ROBOMECH Journal

RESEARCH ARTICLE

Open Access

Development of ROS2-TMS: new software platform for informationally structured environment

Tomoya Itsuka*, Minsoo Song, Akihiro Kawamura and Ryo Kurazume

Abstract

This study proposes a new software platform, called ROS2-TMS, for an informationally structured environment. An informationally structured environment is vital for developing intelligent service robots by embedding various sensors in the environment to enhance the sensing capability and intelligence of robots. Thus far, we have been developing a software platform, named ROS-TMS, for an informationally structured environment, which connects various sensors and robots using ROS architecture. In recent years, ROS2, a next-generation version of ROS, has been released. ROS2 has many advantages, such as enhanced security, QoS control, and support for various platforms. ROS2-TMS, a new version of ROS-TMS, is developed not only by porting existing modules in ROS-TMS, such as the control system for a communication robot, but also by adding useful functions utilizing new features in ROS2. For instance, we added a voice user interface to control robots and various devices in the environment, such as lights or a bed. In addition, we implemented a new task scheduler that provides a cancelation function to stop running tasks and improve the security of the platform.

Keywords: Service robot, Informationally structured environment, Internet of things, Cyber physical system, Ambient sensing

Introduction

In recent decades, with the decline in birthrate and aging of population, labor shortage has become a crucial issue in various fields such as medical and nursing care. As a solution to this problem, the realization of life support services using artificial intelligence and service robots has attracted considerable attention. However, unlike robots operated in factories, the environment in which service robots work for daily life support is diverse and dynamically changing. Therefore, it is difficult to fully understand the surrounding situation based on the embedded sensors of the robot alone. One of the key solutions to this problem is an informationally structured environment (ISE), in which sensors are distributed throughout the surrounding environment to collect, analyze, and retain environmental information. As a software platform for an ISE, we have been developing ROS-TMS [1], which realizes service robots that coexist with humans. ROS-TMS connects various sensors embedded in the environment and service robots using a robot operating system (ROS) [2], which is a general-purpose robot middleware. In recent years, ROS2 [3], a next-generation version of ROS, has been released. ROS2 has many advantages, such as enhanced security, QoS control, support for various platforms, and advanced navigation algorithms [4].

In this study, we propose a novel software platform for an ISE, named ROS2-TMS. The characteristics of ROS2-TMS are as follows. (1) Upgradation of the middleware from ROS to ROS2: Enhanced security, QoS control, and latest navigation algorithms are available. (2) Unified task management mechanism: Regardless of the type of device (for example, robots, room lighting, and

*Correspondence: itsuka@irvs.ait.kyushu-u.ac.jp
Kyushu University, 744 Motooka, Nishi-ku, Fukuoka-shi, Fukuoka 819-0395, Japan

intelligent home appliances), service tasks can be managed uniformly as subtasks in the task manager. This makes it easier to perform advanced coordination of various devices. (3) Cancelation function: By managing subtasks in all devices using the ROS2 Action protocol, all service tasks can be terminated during execution regardless of the complexity of the task, thus, increasing safety.

Related works

ISE enables the flexible operation of service robots by collecting, analyzing, and managing data for a complex environment by deploying distributed sensors not only on the robot but also in the environment. Several systems have been developed on ISE thus far. In the "Robotic Room" [5] at The University of Tokyo, an approach to deploy various sensors in the environment was proposed to observe the status of patients. In the "Intelligent Room" [6] at MIT, AILab deployed multiple cameras in a room to track the location of humans and determine their direction to provide services. "Intelligent Sweet Home" [7] proposed a robotic platform that includes an intelligent bed with pressure sensors and an autonomous mobile wheelchair in the context of ambient assisted living (AAL), which aims to promote independent living for the elderly and disabled. These studies on ISEs have focused on two aspects: (1) deploying sensors in the environment to collect advanced information regarding humans, robots, and objects, and (2) providing services to assist humans through robots and intelligent home appliances.

In recent years, various systems related to ISE have been proposed and commercialized, such as AAL, smart houses, and the Internet of Robotic Things (IoRT). In the field of smart homes, smart speakers, such as Amazon's "Alexa" [8] and Google's "Google Assistant" [9], are widely used in households, and the technology to control home appliances, lighting, and other Internet of Things (IoT) devices through voice interfaces is widely adopted in households. In [10], it was stated that the presence of a central interface, such as a smart speaker, increases the quality of the robot service. IoRT is a field that incorporates IoT technology, in which various devices in the home environment are connected to the Internet and robotics technology. As a method of component coordination using IoT technology, there is coordination between sensors to collect information using multiple sensors, as well as coordination on actuators to perform various services; for example, [11] identifies user activities and habits by collecting information from a group of sensors installed in a smart house, such as a pressure sensor installed in a bed. In [12], a service robot was connected to an elevator control unit to move the service robot across multiple floors of an apartment building. Various research topics and references on IoRT are introduced in [13].

The robot operating system (ROS) [2], an open source platform, has contributed significantly to the development and research of mobile and service robots. The new generation of ROS, ROS2 [3], has enhanced security and QoS control and supports the latest development environments, such as C++14 and Python3. The latest robot autonomous mobility package Navigation2 is also available in ROS2. Navigation2 [4] consists of a behavior tree, which represents a management mechanism that is higher than the autonomous movement of the robot, and it is easy to describe complex control such as temporary avoidance behavior for environments wherein autonomous movement is difficult. Consequently, it has the potential to cope with complex everyday environments. One of the important updates in the internal design is the action protocol [14], which is useful for time-consuming tasks, such as the moving task of a robot or a picking up task of a manipulator. With the evolution of the design, an identifier is issued for each action protocol to distinguish each action. This action protocol is incorporated into the task scheduler proposed in this research and plays a major role in connecting individual tasks and subtasks.

From ROS-TMS to ROS2-TMS

The authors began to develop an ISE in the "Robot Town Project" in 2005 and have been developing a software platform named Town Management System (TMS). In [1], we proposed a software platform, ROS-TMS 5.0 (Fig. 1), and a hardware platform, Big Sensor Box (Fig. 2), for an ISE. ROS-TMS 5.0 is developed as a core software platform for IoRT and has some novel functions, such as a care receiver-watching service and a voice control for service robots.

However, there were four concerning points about ROS-TMS:

1. Using ROS as middleware: ROS initially supported Python2; thus, the ROS-TMS executable used Python2. However, support for Python2 was dis-

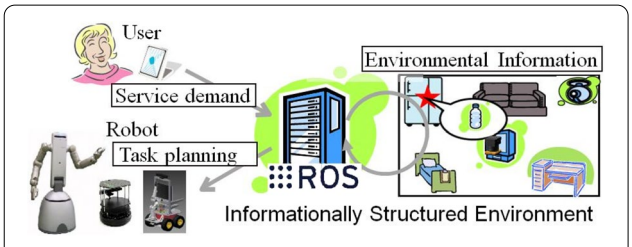


Fig. 1 ROS-TMS

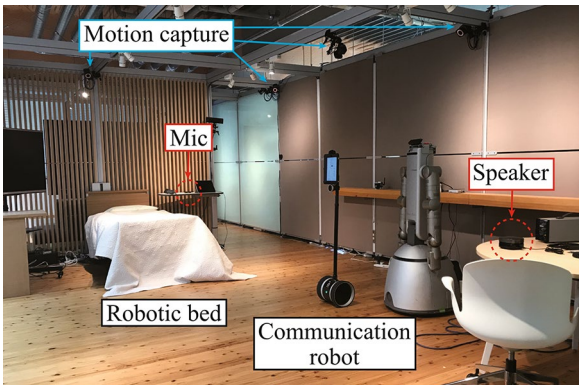


Fig. 2 Big sensor box

continued in 2020. We believe that the platform should support Python3 and the latest development resources.

2. Difficulty in task implementation: The ROS-TMS microphone module, which is responsible for voice input, transcribes the user's speech, searches for service tasks, and executes tasks specific to those services (sending commands to devices such as robots, lightings, and robotic beds using ROS topics, services, socket communication, etc.). Therefore, the dependency between task implementation and the microphone module is high, and it is necessary to develop the microphone control node again when tasks are added.
3. Limited subtasks: In ROS-TMS, the objects to be managed by the task scheduler were limited to robots only, and there were only three types of subtasks: robot movement, robot arm grasp, and release. Together with the problems discussed in 2) above, it was difficult to implement services that scheduled robot movements and other devices.
4. Cannot stop a task while it is running: For example, when a robot is moving and bumps into something inadvertently, it is necessary to perform an emergency stop of the service task. However, ROS-TMS does not allow the user to perform an emergency stop while the task is in progress.

Therefore, we developed a new software platform for an ISE, named ROS2-TMS. This platform has the following four features to solve these four problems:

1. Using ROS2 as middleware: While ROS uses C++03 / Python2 as its development environment, ROS2 can now use C++14 / Python3. Therefore, ROS2-TMS ports the core modules of ROS-TMS to ROS2, which can be used in the C++14 / Python3 environment. Also, advanced ROS2 technologies such as QoS control and Navigation2 are applicable in ROS2-TMS.
2. Redesign of task execution: In ROS-TMS, the microphone module was responsible not only for understanding the user's speech but also for task search and task execution. In addition, ROS2 does not have SMACH, the task execution machine used in ROS-TMS; therefore, the task execution had to be rebuilt. Therefore, we separated task search and task execution from the roles of the microphone module and implemented task search in the task search node and task execution and management in the task scheduler module. Furthermore, task execution was performed using ROS2 actions, and we were able to add functions for failure behavior and stopping during execution.
3. Redesign of tasks and subtasks: In ROS-TMS, there were only three types of subtasks: moving the robot and grasping and releasing by the robot hand. In ROS2-TMS, the scope of subtasks was expanded to include robots as well as robotic beds, room lighting, and speakers. As a result, all service tasks can be managed using a task scheduler. In addition, tasks linked to multiple devices can be added simply by entering the configuration information of these subtasks in the database.
4. Addition of cancel function: In ROS-TMS, a user cannot stop a task in the middle of the execution. ROS2-TMS has a newly developed task scheduler with ROS2 action, which can stop a task in the middle of an execution according to the user's request. Each subtask defines its behavior when canceled such that adding a task does not need to define a new behavior.

