Hanai *et al. Robomech J (2018) 5:21*
https://doi.org/10.1186/s40648-018-0120-z

ROBOMECH Journal

RESEARCH ARTICLE Open Access

Design of robot programming software for the systematic reuse of teaching data including environment model

Ryo Hanai[1]*, Kensuke Harada[2], Isao Hara[1] and Noriaki Ando[1]

Abstract
The motion of the robot to realize assembly work includes the part where the reuse of the motion adjusted for the real objects is effective and the part where automatic generation in the simulator is suitable. In order to smoothly teach such assembly work, teaching software that enables to combine previously used motions and perform overall adjustment of the workflow and integrated environment representation in the simulator is expected. Some teaching tools focus on the function of making robot motion in detail, and it assumes that the adjustment of the whole workflow including system layout using the real work environment. For this reason, the environmental expression is not sufficient for the above purpose. Although offline teaching tools and motion planning tools are rich in the representation of the environment, there are not many studies on a systematic reuse mechanism of motions adjusted in a real environment, including environment representations. In this paper, we present software design to solve this problem and the implementation of it as a plugin for Choreonoid. By an experiment, we confirmed that we can describe a comparatively complicated assembly work with the proposed software.

Keywords: Teaching, System integration, Manipulation, Task model, Reuse

Introduction

How to build flexible robot systems that are easy to cope with changes in product design and production volume is an important issue in robot system integration. Basically, there are two approaches for such flexible robot systems.

1. Reduction of manual work by raising an intelligence level of robots.
2. Improvement of engineering work related to robot introduction including teaching.

The first approach aims to make computers recognize objects and environment, and generate motions of the robots that enables intended tasks [1, 2]. This results in the reduction of information given by humans explicitly when each system is built. This is an actively researched field, and many recognition and motion planning algorithms have been proposed. In recent years, methods based on machine learning using data obtained from the real environment have also been explored [3, 4].

In the latter approach, it is assumed that humans program the robot motion over relatively detailed parts. The focus of this approach is on software or system design to make teaching work efficient or easy for unskilled users. For example, Rethink Robotics released a new teaching software that enables teaching by combining graphical control blocks. In many of the collaborative robots released in recent years, efforts are made to make teaching work more efficient by integrating direct teaching to designate positions in a real environment with teaching tools [5–7].

If accomplishing a task goal automatically is possible by an intelligent robot, the work required for the robot introduction is greatly reduced. However fully automatic approaches have many difficulties. For example, in order to generate motion of robot with contact, which is the core of assembly work, in simulation, it would be

*Correspondence: ryo.hanai@aist.go.jp
[1] Robot Innovation Research Center, National Institute of Advanced Industrial Science and Technology, 1-1-1, Umezono, Tsukuba, Japan
Full list of author information is available at the end of the article

necessary to model the details of objects such as rigidity, material and surface condition in addition to highly accurate geometry. The models of the dynamic behavior of the robot and its lower control system are also needed. This is difficult for users who aim for a simple introduction of robots. Rather, it is easier in many cases to make this type of motions using the real environment. Likewise, as for non-rigid bodies such as cables, modeling methods and algorithms for dealing with them are not established. Information to be defined in advance tends to be large to perform a high degree of automation. This is a problem especially for users planning to introduce robots experimentally.

The method of making the robot learn through the trials in the real environment has the possibility of reducing human intervention if it can be applied successfully. However, such trials are not necessarily possible in all tasks, robots and work environments. There is a problem that it takes a long time to make a sufficient number of trials. For these reasons, it is the case that a method of teaching the details of robot motions is still common.

Directly writing a sequence of detailed robot motions is efficient at least when one system construction is performed. However, when similar systems need to be implemented or reproduced, this approach tends to lose its efficiency. Similar systems are often required when a part of a product is updated or the assembly system for the product is improved. In the build-to-order production, some part variation might be added. The pre-built system might be reproduced in other places with some layout changes in order to cope with the increase in production amount.

