ROS_M node operated on the MCU/DSP platform, and node_B is the ROS node operated in the PC platform with the installed ROS framework. Two nodes publish topic_A and topic_B, respectively. The node_A subscribes to topic_B by receiving node_B's IP address and port number from ROS Master running on the PC platform. Then, it connects directly to the node_B by using the received IP address and the port number, then starts receiving the topic. The process whereby node_B subscribes topic_A from node_A works in the same manner. However, the latency between nodes cannot be measured by using topic packets because sending and receiving of topic packets are processed by ROS Master which is the top-level controller of ROS framework. Therefore, we cannot know the exact moment at the node side when the topic packet had been sent and had been received. So, we used service packets instead in order to measure the latency between nodes because we can control the exact motion of sending and receiving service packets when we write application codes.

[Fig. 10] shows the captured signal from the node A and node

B. The node_B triggers a signal just after sending the request packet to the node_A. Then, the node_A triggers a signal just after receiving the request packet from the node_B. The exact moment of sending and receiving service packets is representable by triggering the signals in the application codes. Also, the triggered signals are captured by using an oscilloscope. The distance between triggered signals is the exact latency of request packet between nodes. The node_B triggers the signal again just after sending the response packet as the result for the requested service to the node_A. Then, the node_A triggers the signal just after receiving the response packet from the node_B. This is the exact latency of response from between nodes. Also, the sum of two latencies will be the round-trip latency which should be measured to determine the control loop period of the system. [Fig. 11] shows the experimental system for measuring communication latency when considering ethernet switches using the ROS_M node implemented in the MCU/DSP platform. The system has four nodes that are operated in their respective platforms. Node 1 is the standard ROS node operated on the PC platform as a software node, in which the ROS framework is installed. Nodes 2, 3, and 4 are the ROS_M nodes operated in the MCU/DSP platform as hardware nodes. Three ethernet switches are used to construct LAN. Nodes 1 and 2 are connected to switch 1, and nodes 3 and 4 are connected to switches 2 and 3, respectively. Every node in the system has a 10.0.0.x IP address. Each node then sends the service request packet to other nodes, and the nodes that received the service request packet send the response packet back. The latency between nodes is measured as shown in [Fig. 10]. The RS232/Tx pin of the PC and the GPIO pin of STM32F4 are used to send the signals to the oscilloscope, respectively. In our experiment, latency time was measured 10 times for each connection. In addition, only a four-byte payload was used for both the request and response, as it has already been proven that the size of the payload under MTU does not affect the increase in latency[19]. Software features such as the service persistence option and Nagle's algorithm were considered for increasing the transmission speed. [Table 1] lists the specifications of our experiment.



[Fig. 10] Captured signals from node B and node A



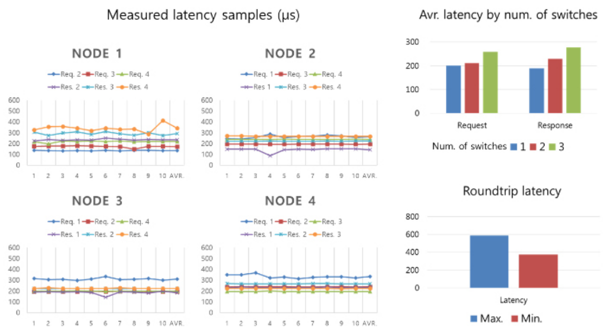
[Fig. 11] Experimental system for latency measurement

[Table 1] Specifications of the experimental system

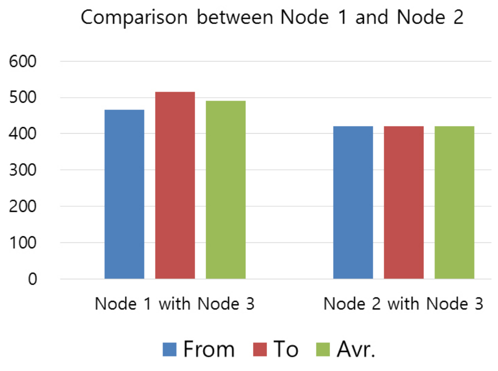
Feature	Remarks
Num. of samples	10 samples for each connection by direction (total 240 samples)
Ethernet switch	netis ST3105S (10/100 Mbps)
Cable	CAT.5e (length : 2m)
Payload	4 byte for the both request and response
Service persistence	Enabled
Nagle's algorithm	Disabled