The rest of the paper is organized as follows: The ROS2-TMS modules section describes the structure of the modules in ROS2-TMS. The Robots, Sensors, and User Request Devices section describes the modules related to the devices in ROS2-TMS, such as robots and robotic beds. The Task execution flow section describes an overview of task execution. The Voice interfaces (TMS_UR) section describes the design of the voice user interface. The Database (TMS_DB) section describes the design of the database. The Task scheduler (TMS_TS) section describes the design of the task scheduler. The Robot service experiment section describes the service tasks that can be provided by ROS2-TMS and provides examples of their execution. Finally, the section Conclusions provides the conclusion.

ROS2-TMS modules

ROS2-TMS has a wide variety of functions as an ROS2 node. ROS2-TMS is composed of modules connected in a hierarchical manner as shown in Fig. 3. In a ROS2-TMS service, various modules play roles by interconnecting with each other, such as interpreting the user's speech, selecting a task to be executed, planning, and even the robot's behavior. The contents of each module are described below.

Architecture

Database module (*TMS_DB*) Stores environmental information managed by ROS2-TMS in a database. The database is developed with mongoDB.
User request module (*TMS_UR*) Receives task requests from users and sends task execution requests to TMS_TS.
Task scheduler module (*TMS_TS*) The requested service is executed by combining subtasks. In the previous study (ROS_TMS), only robots were subject to subtask management, whereas in this study, we expanded the scope of task management to intelligent home appliances such as beds and speakers.
Robot planning module (*TMS_RP*) From some subtasks of the robot commanded by TMS_TS, motion planning was performed to correctly execute the subtasks.
Robot controller module (*TMS_RC*) This is a module that executes the planned subtasks using the robot. A dedicated module is implemented for each robot.
Sensor driver module (*TMS_SD*) The system activates various sensors embedded in the environment and publishes the acquired sensor data.

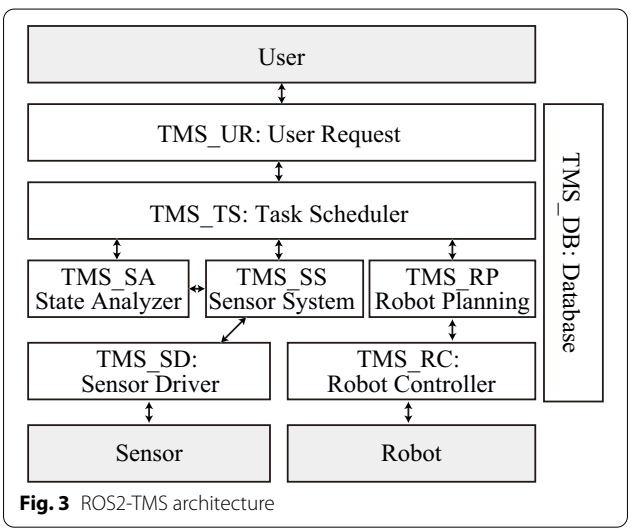


Fig. 3 ROS2-TMS architecture

Sensor system module (*TMS_SS*) Sensor data are interpreted and converted into higher-level environmental information and stored in TMS_DB.
State analyzer module (*TMS_SA*) This module receives information from TMS_SS and estimates the state of the environment. For example, we plan to estimate the user's health status, but we are still in the concept stage. Details are given in the "Conclusions" section.

Robots, sensors, and user request devices

In this study, we upgraded the middleware for each device from ROS to ROS2, improved the communication quality of the Data Distributed Service (DDS), and expanded the functions of ROS2, such as the cancelation function. Some sensors and robots used in Big Sensor Box and ROS2-TMS are shown in Fig. 4.

Robot controller module (TMS_RC)

Robotic bed An electric bed (Rakusho Z KQ-7302, Paramount Bed) is controlled by RaspberryPi Zero, which can raise the upper body and the height of the bed.
Communication robot (Double 2, Double 3) Double 2 [15] and Double 3 [16] are communication robots manufactured by Double Robotics. Double 2 and Double 3 are controlled by attached iPad and Linux PC, respectively. We implemented a moving task for the robots to a specified position and angle using a motion capture system and the ROS2 Navigation2 package.

Sensor system module (TMS_SS)

Wearable Heart rate sensor (WHS-1) Whs-1 is a wearable heart rate sensor developed by Union Tool Corporation that updates the heart rate information in the database at each timestep at which the user's heart rate is measured.
Motion capture system (VICON) VICON is a motion capture system that uses multiple motion-tracking cameras to recognize the position of an object. We used it to estimate the position of the robot.

User request module (TMS_UR)

Microphone device A USB microphone is attached to Intel's NUC to receive voice requests from users.

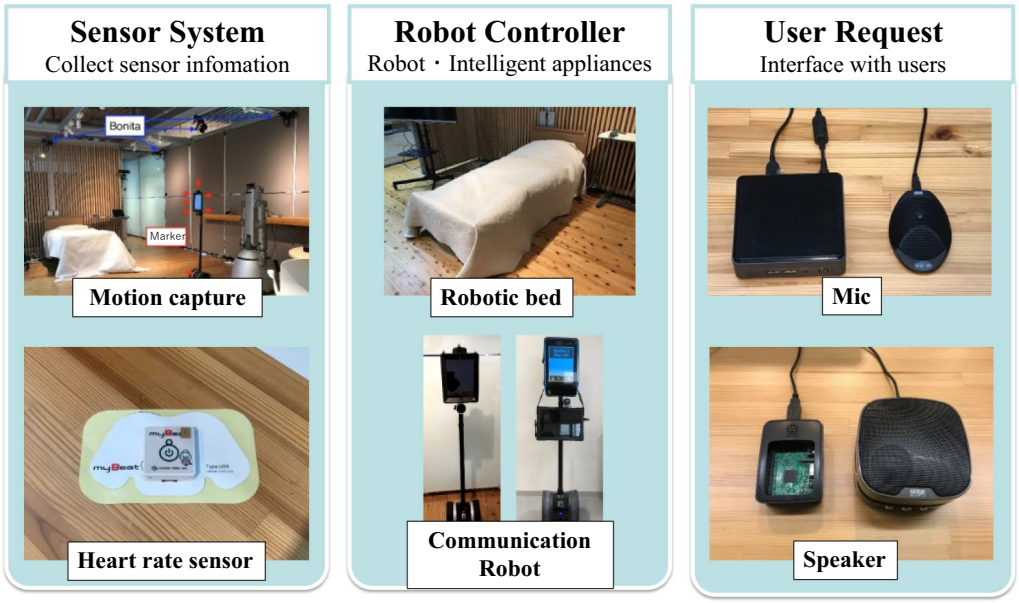


Fig. 4 Sensors and robots in TMS_SS, TMS_RC, TMS_UR modules

Speaker device A USB speaker is attached to the Raspberry Pi 3 to play a conversation with the user using the Google Assistant API, as well as announcements and sound effects when the task starts using TMS_TS.

Task execution flow

To run the ROS2-TMS service, the modules shown in the previous sections need to control corresponding devices and tasks coordinately. In TMS_RC, the robots and the robotic bed are controlled. In TMS_RP, the robots are connected to TMS_TS through Navigation2, and the robotic bed is connected directly to TMS_TS. The microphones and the speakers are managed in TMS_UR. TMS_TS, TMS_DB, and the task search node in TMS_UR were executed on a dedicated ROS2-TMS server in our experiments.

An overview of task execution from a user's voice command to the robot and lighting control is presented in Fig. 5. The following bullet points correspond to the following figure:

1. The user requests a command from the microphone.
2. The microphone control node transcribes the voice and sends a string to the task search node.
3. The task search node searches for related task IDs and objects in the database.
4. The task search node passes the task IDs and objects to TMS_TS.

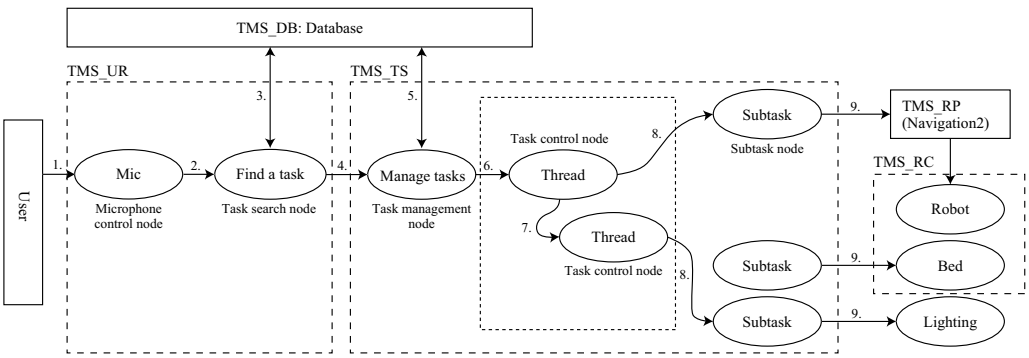


Fig. 5 Flowchart of processes from user request to task execution in ROS2-TMS

5. The task management node integrates task and object information.
6. The task management node creates a task control node to execute the task.
7. If necessary, more task control nodes are generated and coordinated by an additional number of tokens for parallel execution.
8. Each task control node eventually requests one subtask node.
9. Each subtask node executes its process.