This paper discusses what kind of support is possible for the systematic development of such similar systems from the aspect of software and presents a concrete design of the support software. As for the motions that are difficult to automatically generate and need some adjustment in the real environment, we consider accumulating the motion data in a form easy to reuse. Meanwhile, we integrate the environment and parameters defined for each motion data so they can be adjusted using the whole environment. It is troublesome to teach motions such as the one that transports a lifted part to a next assembling position in a complicated environment with many objects, each time the layout of the system is changed. With the integrated environment model, this kind of motion is relatively easy to generate using kinematic motion planning.

This paper also presents the implementation of the above-mentioned concept as a plugin for Choreonoid [8], which is an integrated development environment for robot tools.

In the following sections, we will focus on assembly task which is the work expected to be further automated by the robot.

Related work

The software proposed in this paper aims to give means for systematic reuse of the motion data by system integrators and users.

Most teaching tools of commercial robots focus on directly describing robot motion. This is because this approach enables to fully utilize the functions of each robot when implementing tasks, has a flexibility to encode users' know-how explicitly. Various kinds of teaching tools friendly to users have been proposed. Intera 5.0 [9], developed by Rethink Robotics, describes a program of a robot by graphically combining blocks similar to a syntax tree of a programming language. However, descriptions generated by these tools have less information on work content and dependence between submotions is not well structured. If the information such as the contents of work, robots, and tools used are not sufficient, it is difficult to determine whether a certain motion pattern can be reused well for a target task. Therefore, in our software, 3D models of related objects are managed together with motion patterns of robots. In addition, it is possible to link data such as images, movies, documents as a means to supplement information which is difficult or time-consuming to represent in the 3D model environment.

In this paper, we use the word task as a unit of reuse and call the representation for it task model. How to design the task model is up to users for the reasons discussed below. In robotics, the term task is used as a concept closely related to work content. The word task focuses on the objectives of work. The functions of robots to achieve the tasks are called skills. Ogawa et al. say task expresses "what to do" and skill expresses "how to do" [10].

However, it is difficult to define these concepts generally for software expected to be used for various applications completely independent of target systems, application or abstraction level. Many authors defined skills suitable to their problems and applications. Huckaby [11] presented a framework to enable knowledge transfer between processes, systems and application domains. The skills they defined to describe assembling tasks included recognition, the use of tools. Szynkiewicz et al. [12] defined skills for bimanual manipulation. Task models for taking an elevator [13], the use of a tool [10] and picking [14] were also presented. In practical task programming, fairly low-level operations such as opening/closing a gripper and joint-space motions of robots are sometimes included in skills.

In our approach, task models are user-defined entities. We assume that skillful users with the experience of robot programming design the task models and other users use them. This idea of users and experts are mentioned in Ref. [15]. We have also worked on explicitly expressing the process of implementing tasks based on this idea [16]. We expect to support the skillful user to improve the design and accumulate know-how while gaining feedback from ordinary users instead of giving a complete task model set from the beginning.

Tasks and skills have hierarchy [13, 17]. Tasks are decomposed into sub-tasks. Thomas et al. defined these words in a bottom-up way in their framework called UML/P [15]. They called commands of the system devices elemental action (EA), a network of EA skill, and a network of skills task. They used a different term "process", referring to a larger unit composed of tasks.

Our approach is close to UML/P. A set of commands are implemented as controllers. Commands are integrated into a task model and task models are integrated into another layer, which is a workflow. Task models are designed by skillful users to execute the commands. The granularity of the task is chosen by the skillful users. There is a difference that UML/P does not have 3D models or a mechanism to integrate them.

In real workplaces, it is often difficult to abstract away the differences of robots, behaviors of controllers, and other implementation methods. Users' know-how is sometimes encoded in the way of utilizing a specific robot. We think that the accumulation of implementation level information is also important. So, the task model of our framework is not purely the work purpose. It allows implementation level descriptions.