3.2 Result

[Fig. 12] shows the results of our experiment. As can be clearly seen, the latency increased with the number of switches that the packets were required to pass through. The average increase in



[Fig. 12] Latency results of service request/response message



[Fig. 13] Comparison of responsiveness

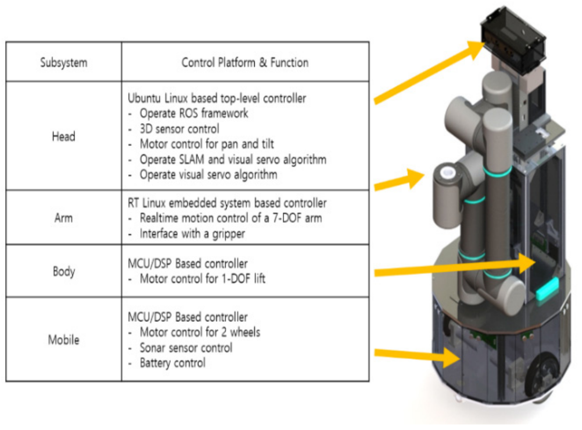
latency with the number of switches was 36.3 µs. The minimum and maximum latency of the round-trip was 372.9 and 587.2 µs, respectively. Thus, the 1-ms (1-kHz) control period for a loop control was available in soft real-time.

[Fig. 13] compares node 1/node 3 with node 2/node 3, which had the same number of switches that the transmitted packet was required to pass through. On average, the latency between the ROS_M nodes (node 2/node 3) was 70.7 µs lower than that between the ROS and ROS_M nodes (node 1/node 3). Therefore, the both data communication between ROS and ROS_M nodes and that between individual ROS_M nodes can be used for feedback control within a 1-kHz control period.

4. Expandable Robotic System for a Mobile Manipulator

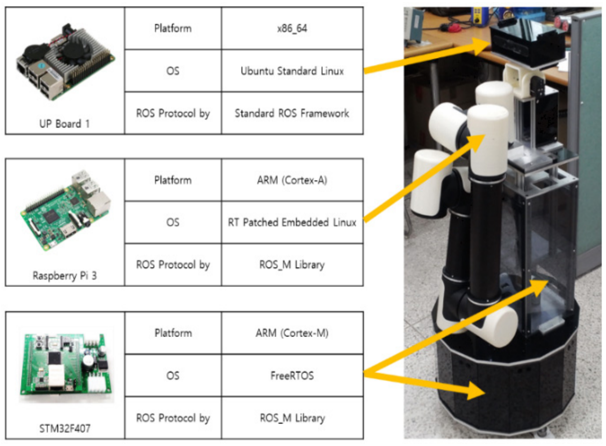
4.1 Expandable Mobile Manipulator

A mobile manipulator is normally used to handle sophisticated service tasks that require cooperation among several different functions such as locomotion, manipulation, grasping, and computer vision. These different functions normally operate in different platforms optimized according to their characteristics. For example, a vision recognition system requires very high computing power and video acceleration capability. On the other hand, a motion controller for a robotic arm requires real-time capable task management in order to calculate the angles of each joint at exact moment and send them to servo drives at the same time. Lastly, a low-level controller such as a servo drive requires digital signal processing capability such as PWM in order to control motors and sensor at the voltage- level side. The modularization of these functions can increase the convenience of system integration in