The subsequent sections on "Voice interfaces (TMS_UR)," "Task scheduler (TMS_TS)," and "Database (TMS_DB)" describe the function of the individual modules in detail.

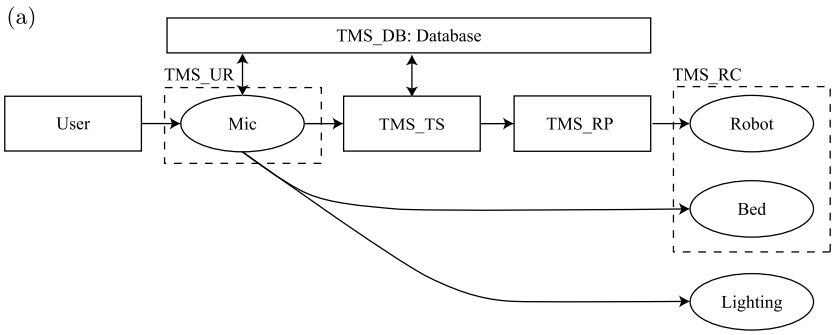
Voice interfaces (TMS_UR)

Currently, smart speakers are being released by various companies, and home automation, in which home appliances are controlled by smart speakers, is also gaining popularity. In ROS2-TMS, we implemented a voice interface using the Google Assistant API [17], a voice assistant service that is also installed in Google's smart speakers, to transcribe text and provide simple responses.

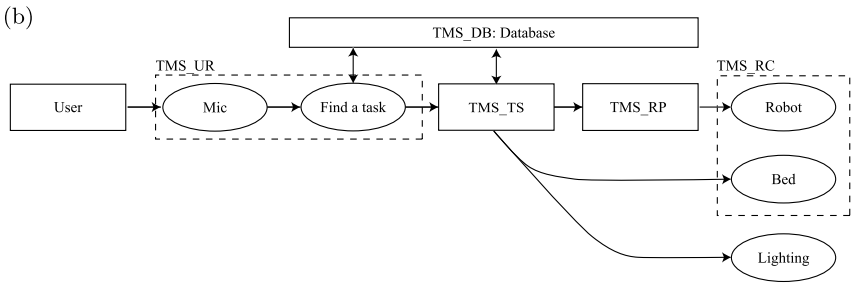
Note that the Google Assistant API is only used for speech recognition and transcription, weather forecasting, and other general-purpose responses, and not for operating devices such as robotic beds. All device operations on ROS2-TMS were performed via ROS2.

The ROS-TMS also had a voice interface. However, a microphone control node implemented the task search and execution functions, which had to be developed when a task was added (Fig. 6a). In ROS2-TMS, the task search and execution functions are outsourced to the other nodes. In particular, the task scheduler can manage the execution of all tasks. As a result, the development of a microphone module is no longer necessary when tasks are added or changed. In addition, tasks that combine multiple devices can be realized using only the task information in the database (Fig. 6b).

First, the wake word "ROS-TMS" is detected by Julius [18]. Once the wake word is detected, Google Assistant transcribes the request. The transcribed text is separated into words by Janome [19] and then matched to tags in the database for task execution by ROS2-TMS.



ROS-TMS task execution flow of User Interface (TMS_UR) and Task Scheduler (TMS_TS). In this implementation, the microphone control node is responsible for speech recognition, task search, and task execution; therefore, it is necessary to develop the microphone control node again when tasks are added.



ROS2-TMS task execution flow of User Interface (TMS_UR) and Task Scheduler (TMS_TS). The microphone control node only transcribes the speech and sends the text to the task search node; therefore, it is no longer necessary to develop the microphone control node when tasks are added or changed.

Fig. 6 Task execution flow of User Interface (TMS_UR) and Task Scheduler (TMS_TS)

If an appropriate task is found, TMS_TS schedules the execution of the task. If a proper task cannot be found, the speaker outputs the response using Google Assistant.

"Cancel" is also a wake word recognized by Julius. When Julius recognizes "cancel," TMS_TS is requested to terminate the current task immediately.

Database (TMS_DB)

The database stores tasks and environmental information, such as a map of the environment, task information, and robot, human, and object positions.

The task information included an ID of the task, a sequence of subtasks that complied with the task, tags used for voice search, and the text that was announced when the task was started.

In addition to these tasks, other objects can be added to the database. The following four pieces of information are required to work with the voice interface:

ID: A unique ID in the database.
Name: A name that represents this object.
Type: The name of the type that the object represents, such as "room_place." The same type must have the same properties.
Tags for searching: Store several related words for searching the object. In addition to this, there shold be properties for each type.

The information in the database can be referenced from other information. For example, the robot movement task (ID: 9001) refers to the patrol points specified by the user and stored in the database; thus, the robot moves autonomously along these points (Fig. 7). It is described in detail in the "Integrate tasks and databases" subsection under the "Task Scheduler (TMS_TS)" section.

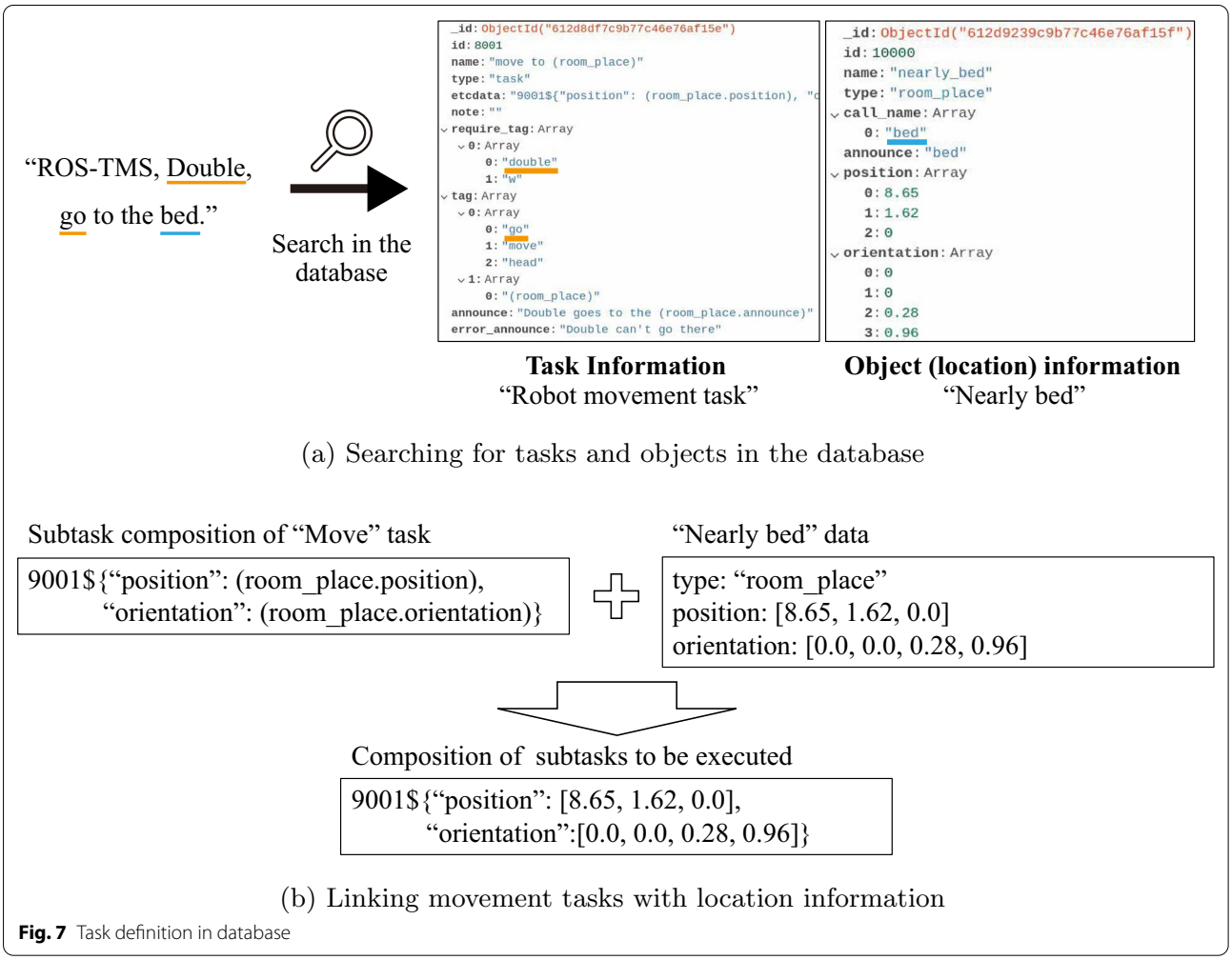


Fig. 7 Task definition in database

Task scheduler (TMS_TS)

The task scheduler module (TMS_TS) receives the task requested from the user request (TMS_UR), analyzes the request, and executes the task while interpolating the necessary information in cooperation with the database.

In our previous study [1], we adopted the approach named "task information structuring." In this approach, a task is defined as a combination of subtasks, which are basic actions shared among different tasks. We implemented moving, grasping, and handling subtasks, all of which are fundamental functions for service robots. In addition to these subtasks, we implemented new subtasks in ROS2-TMS, which control not only robots but also devices such as speakers, beds, and room lighting. In addition, all tasks can be canceled using the ROS2 Action protocol.