Understanding of a motion data of a robot may be relatively easy if a user created it by himself. However, it is difficult when the user looks back to it after a while or shares it with other users. 3D models help users to see what was assumed for the environment and what was expected to happen as a result of executing the motion. In reality, there are cases where some information is difficult or troublesome to represent with the 3D models. We manage various data together with the task model for this information.

Requirements for teaching software

The approach we take is to propose software that enables users to perform the reuse process of robot motion data in a systematic manner. Specifically, we aim to provide software for designing the data reuse unit and managing the designed data together with adjusted parameters, 3D models and other data necessary for understanding the work content and combining these data to describe a longer work sequence.

In the rest of this section, we will discuss the requirements for this software using assembling of a mechanical unit composed of several parts as an example. Given a set of parts and a goal image of an assembled unit, we first consider the order of assembling these parts and how they are assembled. Next, we consider what kind of robot's motion enables each assembly step. As is mentioned in "Introduction" section, the approach of generating this kind of motion automatically by giving necessary information does not pay in many cases. Therefore, we consider using human-designed motions that are previously adjusted for the same or similar objects. The following requirement arises from this discussion.

R1-1. Previously created motion patterns of robots are accumulated, and a longer operation sequence can be described by combining them.

What kind of motion is suitable for assembling some parts mainly depends on the parts. The structure of the robot and hand, assembling method, tools and jigs used for the assembling also affect the choice of the motion pattern. Therefore, we expect that motion patterns are extracted efficiently using the information related to the target assembly.

R1-2. Quick access to target data is possible from the information related to the task.

R1-3. Information for users to understand what each data is like can be attached.

After the order of assembling the parts and the motion of assembling them are determined, the layout of the working environment such as the supply positions of the parts and the assembling position must be decided in order to develop the system that performs the assembling operation. Since these positions are usually different from the recorded ones in the stored data, they need to be adjustable at the time of data reuse. It is difficult to apply motions designed to specific parts or hands to other parts directly. However, in some cases, it is possible to define a motion pattern applicable to a certain kind of parts group by parameterizing the motion empirically. In such cases, it is worth considering to design patterns with high versatility so tasks can be implemented with a small number of patterns.

Some example parameterization would be the grasp position of parts at the time of assembly, the designation of the hand to be used when multiple hands are present, the tightening torque at the time of screwing and other control parameters. Based on this discussion, we add the following requirements.

R2-1. It is possible to change the layout of the work environment when combining the data.
R2-2. Flexible parameterization of motion patterns is possible.

We consider the use of 3D models as a representation of the work environment. 3D models are widely used in offline teaching tools or other robot applications, but their objectives are varied. In our software, the 3D models are used for the following two purposes.

1. Visualization for users.
2. Environment for kinematic motion planning.

Generally, the number of parts and work procedures are large in assembly work. It is usual to add minor improvements to once deployed system by adjusting the motions of robots or changing the grasp position of the objects. Visualization of the objects in the whole environment helps a lot to design the layout of the objects in the virtual environment and adjust motion patterns. As described in the previous section, assembly operations are performed by reusing fixed patterns. If motion planning can be used for the generation of motion connecting these assembling motions, the teaching work will be reduced especially when the workspace layout is changed. This can be implemented by introducing specific task models which execute trajectories avoiding obstacles in the environment by using motion planning internally. To realize this, the environment as the representation of obstacles for the robot is needed.

It should be noted here that only items related to each motion pattern are registered naturally. So, it is necessary to create the representation of the whole environment by integrating 3D models associated with each partial motion pattern.

R3-1. Each motion pattern should be associated with 3D models as an environment representation.
R3-2. These environment models can be integrated easily when combining the motion patterns.

Finally, teaching is performed by human users. The design of the user interface is an important factor.

R4-1. GUI should present necessary and sufficient information at each phase of teaching.

Design of the teaching software

This section describes the details of software design to satisfy the requirements listed in "Requirements for teaching software" section.