Subsystem	Control Platform & Function
Head	Ubuntu Linux based top-level controller - Operate ROS framework - 3D sensor control - Motor control for pan and tilt - Operate SLAM and visual servo algorithm - Operate visual servo algorithm
Arm	RT Linux embedded system based controller - Realtime motion control of a 7-DOF arm - Interface with a gripper
Body	MCU/DSP Based controller - Motor control for 1-DOF lift
Mobile	MCU/DSP Based controller - Motor control for 2 wheels - Sonar sensor control - Battery control

[Fig. 14] Modularized subsystems of the mobile manipulator



UP Board 1		
Platform	x86_64
OS	Ubuntu Standard Linux
ROS Protocol by	Standard ROS Framework

Raspberry Pi 3	
Platform	ARM (Cortex-A)
OS	RT Patched Embedded Linux
ROS Protocol by	ROS_M Library

STM32F407	
Platform	ARM (Cortex-M)
OS	FreeRTOS
ROS Protocol by	ROS_M Library

[Fig. 15] Constructed prototype and its controllers

terms of the increase of the reusability of modularized functions.

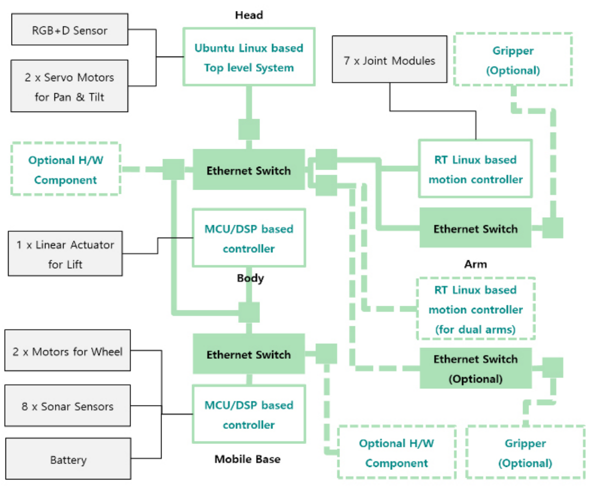
However, if different communication methods among modules are used, the advantage of the modularization will be reduced. Therefore, the modularization has to be implemented in terms of consistency of their communication method. [Fig. 14] shows a mobile manipulator that modularizes these different functions into their respective subsystems. The prototype robot consists of four subsystems. The mobile subsystem uses two wheels for differential driving locomotion and employs eight sonar sensors for detecting obstacles. The body has a 1-DOF lift to adjust the height of the head and arm modules. The arm is a 7-DOF manipulator, which is controlled by the embedded RT-Linux-based motion controller.

The head has an RGB+D camera for image processing and two servo motors for pan and tilt motion. It also contains an x86_64-based SBC that operates the ROS framework by arbitrating the required information for communication between the ROS framework and subsystems. Even though the subsystem in the robot operate in different platform, but they can communicate with each other by using only ROS protocol consistently. [Fig. 15] shows the controllers used for each subsystem of the constructed prototype robot and the method to implement ROS protocol in each controller. The head module used the x86_64-based SBC to operate Ubuntu Linux and the standard ROS framework. The arm module used Raspberry Pi 3 SBC that ported RT kernel. Also, it implemented ROS communication functionality by porting ROS_M library into its operating system. Lastly, the body and base subsystems used the constructed Cortex-M4 MCU/DSP control board that ROS_M library is also ported. Therefore, all subsystems can communicate with each other by using only ROS protocol consistently despite their platform difference.

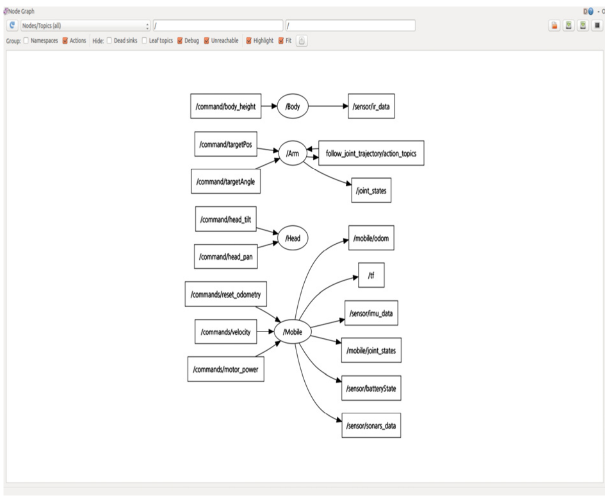
Meanwhile, the modularized subsystems need to provide convenient combine method. The most widely used combine method in reconfigurable modular robotic system is daisy chain that can reduce wires. The signal cables in a daisy chain structure system don't need to reach to its master controller or its master hub because each module in the system routes the packets to the module attached with it.