Consequently, ROS2-TMS has the following advantages compared to ROS-TMS.

- Controllable devices are increased. Various devices, such as speakers, room lighting, and beds, can be controlled simultaneously to provide service tasks with robots (Fig. 8).
- Since all operations are centrally controlled by the task manager using the ROS2 Action protocol, any task can be terminated or canceled immediately.

Two-layer service structure: tasks and subtasks

We defined and provided services using a two-layer structure in ROS2-TMS, that is, the task and subtask layers.

A task is defined by a one-to-one correspondence with a service that can be requested in ROS2-TMS. The task consists of the following contents. All tasks are stored as data structures in the database and thus have no substance as programs.

- Tags for searching by a requested text
- Set of subtasks
- Text announced when starting a task

If a task needs to refer to environmental information in the database, such as the destination of a robot in a moving task, the type of necessary information is also specified in the data structure above.

A subtask is implemented as an ROS2 node that can be executed by robots or devices. A subtask is composed of the following contents.

- Unique ID for each subtask
- Processing implementation
- Implementation of the cancelation process

Because a task is defined as a set of subtasks, a new task can be represented as a set of existing subtasks; thus, it is not necessary to implement the execution code of robots or devices for each task individually.

Method of adding a subtask

It is implemented by inheriting the SubtaskNodeBase class, which inherits from the ROS2 Node class and then overrides the following items:

Name: Name of the ROS2 node.
The ID of the subtask: Assign a number between 9000 and 9999.

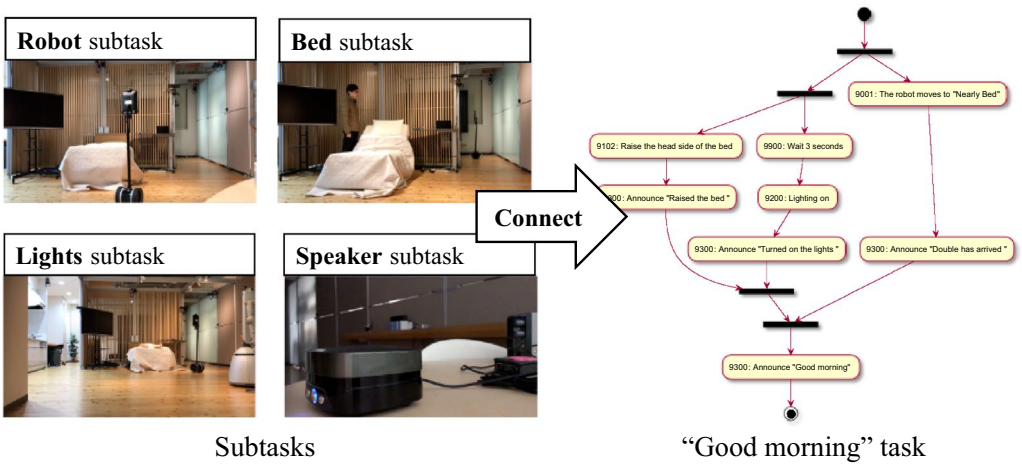


Fig. 8 "Good morning" task consisting of subtasks for various devices

Function at runtime: Define the process at runtime. Because arguments assigned by the task scheduler are allocated, it is possible to change the behavior according to the arguments.
Function when canceled: The action to be taken can be defined when cancelation is requested.

Implementation of subtasks

Subtasks are implemented as the minimum processes executed by the robots and various devices. All subtasks were implemented as nodes in the ROS2. IDs from 9000 to 9999 were assigned to the subtasks. Currently, 13 subtasks are defined in the database. The detailed behaviors when executing the subtasks are shown in Table 1.

Subtask for communication robot (tms_rc_double)

ID 9001 is a subtask for a robot, such as the communication robot Double 2, to move to a destination. The destination is defined with a position and an orientation and specified when the subtask is called. This movement subtask uses the Navigation2 package and a behavior tree to move.

- Double 2 moves to the destination (9001).

Subtasks for robotic bed (tms_rc_bed)

IDs in the 9100s are subtasks for a robotic bed. For each subtask, the execution time in seconds is defined as an argument.

- Raise (9100) or lower (9101) the head and leg sides of the bed.
- Raise (9102) or lower (9103) the head side of the bed.
- Raise (9104) or lower (9105) the leg side of the bed.
- Raise (9106) or lower (9107) the height of the bed.

Other subtasks

- Turn on (9200) or off (9201) the lights in the room.
- Play a text from the speaker (9300).
- Wait for specified seconds (9900).

Connecting subtasks within a task

A task is defined as a set of subtasks and stored with string information in the database. The subtask execution token is represented by "subtask_id" or "subtask_id $ json_format_argument," and the task is specified by connecting them with the sequential execution token "+" or the parallel execution token "|." Examples of connecting multiple subtasks using this notation are shown in Fig. 9.

More complex tasks can be represented with the Backus-Naur Form (BNF) notation as follows:

$$
\begin{aligned}
\langle task-structure\rangle &::= \langle task\rangle \\
\langle task\rangle &::= \langle task\rangle\ \langle task\rangle\ \langle operator\rangle \\
&\quad\ |\ \langle subtask\rangle \\
\langle operator\rangle &::= \text{``}+\text{''}\ \ |\ \text{``}|\text{''} \\
\langle subtask\rangle &::= \text{``}subtask-id\text{''} \\
&\quad\ |\ \text{``}subtask-id\$json-arguments\text{''}
\end{aligned}
$$

Table 1 Definition of the implemented subtasks

Name	IDs	Arguments	Behavior	Behavior on cancelation
Robot movement subtask	9001	Goal position (x, y, z) Goal angle (w, x, y, z)	Requests the Navigation2 package to autonomously move the robot to the target location using ROS2 action communication. If the request fails, it informs the upper-level task control node that it has failed.	Cancels the currently executed request to Navigation2 by ROS2 action.
Subtasks for robotic bed	9100 9101 9102 9103 9104 9105 9106 9107	Execution seconds	Requests the bed control node (websocket operation) in TMS_RC to perform the respective operation via a websocket, such as raising the bed. If another task has already made the bed request, this subtask informs the upper task control node of the failure.	Sends a stop command to the bed control node via a websocket.
Subtasks for the lights	9200 9201	None	An HTTP request is sent to the lighting management server located in the big sensor box to turn the lights on and off. Failure is reported to the upper task control node if no response is received from the server within the 5-second timeout period.	None
A subtask for the speaker	9300	Text to be spoken	Request the speaker control node in TMS_UR to speak using the ROS2 service.	None
A subtask for the speaker	9900	Execution seconds	Wait for the specified time.	None

ex.1	ex.2	ex.3	ex.4	ex.5
9001	9002 9003 +	9004 9005 |	9006 9007 + 9008 |	9009 9010 9011 | + 9012 + 9013 |

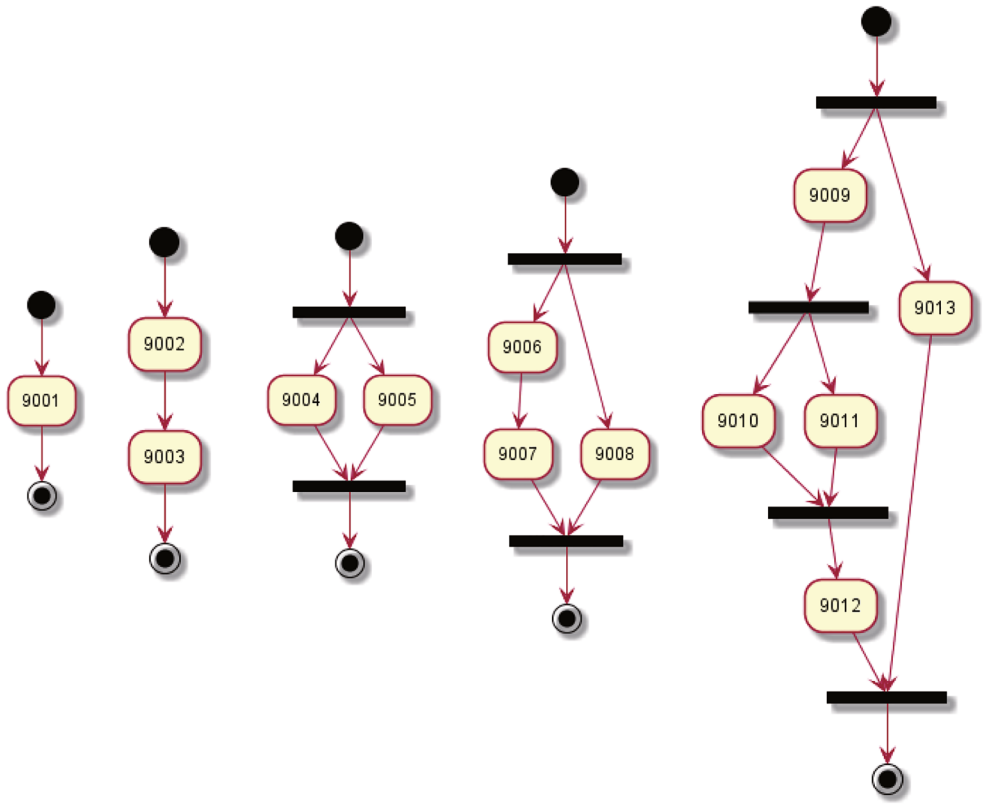


Fig. 9 Task notations

⟨*task* − *structure*⟩ represents the connection between the subtasks. "*subtask* − *id*" is the ID of the subtask to be executed, and "*subtask* − *id*$*json* − *arguments*" is the ID and some arguments. For example, the string command for raising the height of the bed for 17 s is represented as

```
9106${"sec":17.0}
```

This command implies that the subtask to raise the height of the bed (ID: 9106) is executed for 17 s.