Task model and a unit of data reuse

Corresponding to R1-1, Fig. 1 shows the unit of data accumulation and reuse. We call a semantically delimited motion pattern *task model*. The task model is a user-defined entity for the reasons below. First, since there is no established way to delimit motions, there are different ways of abstraction depending on the task designer. Second, the domain of the target work is not closed, and it is impossible to prescribe a set of task models used for the task domain in advance and offer them together with the teaching software. Our task model consists of:

1. A state-machine representing a motion pattern of the robot.
2. A set of task parameters allowing the adjustment of the above motion pattern.

We consider a sequence of motion such as "pick one part and assemble it to another part" as a unit of reuse. This is represented as a sequence of command execution offered by robot controllers. Strictly speaking, the state-machine is a mid-level task program rather than a simple motion pattern. It describes command calls to lower-level controllers depending on each state. The control flow of the program may change according to the result of the previous command execution. It is assumed that commands that can be executed by the low-level controller are defined. Apart from the basic ones, it is difficult to share the interface of the controller of the robot. So we do not premise a specific controller interface. If the commands referenced by a state-machine is implemented by the controller, the task model is executable on that controller. If not, the task model is not executable on that controller.

The task parameters correspond to R2-1 and R2-2. They enable to change the layout of objects and adjust motion patterns.

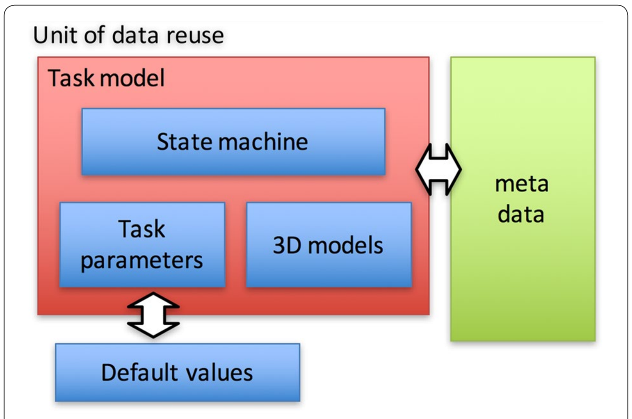


Fig. 1 Unit of data reuse

When defining a state-machine, expressions using task parameters are described as arguments of command calls in each state. The goal positions of the hand or joint angles are not always given directly by users. Appropriate task parameters are chosen to model each task. These expressions describe how to calculate the motions of the robots from these selected parameters.

Figure 2 illustrates the relations among the above-mentioned elements using an example. We assume a low-level controller is given and the controller has commands such as *moveArm* and *closeGripper*. *hole position* is a task parameter. A trajectory of screwing are arguments for *moveArm* command executions. Mapping functions define the trajectory, which depends on the value of *hole position*.

The unit of data reuse is the combination of a task model, metadata which is information added by users, and initial values of task parameters. The "3D models" section, "Meta-data" section, the initial value "Default values of task parameters" section will be explained in detail.

3D models

In order to satisfy R3-1, we allow users to register 3D models in each task model and associate them with some task parameters. This mechanism enables to change the associated task parameter by moving the 3D model in the simulator interactively. As a result, the motion of the robot changes indirectly.

This mechanism is not sufficient to support the movement of items caused by the robots. When expressing the environment in the 3D model, we expect that environmental changes due to robot motion will be reproduced in the simulator. Otherwise, it is difficult to check if the combined motion sequence works well and the intended assembly will be achieved. In order to express the environmental changes caused by the robot, we add a mechanism to describe the changes in connections between hands and objects at the time of grasp and release actions, and changes in the connection between objects when they are assembled. These side effects can be added to each command execution.