[Fig. 16] shows the system structure of a modularized mobile manipulator. Each subsystem is connected to each other through ethernet switches that are used to implement an ethernet-based daisy chain structure. Ethernet switches have extra ports for subsystem expansion, which enables the robot to be easily configured in terms of expansion and modification.

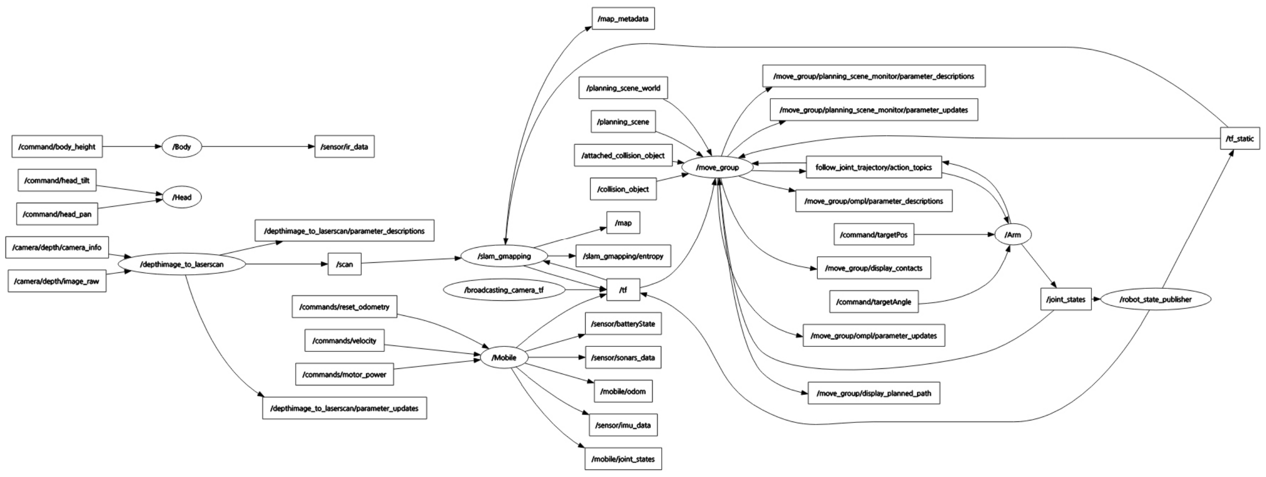
For example, if the mobile manipulator currently configured as a single arm needs to be expanded to dual arms, an additional arm can be attached using the ethernet switch of the body. In addition, if the current 1-DOF lift structure body need to be replaced with more flexible body, such as 2-DOF rotary type, the new 2-DOF body that fits the mechanical interface with the arm



[Fig. 16] Expandable system of the mobile manipulator



[Fig. 17] Recognized subsystems as a ROS_M node



[Fig. 18] Integrated subsystems with the ROS nodes of Gmapping and Moveit packages and base module can substitute for the existing body. Additional subsystems such as grippers and contents monitors can also be added to the switch when implementing each subsystem by the ROS communication protocol using the ROS_M library. All nodes in the system can make full closed control loop under 1-kHz control period based on the result showed in section 3.