Integrate tasks and databases

The task description in the database can be linked with other objects.

(⟨*object* − *type*⟩.⟨*property* − *name*⟩)

The syntax above can be replaced by writing it in the subtask structure or the startup readout property. ⟨*object* − *type*⟩ is the type of information in the database, and ⟨*property* − *name*⟩is the property's name of the database object.

The integration of object information and the task is performed by the task search node in TMS_UR and the task management node in TMS_TS. For example, details of the operation from 3 to 5 shown in the "Task execution flow" section are as follows:

1. The task search node in TMS_UR checks the string against the task tag in the database and obtains the task ID and object.
2. The task search node sends the task ID and objects to the task management node in TMS_TS.
3. The task management node retrieves the task from the task ID. After that, it replaces the above syntaxes of the startup readout text and subtask structure with the information of the object.
4 A task control node is generated, and the task is executed.

Here is an example of a robot moving task. When a user says "ROS-TMS, Double, go to the bed," a task search node in TMS_UR retrieves the "Robot movement task" and the "Nearly bed" location information from the tags in the database (Fig. 7a). Subsequentry, the task management node in TMS_TS replaces the information in the "Robot movement task" with the contents of the "Nearly bed" (Fig. 7b). It then executes the task.

Method of adding a task

Adding a task is performed by storing the following information in the database.

ID: Set a unique number that does not overlap with any other object in the database.
Name: Give a name to the task content.
Type: Set the string "task" to distinguish it from other objects in the database.
Subtask composition: Set the composition of the subtasks using the syntax described in the "Connecting subtasks within a task" subsection.
Required tags and tags: If a word in a required tag is included in a sentence, this task is searched. If a word in the tag is included in a sentence, the priority of this task is increased in the searched tasks.
Announcement text: An announcement at startup and when object integration fails (optional). Set the text to be spoken by the speaker at startup. The text can also be set when the integration with the database fails; for example, when the target point of the robot's movement task does not exist in the database.

Implementation of tasks

Currently, there are 13 types of tasks. Tasks exist in a database, and their behavior can be based on a single subtask or a combination of subtasks.

A task related to the communication robot (1 type)

- Double 2 moves to the destination.

  Currently, there are 3 target locations in the database: near the entrance, near the kitchen, and the bed, and Double2 can move to them.

Tasks related to the robotic bed (8 types)

- Raise or lower the head and leg sides of the bed for a certain period.
- Raise or lower the head side of the bed for a certain period.
- Raise or lower the leg side of the bed for a certain period.
- Raise or lower the height of the bed for a certain period.

Tasks related to the lights (2 types)

- Turn on or off the lights in the room.

Tasks that combine multiple devices (2 types)

- "Good morning" task
- "Good night" task

Task execution

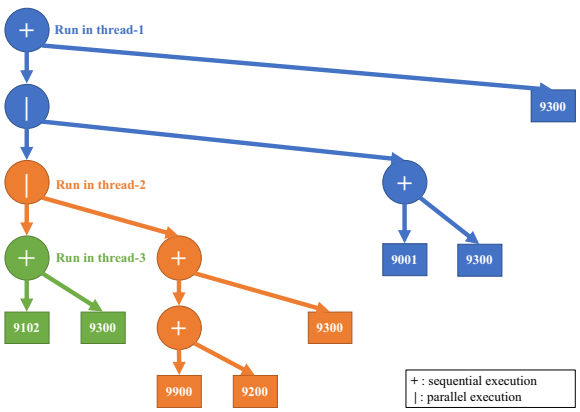
TMS_TS dynamically allocates a thread to execute the requested task when the execution of a task is requested by TMS_UR (Fig. 10a). The number of threads is set to $n + 1$, where n is the number of tokens "|" that represent the parallel execution.

A control node is an ROS2 node that corresponds to each thread. Several control nodes are created when tasks start, and these nodes are connected to each other by the ROS2 action protocol in the tree structure. The task scheduler represents the root, the control nodes represent the branches, and the subtask execution nodes represent the leaves. Action communication in ROS2 [14] handles accept or reject requests, and if accepted, the status of the action is returned when the action is concluded, such as succeeded, failed, or canceled. In addition, a cancelation request can be accepted while the action is being executed.

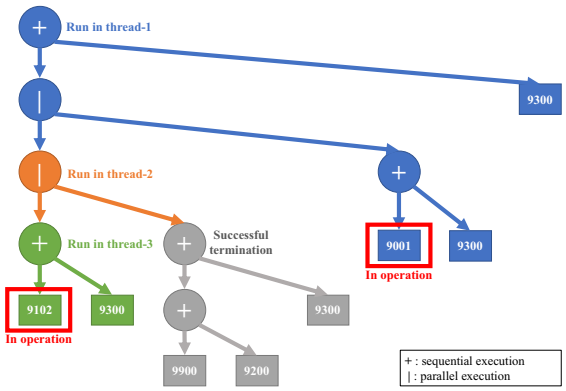
Cancel function

In ROS-TMS, a user cannot stop a task in progress. To implement this cancelation function, an update to the action function in ROS2 is required. ROS action communication consists of two types of entities: clients and servers. However, the server could not recognize which client requested it. In ROS2, when a client requests the server, the client generates a UUID [14]. This UUID is used as an identifier of the request between the server and the client to manage each request individually. This new feature was used to map tasks to subtasks in the task scheduler.

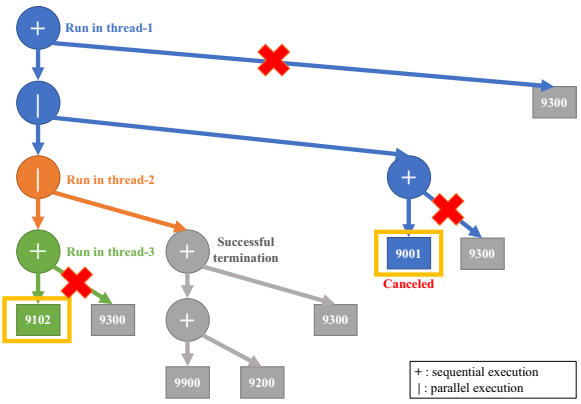
In ROS2-TMS, even when a task is running, the task can be stopped immediately by receiving a "cancel" voice request issued by the user or other cancelation commands. The cancel operation is realized by sending a cancel command to the subtasks currently running. It is an emergency stop function, and there is currently no



(a) Threading



(b) In operation



(c) Canceling

Fig. 10 Thread allocation for "Good morning" task

function to resume a task that has been stopped in the middle.

Even if the execution time is specified when starting subtasks, the execution is stopped immediately, and further serial operations of subtasks are terminated (Fig. 10b, c).

Error termination

If the subtask is not completed or canceled, its subtask node informs the task search node in the TMS_UR of the failure through the requested task control node or its upper-level task management node. When the task search node is informed of the failure, it sends a message to the speaker that the task has failed or has been canceled.

Tasks that combine subtasks

The task scheduler in ROS2-TMS can manage not only robots but also IoT devices such as beds, lights, and microphones, and ROS2-TMS can provide various services by combining robots and intelligent home appliances. Examples of services that are newly added to the ROS2-TMS, such as the "Good morning" and "Good night" tasks, are shown in Fig. 11. Both tasks combine the subtasks for the communication robot Double 2, a robotic bed, lighting, and speaker. For example, the "Good morning" task is defined as shown in Fig. 12.

As shown above, these tasks consist of sequential and parallel executions of several subtasks and are highly complicated. We confirmed that the cancel function is appropriately executed even for these complex tasks.

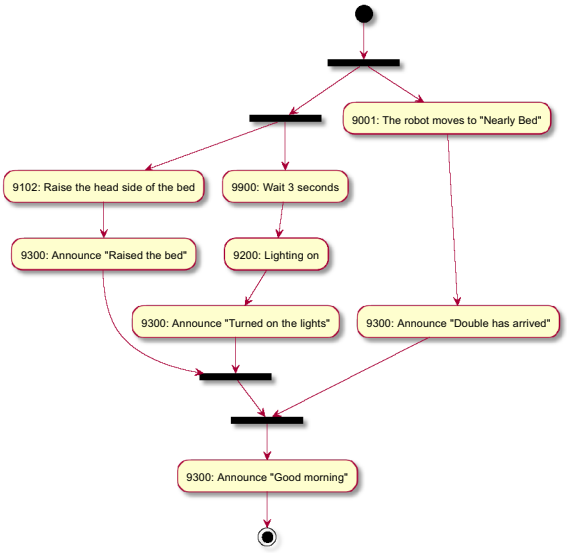
Service experiments using ROS2-TMS and big sensor box

We conducted an experiment to verify the operation of services with a voice interface using ROS2-TMS in Big Sensor Box (Fig. 2).