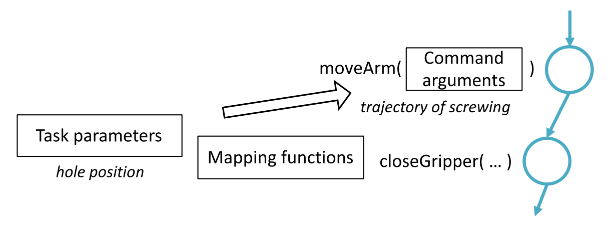


Fig. 2 State machine for a task model

Meta-data

We introduce an additional mechanism to annotate task models with texts, images, movies and other documents corresponding to the requirements R1-2 and R1-3. The meta-data is used to make task models searchable. The system uses the meta-data as clues to retrieve task models.

From the search result, a user selects a motion pattern that can be used for his target assembling work. At this time, if the user does not understand what kind of motions they are, the user can not evaluate their availability. Our tool provides two confirmation means. One is the execution of the motion in the simulator. The other is the meta-data. The meta-data plays a dual role of clues for the user to understand the contents of tasks and the key of the search.

Another view of this mechanism is the unified management of related data for the tasks. Users associate what they think is important for each task model. Concretely, this is used by users to present know-how on task implementation and to consider the re-usability of task models. The texts typically include but not limited to generic terms or model numbers of items related to the task, verbs expressing actions used for the task, phrases describing the task objectives. Extensive word sense is employed for texts because we assume the users who have not sufficiently formalizing what words are appropriate to annotate their tasks. If it is possible to give a set of appropriate words, it may be provided as a dictionary.

The images typically include photographs of objects such as parts, fixtures, tools or devices, key poses of robots while performing the task, and geometric relations between objects when they are assembled. This idea is borrowed from information management tools such as *Evernote* [18]. The concept of the annotation is *Capture what's on your mind on tasks*. Know-how for task implementation includes various things. Some are well represented by data suitable for tool interpretation such as CAD models. However, some are useful to assist human understanding but do not have a well-accepted way of representation. So the framework does not prescribe strictly what kind of data should be associated.

Default values of task parameters

Task models are recorded together with default values of their task parameters. There are two reasons for this.

The first reason is to make task models executable as a partial motion pattern. After searching for a task, users expect to check its contents in the simulator view. The task model only describes the relationship between various parameters and the motion pattern of the robot. In order to execute it, each task parameter needs to be initialized with some appropriate value.

The second reason is that parameter values adjusted in the real environment is worth reusing as mentioned in the introduction. They include values adjusted to archive the task efficiently and robustly. Some of the examples are control parameters such as stiffness, thresholds for mode switching, parameters to control trajectories such as movement speed or waiting time in some state.

Workflow

We call a longer motion sequence obtained by combining tasks *workflow* (R1-1). Figure 3 shows an example of the workflow. The workflow is also expressed as a state machine diagram like a task. Nodes of the workflow are tasks instead of command executions. The syntax elements are sequential execution, conditional branch, the merge of the control flow, initial node, final node, and task node. The conditional branch switches control flow depending on the result of the previous task execution. It is possible to describe an expression using flow parameters explained in "Flow parameters" section for a more complicated condition. The task node has a triangular port for connecting control flow. On the other hand, a node representing a flow parameter described later has only a round port. A node representing a 3D model node has a square port together with an image icon.

Flow parameters

If there is data that fits exactly to a target assembly task, it can be used as it is in a new workflow. However, there are many cases where some adjustments are needed such as the change of the position of an object, the replacement of some part. This is realized by changing task parameters and 3D models (R3-1, R3-2).