Finally, [Fig. 17] shows that the subsystems recognized in the ROS framework. Each subsystem provided topics and services to the control interfaces for the attached actuators. In addition, for the actual operation of SLAM and visual servoing, the subsystems can communicate with the nodes of Gmapping and Moveit packages provided by the standard ROS framework. [Fig. 18] shows the subsystems associated with the nodes provided in the Gmapping and Moveit packages. Note that if a subsystem needs to be replaced because of changes in requirements, the user can exchange it with another subsystem that has the same topic and service interface. Once the replaced subsystem connects to the network, the system will operate without additional effort at system integration.

4.2 Limitation and Future Works

In the expandable system presented in this study, the ROS_M nodes represent the controllers of each functional subsystem that operate together in the ROS framework. However, no test has been conducted to verify that data interlocking among subsystems is stable under the actual operation of the loop control functions of each controller that controls the actuators and sensors.

After we implement these specific functions in a future study, we plan to examine whether the proposed system, including the implementation of high-level services, works organically in the expandable structure.

5. Conclusion

In this paper, we proposed an expandable robotic system that can have subsystems removed or added efficiently to respond to changes in a robot's environment and tasks. The proposed system is a structure in which each subsystem is integrated as an ROS node in an ROS framework by directly supporting an ROS protocol in the controller of each subsystem. We presented a developed ROS_M library to implement the ROS protocol in different subsystem control environments. The results of a latency measurement test showed the constructed system when using our ROS_M library can operate a soft real-time-based loop control function below a 1-kHz control period, as the worst-case round-trip latency between nodes when ethernet switches were considered was approximately 587.2 us. Finally, we introduced an expandable robotic system for implementing a mobile manipulator modularized into four subsystems. The controllers of each subsystem were implemented using the ROS_M library on different control platforms, and they interlocked with other ROS nodes of Gmapping and Moveit packages under the ROS framework in terms of software communication.

References

[1] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS: An Open-Source Robot Operating System," *ICRA Workshop Open Source Software*, 2009, [Online], http://www.willowgarage.com/sites/default/files/icraoss09-ROS.pdf.

[2] H. Bruyninckx, "Open Robot Control Software: The OROCOS Project," *2001 ICRA. IEEE International Conference on Robotics and Automation*, Seoul, South Korea, 2001, DOI: 10.1109/ROBOT.2001.933002.

[3] B. P. Gerkey, R. T. Vaughan, and A. Howard, "The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems," *International Conference on Advanced Robotics (ICAR 2003)*, 2003, Coimbra, Portugal, pp. 317-323, [Online], https://faculty.ontariotechu.ca/shi/DMCG/seminar/Robert%20-%2025.09.2013%20-%20%20The%20PlayerStage%20Project-%20Tools%20for%20Multi-Robot%20and%20Distributed%20Sensor%20Systems.pdf.

[4] M. E. Munich, J. Ostrowski, and P. Pirjanian, "ERSP: A Software Platform and Architecture for the Service Robotics Industry," *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Edmonton, Alta., Canada, 2005, DOI: 10.1109/IROS.2005.1545468.

[5] J. Jackson, "Microsoft Robotics Studio: A Technical Introduction," *IEEE Robotics & Automation Magazine*, vol. 14, no. 4, 2007, DOI: 10.1109/M-RA.2007.905745.

[6] C. Jang, S.-I. Lee, S.-W. Jung, B. Song, R. Kim, S. Kim, and C.-H. Lee, "OPRoS: A New Component-Based Robot Software Platform," *ETRI Journal*, vol. 32, no. 5, 2010, DOI: 10.4218/etrij.10.1510.0138.

[7] C. Crick, G. Jay, S. Osentoski, B. Pitzer, and O. C. Jenkins, "Ros-bridge: Ros for Non-Ros Users," *15th International Symposium on Robotics Research*, 2011, [Online], http://www.isrr-2011.org/ISRR-2011/Program_files/Papers/Jenkins-ISRR-2011.pdf.

[8] J. Lee, "Web Applications for Robots Using Rosbridge," 2012, [Online], http://cs.brown.edu/research/pubs/theses/masters/2012/lee.pdf.