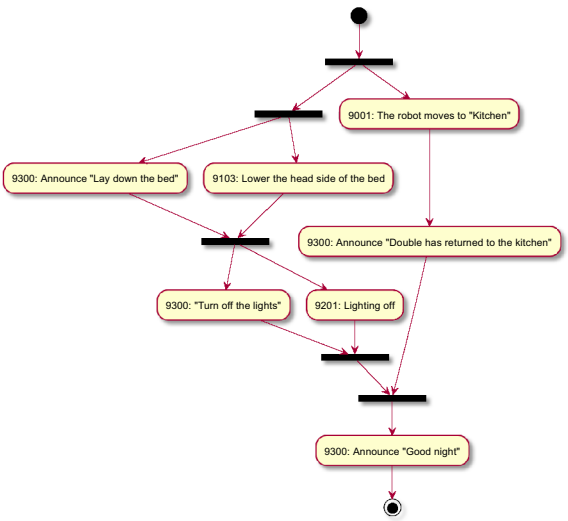
The big sensor box is a hardware platform for an informationally structured environment. As described in [1], it is located on the second floor of the COI building at the Ito campus of Kyushu University and is a habitable space with a living room, bedroom, and kitchen. In the room, there is a service robot, such as Double 2 and Double 3, and a motion capture system that can acquire the robot's position, a robotic bed, and a server that manages the room lighting. In this study, we interconnected them using ROS2 communication and experimented with providing services using ROS2-TMS.

We confirmed that ROS2-TMS can launch proper tasks according to a user's voice request through the microphone device and perform tasks such as user interaction (daily conversation), bed operations, lighting control, and robot motion by combining the proper information in the database.

The following subsections show how each task was performed. Detailed timecharts of each task are shown in Figs. 13 and 14.



(a) "Good morning" task



(b) "Good night" task

Fig. 11 Example of tasks combining subtasks

Daily conversation

When a user asks ROS2-TMS questions such as "ROS-TMS, What's the weather today?" or "ROS-TMS, How high is Mount Fuji?," proper responses were provided by Google Assistant (Fig. 15).

Lighting control

When a user requests tasks related to the lighting such as "ROS-TMS, turn on the lights" or "ROS-TMS, turn off the lights," ROS2-TMS chooses and executes the proper tasks by sending commands to the lighting management server in Big Sensor Box (Fig. 16).

Robotic bed control

When a user requests tasks related to a robotic bed, such as "ROS-TMS, raise the height of the bed," ROS2-TMS executes the proper tasks to control the robotic bed (Fig. 17a).

Robot control

When the user requests "ROS-TMS, Double, go to the kitchen," TMS_TS in ROS2-TMS generates a motion plan for the communication robot Double 2 to move to the kitchen. The coordinates of the kitchen are stored in the database and interpolated to generate motion commands in TMS_TS. In addition, if the user requests "ROS-TMS, Double, go to the bed," Double 2 moves to the vicinity of the bed by linking with the bed coordinates stored in the database. (Fig. 18a, b).

Execution of complex tasks

We implemented the "Good morning" and "Good night" tasks to confirm the sequential and parallel executions of subtasks. Both tasks consisted of several subtasks related to robot motion, bed operation, and room lighting control connected sequentially or in parallel. To confirm the flexibility of the task representation, we implemented these tasks with several different connections of subtasks and confirmed that these tasks were executed appropriately even with complex task structures.

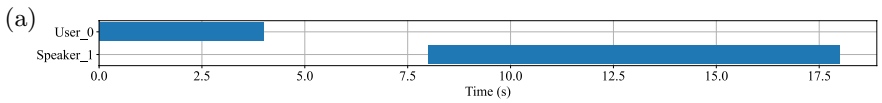
"Good morning" task

The "Good morning" task performs the following three subtasks in parallel.

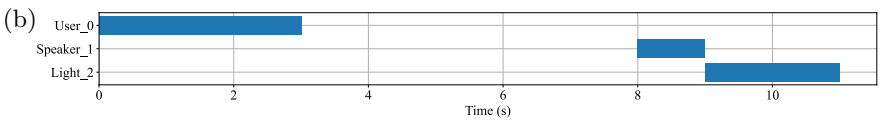
- Raise the upper body of the bed and announce "Raised the bed" from the speaker.
- Wait for 3 s, turn on the lights, and announce "Turned on the lights."

9102${"sec": 20.0} 9300${"announce": "Raised the bed"} + 9900${"wait_sec": 3.0} 9200 + 9300${"announce": "Turned on the lights"} + | 9001${"position": [8.65, 1.62, 0.0], "orientation": [0.0, 0.0, 0.280, 0.960]} 9300${"announce": "Double has arrived"} + | 9300${"announce": "Good morning"} +

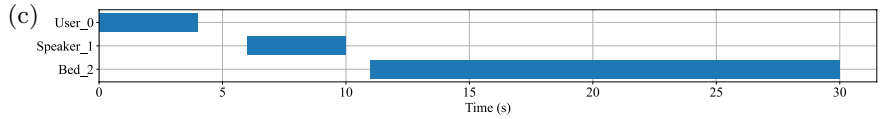
Fig. 12 Task notation of "Good morning" task



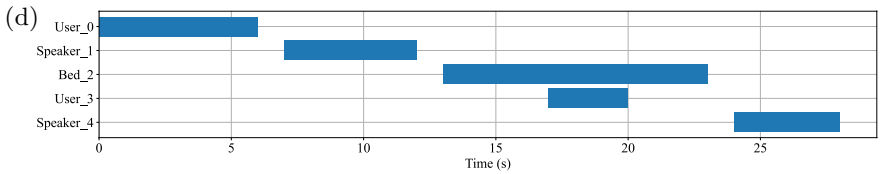
"Weather forecast" time chart. User 0: Ask the weather to microphone. Speaker 1: Answer the weather.



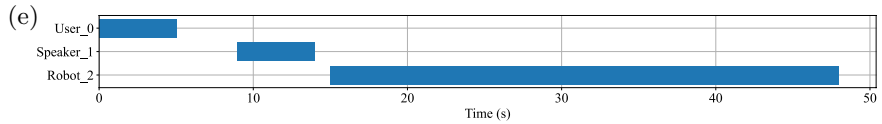
"Switching off the light" time chart. User_0: "ROS-TMS, turn off the lights". Speaker_1: Announcement at task startup. Light_2: Turned off.



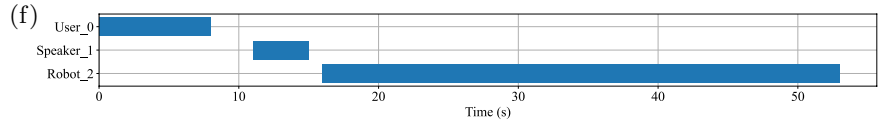
"Lower the head side of the bed" time chart. User_0: "ROS-TMS, lower the head side of the bed". Speaker_1: Announcement at task startup. Bed_2: Lowered the head side.



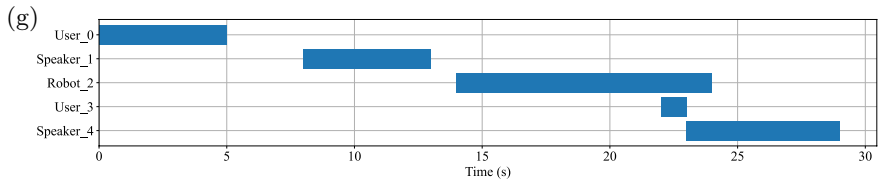
"Canceling bed control task" time chart. User_0: "ROS-TMS, lower the head side of the bed". Speaker_1: Announcement at task startup. Bed_2: Lowered the head side. User_3: "Cancel". Speaker_4: Announcement at task cancelation.



"Double goes to the kitchen" time chart. User_0: "ROS-TMS, Double, go to the kitchen". Speaker_1: Announcement at task startup. Robot 2: Went to the kitchen.

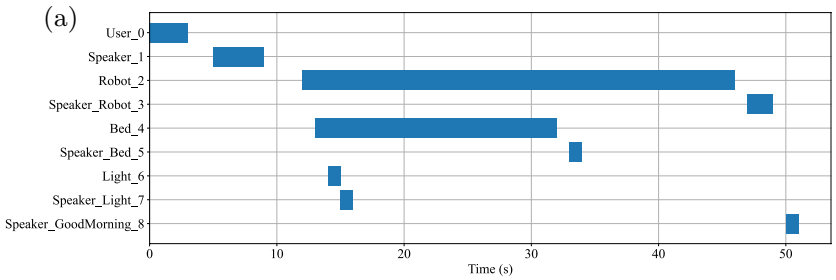


"Double goes to the bed" time chart. User_0: "ROS-TMS, Double, go to the bed". Speaker_1: Announcement at task startup. Robot_2: Went to the near side of the bed.

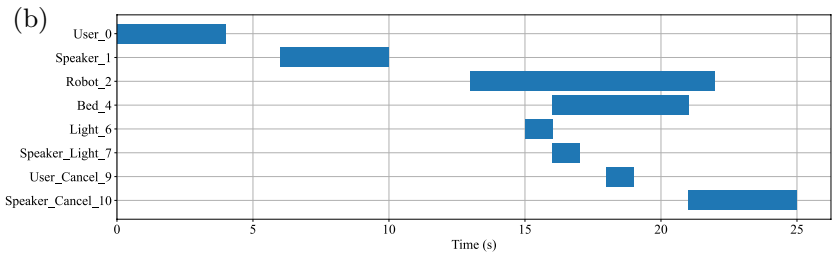


"Canceling move task" time chart. User 0:_"ROS-TMS, Double, go to the bed". Speaker 1:_Announcement at task startup. Robot_2: Went to the near side of the bed. User_3: "Cancel". Speaker_4: Announcement at task cancelation.