Next, we need to think about what kind of operations are suitable to make this adjustment. There is a problem with the method of directly rewriting the value of the parameter each task model has. Consider the situation where a robot places a screw in a hole and tightens it with a screwdriver. If this is represented by two task models, the screw that appears in two task models is the same entity. If the result position of the first task execution is not the same as the initial position of the second task, the whole workflow is not consistent. It takes time for users to adjust all the values of the respective task parameters to make the workflow consistent when the number of tasks and parts in the workflow is large, and fine adjustment of the layout is repeated. Therefore, we introduce a flow parameter which is a parameter that can be referred to throughout the entire workflow. Task parameters which are expected to keep the same value over several tasks are associated with the flow parameter. A user can graphically make a new association by connecting a port of a task node representing a

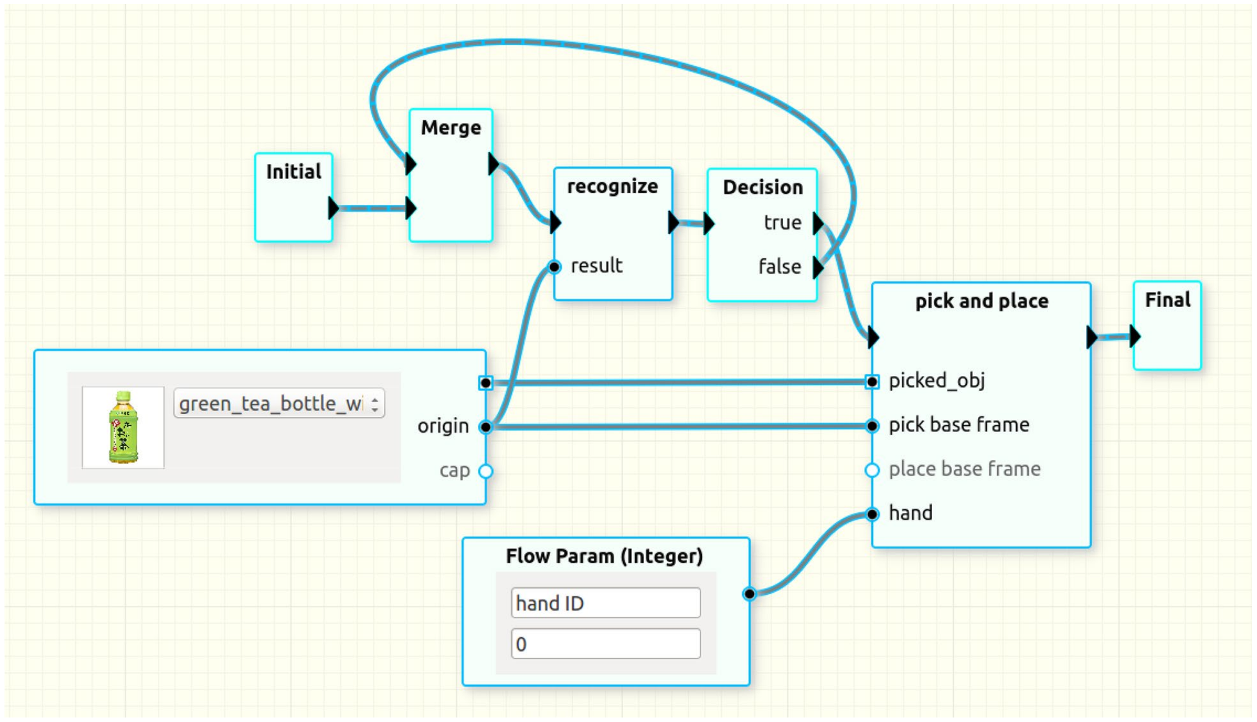


Fig. 3 An example of workflow

task parameter and a flow parameter node as shown in Fig. 3.

A round port of a task node represents a task parameter of it. When this port is connected to a flow parameter node, the value of the task parameter is synchronized with the value of the connected flow parameter. That is, when a certain task changes the value of a flow parameter, the value of the task parameter of another task linked to that flow parameter also changes. The read/write timings are determined by the definition of the tasks.

Sharing and replacement of 3D models in a workflow

If multiple tasks and flows have the copies of 3D models, the delay of model loading unacceptably increases. The size of the database also becomes remarkably large. To avoid this, 3D models are registered in advance and each task and flow have references to them. This 3D model data registered in advance is called model masters. Replacement of a 3D model in a workflow is performed using this model master. Consider a case where a user wants to replace a 350 ml plastic bottle used in an existing task with a 500 ml plastic bottle. In this case, the user add a 3D model node and make it refer to a master of the tall bottle.