[9] B. Alexander, K. Hsiao, C. Jenkins, B. Suay, and R. Toris, “Robot Web Tools,” *Robotics Automation Magazine*, vol. 19, no. 4, 2012, DOI: 10.1109/MRA.2012.2221235.

[10] M. Blaha, M. Krec, P. Marek, T. Nouza, and T. Lejsek, “Rosbridge Web Interface,” *Czech Technical University*, Czech Republic, May, 2013, [Online], https://klein.felk.cvut.cz/w/_media/misc/projects/nifti/sw/web_interface-report.pdf.

[11] L. Jurišica and R. Murár, “Mobile Robots and Their Subsystems,” *AT&P Journal Plus 1*, pp.14-17, 2008, [Online], https://www.atpjournal.sk/buxus/docs/casopisy/atp_plus/plus_2008_1/plus14_17.pdf.

[12] S. Pedre, M. Nitsche, F. Pessacg, J. Caccavelli, and P. De Cristóforis, “Design of a Multi-Purpose Low-Cost Mobile Robot for Research and Education,” 2014, DOI: 10.1007/978-3-319-10401-0_17.

[13] H. Gao, S. Lu, G. Tian, and J. Tan, “Vision-integrated Physiotherapy Service Robot using Cooperating two Arms,” *International Journal on Smart Sensing and Intelligent Systems*, vol.7, no. 3, pp.1024-1043, 2014, [Online], https://exeley.mpstechnologies.com/exeley/journals/in_jour_smart_sensing_and_intelligent_systems/7/3/pdf/10.21307_ijssis-2017-692.pdf.

[14] M. Migliavacca and A. Zoppi, µROSnode: running ROS on microcontrollers, *ROS Developers Conference*, 2013.

[15] M. Migliavacca, A. Bonarini, and M. Matteucci, “Modular Development of Mobile Robots with Open Source Hardware and Software Components,” *Robot Soccer World Cup*, 2013, DOI: 10.1007/978-3-662-44468-9_52.

[16] A. Bonarini, M. Matteucci, M. Migliavacca, and D. Rizzi, “R2P: An open source hardware and software modular approach to robot prototyping,” *Robotics and Autonomous Systems*, 2014, DOI: 10.1016/j.robot.2013.08.009.

[17] D. A. Cucci, M. Migliavacca, A. Bonarini, and M. Matteucci, “Development of Mobile Robots using Off-The-Shelf Open-Source Hardware and Software Components for Motion and Pose Tracking,” *Intelligent Autonomous Systems 13*, 2016, DOI: 10.1007/978-3-319-08338-4_104.

[18] Steffi Paepcke, *Morgan Quigley (OSRF): ROS 2 on “Small” Embedded Systems*, [Online], https://www.osrfoundation.org/morgan-quigley-osrf-ros-2-on-small-embedded-systems, Accessed: Sept. 28, 2019.

[19] H. S. Kang and D. H. Shin, “OPRoS_M: a Library to Develop a H/W Device Component of OPRoS Platform,” *Intelligent Service Robotics*, 2015, DOI: 10.1007/s11370-015-0168-z.

Hyeong Seok Kang

2013~2016 Mechanical and Information Engineering, University of Seoul, Korea (Ph.D.)

2016~ MINTROBOT Co., Ltd., Korea (CEO/Founder)

Interests: Low-cost robotics, Modular robot system, Gearbox

Dong Won Lee

2010~2016 Mechanical and Information Engineering, University of Seoul, Korea (B.S.)

2016~2018 Mechanical and Information Engineering, University of Seoul, Korea (M.S.)

Interests: Robotics, Computer Engineering

Dong Hun Shin

1987~1990 Civil Engineering, Carnegie Mellon University (Ph.D.)

1990~1992 Robotics Institute, Carnegie Mellon University (Researcher)

1992~1994 Korea Institute of Industrial Technology (Principal Researcher)

1994~ Mechanical and Information Engineering, University of Seoul, Korea (Professor)

Interests: Robotics