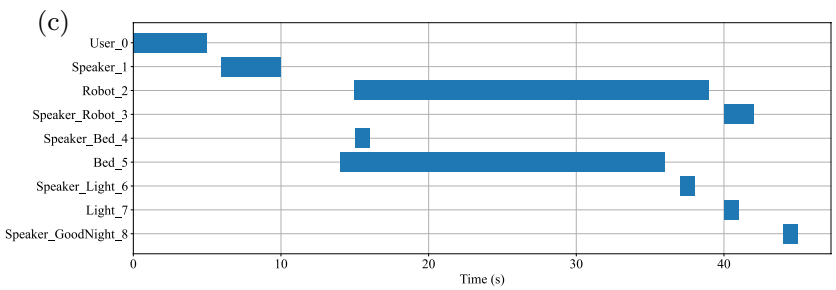
Fig. 13 Time chart of task execution (1 of 2)



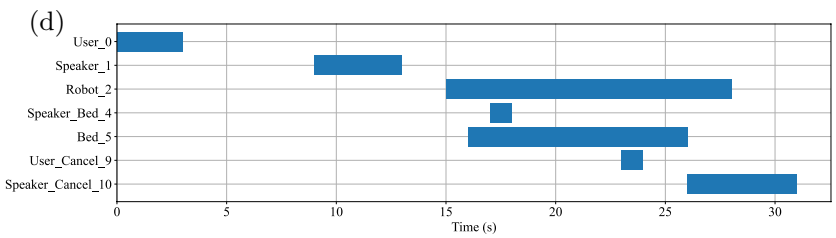
Execute "Good morning" task time chart. User_0: "ROS-TMS, good morning". Speaker_1: Announcement at task startup. Robot_2: Went to the near side of the bed. Speaker_Robot_3: Announced "Double has arrived". Bed_4: Raise the head side of the bed. Speaker_Bed_5: Announced "Raised the bed". Light_6: Turned on. Speaker_Light_7: Announced "Turned on the lights". Speaker_GoodMorning_8: Announced "Good morning".



Canceling "Good morning" task time chart. User 0: "ROS-TMS, good morning". Speaker 1: Announcement at task startup. Robot 2: Went to the near side of the bed. Bed 4: Raise the head side of the bed. Light_6: Turned on. Speaker_Light_7: Announced "Turned on the lights". User_Cancel_9: "Cancel". Speaker_Cancel_10: Announcement at task cancelation.



Execute "Good night" task time chart. User_0: "ROS-TMS, good night". Speaker_1: Announcement at task startup. Robot_2: Went to the near side of the kitchen. Speaker_Robot_3: Announced "Double has returned to the kitchen". Speaker_Bed_4: Announced "Lay down the bed". Bed_5: Lowered the head side of the bed. Speaker_Light_6: Announced "Turn off the lights". Light_7: Turned off. Speaker_GoodNight_8: Announced "Good night".



Canceling "Good night" task time chart. User 0: "ROS-TMS, good night". Speaker 1: Announcement at task startup. Robot_2: Went to the near side of the kitchen. Speaker_Bed_4: Announced "Lay down the bed". Bed_5: Lowered the head side of the bed. User_Cancel_9: "Cancel". Speaker_Cancel_10: Announcement at task cancelation.

Fig. 14 Time chart of task execution (2 of 2)

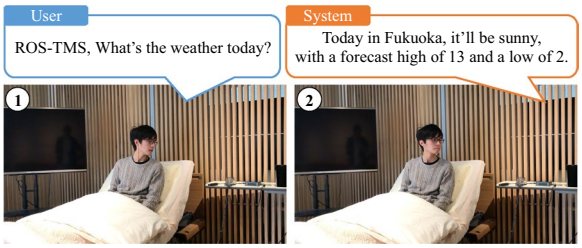


Fig. 15 Weather forecast

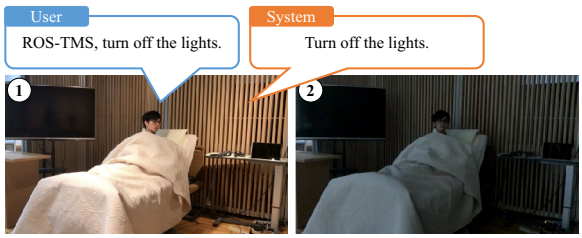
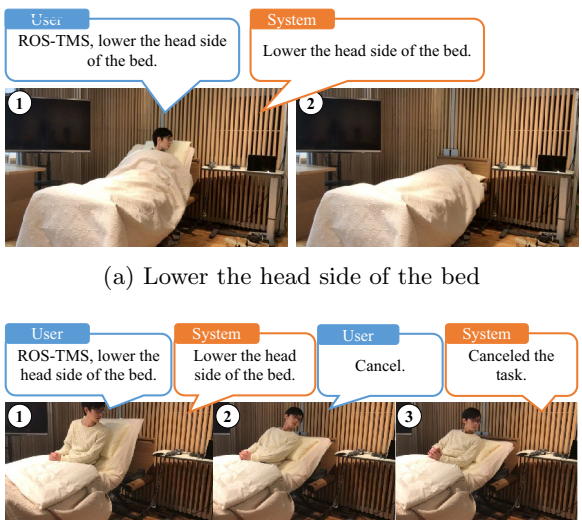


Fig. 16 Switching off the light



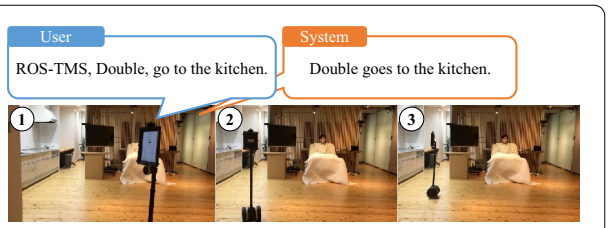
(a) Lower the head side of the bed

(b) Canceling bed control task

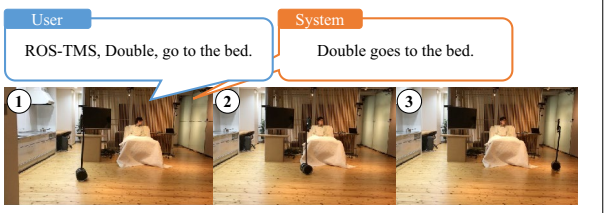
Fig. 17 Bed control task

- Move the communication robot Double 2 close to the bed and announce "Double has arrived."

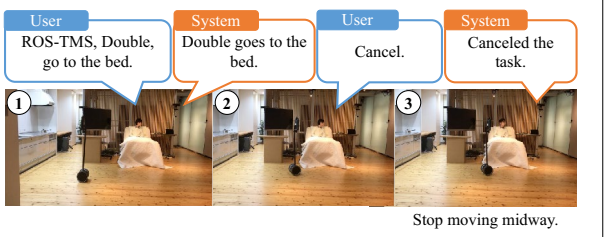
After completing all the subtasks, "Good morning" is announced from the speaker (Fig. 19a).



(a) Double goes to the kitchen



(b) Double goes to the bed



(c) Canceling move task

Fig. 18 Move task for communication robot

"Good night" task

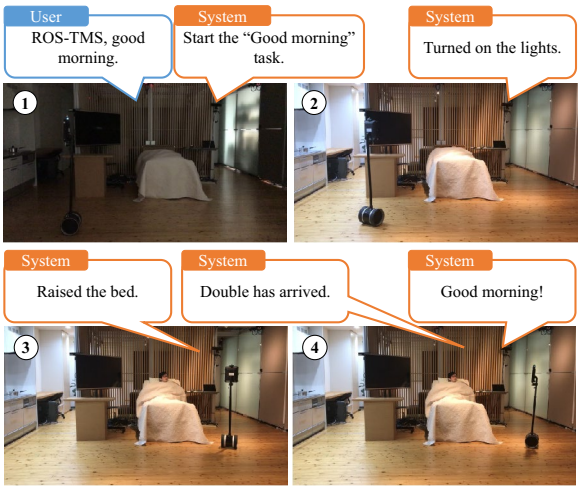
The "Good night" task executes two subtasks in parallel. In addition, there is a further parallel operation within the first operation. The following two operations are executed in parallel.

- A task lays the head side of the bed down and announces "Lay down the bed" from the speaker and then turns off the lights in the room and announces "Turn off the lights."
- After Double 2 returns to the kitchen, the speaker announces, "Double has returned to the kitchen."

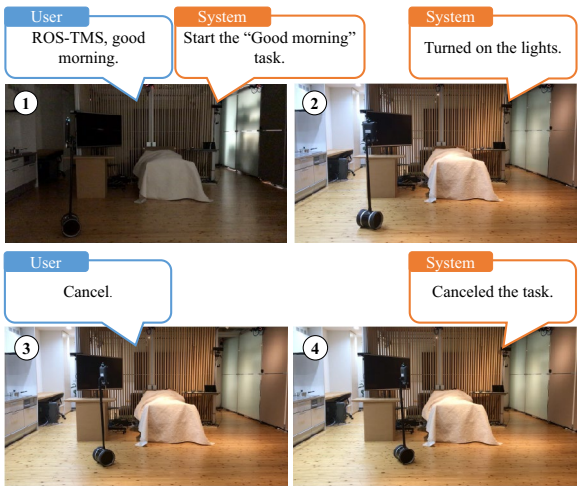
After completing all the subtasks, "Good night" is announced from the speaker (Fig. 20a).