In Fig. 3, the node with the image of a plastic bottle is the 3D model node. The 3D model node has a square port that can be connected to a square port of a task. In addition to robot motion, the task performs attach and detach operations on 3D models as described in "3D models" section. The square port is used to change the target of these operations.

Frames can be defined in addition to the origin on 3D models. For example, you can define screw hole positions with the names of *hole*1 to *hole*4 for a part with 4 screw holes. Flow parameters can be associated with task parameters. In the same way, these frames defined on 3D models can also be associated with task parameters. These frames are a kind of flow parameters and represented as round ports on 3D model nodes.

Output to parameters

The values of task parameters and flow parameters sometimes need to be updated while executing a task. A typical use case is recognition of an object. The recognized position and posture is usually used in subsequent tasks. This pattern can be implemented by outputting the recognition result to an appropriate flow parameter in the task of recognition and associating the flow parameter with a task parameter of another task that uses the value. In Fig. 3, the position of the plastic bottle is updated using a recognized value.

User interface

We will describe the design corresponding to R4-1. In the case of mass production, the roles of end users and system integrators are comparatively separated. The system integrators design systems taking a long time. However, in order to utilize flexible robot systems, it is necessary for the end user to be more involved in engineering work.

We classify users of the software into two groups: *User* and *Expert* and offer perspectives suitable for their typical use cases. *Expert* has a comparatively high level of knowledge on robots, programming and system integration, which is required to design task models. On the other hand, *User* has limited knowledge of them. *Expert* typically indicates engineers ranging from robot makers, system integrators to skillful people of end-user companies (Fig. 4). *Expert* designs basic patterns of data reuse. *User* customizes the patterns for specific objects or system layouts and combines them to describe workflows. *User* also manages meta-data.

Perspective for User

Figure 5 shows the perspective for *User*. The window consists of 4 views, In the search view, *User* enters keywords such as "gear" or "screwing". Then *User* selects from the search results. Various information related to the selected task model is shown in each view. Meta-data related to the task model is shown in the meta-data view. In the scene view, the robot and 3D models related to the task model is presented. *User* also checks the motion in the scene view.

Next, *User* combines the search results to describe a workflow. By the operation of dragging from the search view into workflow view, the selected task is added to the workflow. The connection among added tasks and flow parameters are edited in the workflow view.

Visibility of parameters

In addition to the control flow of the program, displaying various parameters and their relationships in one figure is suitable to understand the overall structure of a workflow. However, excessive information has a risk of making the figure difficult to understand, when the number of parameters and task nodes is large. The user interface needs some ingenuity to control the amount of