Cancel tasks

If the word "cancel" is included in the user's command, TMS_TS sends a cancel command to the control nodes in all subtasks executing currently. When the control node in the subtask receives the cancel command, the control node sends the cancel command to the subtasks and the control nodes of the child subtasks executing

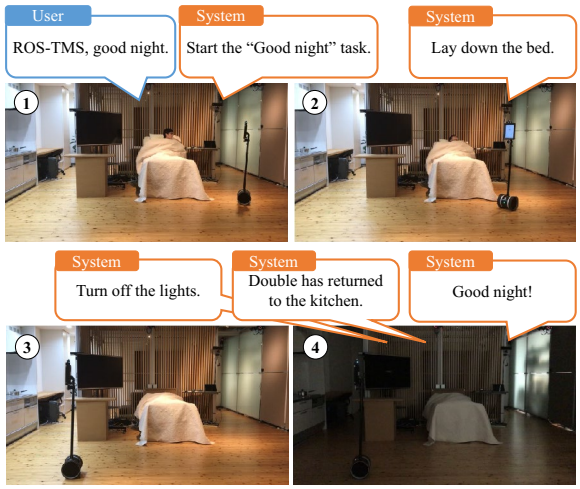


(a) Execute "Good morning" task

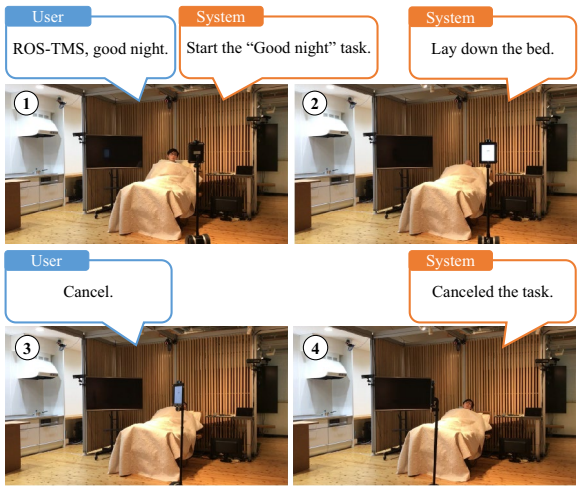


(b) Canceling "Good morning" task

Fig. 19 "Good morning" task



(a) Execute "Good night" task



(b) Canceling "Good night" task

Fig. 20 "Good night" task

in parallel. When all the control nodes in the subtasks receive the cancel command, the task is stopped (Figs. 17b, c, 19b, and 20b).

Conclusions

In this study, we presented a new software platform for an ISE, named ROS2-TMS, and demonstrated a few robot services using ROS2-TMS. ROS2-TMS is publicly available and can be downloaded from [20]. The differences between ROS2-TMS and the previous ROS-TMS system are summarized as follows.

- The middleware was changed from ROS to ROS2, and various modules were ported and newly developed.

  Currently, ROS2-TMS does not actively use security and QoS settings. However, as we continue our research, QoS will used to configure sensor streams, such as cameras, which have a large amount of data. For the robot movement subtask, we used Navigation2 and the behavior tree. The application of ROS2-TMS with this behavior tree is more flexible than ROS-TMS for moving in complex environments.

- The interaction with a user was realized using the Google Assistant API.
- The task scheduler was extended to manage not only robots but also various IoT devices such as intelligent beds, lighting, and speakers. This makes it possible to provide various services that combine robots and IoT devices.
- To improve safety, we added a new function that allows the user to stop running tasks by issuing a cancelation request to the ROS nodes using the ROS2 Action protocol.

Future works includes the following improvements.

Low-cost localization system

In this study, an optical motion capture system was used to estimate the position of the robot. Although this system can identify the robot position with high accuracy, it is considerably expensive. Currently, we are testing less expensive range-based localization systems such as Wi-Fi, RFID, UWB, and visible light [21], and we believe that low-cost autonomous mobile services can be provided by applying these technologies.

Services by health status

In this study, we developed a module for a wearable heart rate sensor. We will develop a service robot that recognizes not only the human's location but also the health condition using environmental sensors as a state analyzer module (TMS_SA, in Fig. 3) and provides proper services according to the health status of the individual.

Dynamic planning of tasks, such as autonomous robot movement

The autonomous movement of robots in large areas, including doors or elevators, requires coordination with surrounding actuators [12, 22]. Therefore, even for the same autonomous movement, the composition of the subtasks differ significantly between movement in a room and movement in a large area across rooms. For such environment-dependent tasks, we will extend the task scheduler so that the necessary functions can be dynamically linked.

In addition, when several tasks are executed simultaneously, it is necessary to monitor and appropriately control the progress of all the tasks currently running. Thus, we will design an efficient interface to present detailed and comprehensive information, including the task status, location of humans and robots, and human conditions.

Acknowledgements
This work was partially supported by the Cabinet Office (CAO) Cross-Ministerial Strategic Innovation Promotion Program (SIP), "An intelligent knowledge processing infrastructure, integrating physical and virtual domains" (funding agency: NEDO).

Authors' contributions
TI drafted the manuscript. RK constructed the study concept. AK and RK managed the study. TI and MS developed the system and carried out the experiments. All members verified the content of their contributions. All authors read and approved the final manuscript.

Funding
This work was partially supported by the Cabinet Office (CAO), Cross-Ministerial Strategic Innovation Promotion Program (SIP), "An intelligent knowledge processing infrastructure, integrating physical and virtual domains" (funding agency: NEDO).

Availability of data and materials
Not applicable.

Declarations

Competing interests
The authors declare that they have no competing interests.

Received: 25 September 2021 Accepted: 28 December 2021
Published online: 31 January 2022

References

1. Sakamoto J, Kiyoyama K, Matsumoto K, Pyo Y, Kawamura A, Kurazume R (2018) Development of ros-tms 5.0 for informationally structured environment. ROBOMECH J 5:24. https://doi.org/10.1186/s40648-018-0123-9
2. ROS.org | Powering the world's robots. https://www.ros.org/. Accessed 3 Aug 2021
3. ROS 2 Documentation: Foxy. https://docs.ros.org/en/foxy/index.html. Accessed 30 July 2021
4. Macenski S, Martin F, White R, Clavero JG (2020) The Marathon 2: a navigation system. In: 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp 2718–2725. https://doi.org/10.1109/IROS45743.2020.9341207. https://github.com/ros-planning/navigation2https://ieeexplore.ieee.org/document/9341207/
5. Sato T, Nishida Y, Mizoguchi H (1996) Robotic room: symbiosis with human through behavior media. Robot Auton Syst 18(1–2):185–194. https://doi.org/10.1016/0921-8890(96)00004-8
6. Brooks RA (1997) The intelligent room project. In: Proceedings of the 2nd International Conference on Cognitive Technology (CT '97). CT '97, p 271. IEEE Computer Society, USA
7. Park KH, Bien Z, Lee JJ, Kim BK, Lim JT, Kim JO, Lee H, Stefanov DH, Kim DJ, Jung JW, Do JH, Seo KH, Kim CH, Song WG, Lee WJ (2007) Robotic smart house to assist people with movement disabilities. Auton Robots 22(2):183–198. https://doi.org/10.1007/s10514-006-9012-9
8. Amazon Alexa Voice AI | Alexa Developer Official Site. https://developer.amazon.com/en-US/alexa. Accessed 23 Aug 2021
9. Google Assistant. https://assistant.google.com/. Accessed 23 Aug 2021
10. Kwak SS, San Kim J, Moon BJ, Kang D, Choi J (2020) Robots versus speakers: what type of central smart home interface consumers prefer? In: 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp 11397–11404. https://doi.org/10.1109/IROS45743.2020.9341748. https://ieeexplore.ieee.org/document/9341748/
11. Melo N, Lee J, Suzuki R (2018) Identification of the User's Habits based on Activity Information. In: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp 2014–2019. https://doi.org/10.1109/IROS.2018.8593873. https://ieeexplore.ieee.org/document/8593873/

12. Cavallo F, Limosani R, Manzi A, Bonaccorsi M, Esposito R, Di Rocco M, Pecora F, Teti G, Saffiotti A, Dario P (2014) Development of a socially believable multi-robot solution from town to home. Cognit Comput 6(4):954–967. https://doi.org/10.1007/s12559-014-9290-z
13. Simoens P, Dragone M, Saffiotti A (2018) The internet of robotic things: a review of concept, added value and applications. Int J Adv Robot Syst 15(1):1–11. https://doi.org/10.1177/1729881418759424
14. ROS2 Design | Actions. http://design.ros2.org/articles/actions.html. Accessed 4 Feb 2020
15. Double Robotics - Telepresence Robot for Telecommuters. https://www.doublerobotics.com/double2.html. Accessed 2 Feb 2020
16. Double Robotics—Telepresence Robot for the Hybrid Office. https://www.doublerobotics.com/double3.html. Accessed 15 Sept 2021
17. Overview | Google Assistant SDK | Google Developers. https://developers.google.com/assistant/sdk/overview. Accessed 2 Feb 2020
18. Julius. https://julius.osdn.jp/. Accessed 14 Sept 2021
19. Janome v0.3 documentation (ja). https://mocobeta.github.io/janome/. Accessed 2 Feb 2020
20. https://github.com/irvs/ros2_tms. Accessed 16 Sept 2021
21. Zafari F, Gkelias A, Leung KK (2019) A survey of indoor localization systems and technologies. IEEE Communications Surveys and Tutorials 21(3), 2568–2599. https://doi.org/10.1109/COMST.2019.2911558. arxiv:1709.01015
22. Martin F, Gines J, Vargas D, Rodraguez-Lera FJ, Matellan V (2018) Planning topological navigation for complex indoor environments. In: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp 1–9. https://doi.org/10.1109/IROS.2018.8594038https://ieeexplore.ieee.org/document/8594038/

Publisher's Note