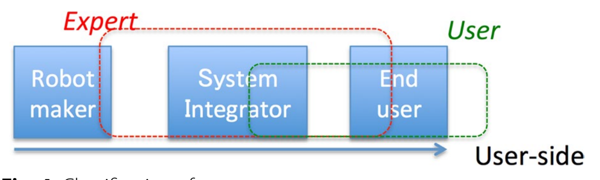
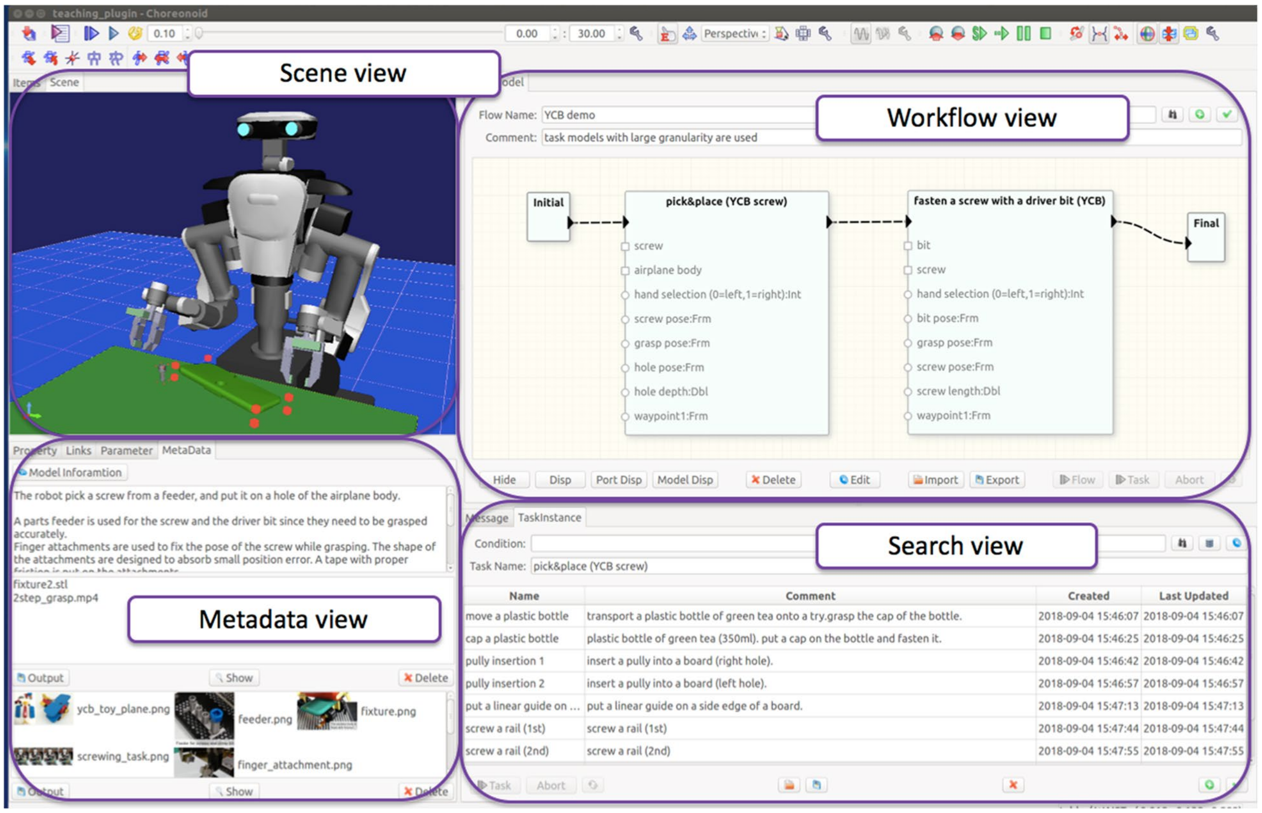


Fig. 4 Classification of users



Flow Name: YCB demo

Comment: task models with large granularity are used

The robot pick a screw from a feeder, and put it on a hole of the airplane body.

A parts feeder is used for the screw and the driver bit since they need to be grasped accurately.
Finger attachments are used to fix the pose of the screw while grasping. The shape of the attachments are designed to absorb small position error. A tape with proper

fixture2.stl
2step_grasp.mp4

Condition:
Task Name: pick&place (YCB screw)

Name	Comment	Created	Last Updated
move a plastic bottle	transport a plastic bottle of green tea onto a try.grasp the cap of the bottle.	2018-09-04 15:46:07	2018-09-04 15:46:07
cap a plastic bottle	plastic bottle of green tea (350ml). put a cap on the bottle and fasten it.	2018-09-04 15:46:25	2018-09-04 15:46:25
pully insertion 1	insert a pully into a board (right hole).	2018-09-04 15:46:42	2018-09-04 15:46:42
pully insertion 2	insert a pully into a board (left hole).	2018-09-04 15:46:57	2018-09-04 15:46:57
put a linear guide on ...	put a linear guide on a side edge of a board.	2018-09-04 15:47:13	2018-09-04 15:47:13
screw a rail (1st)	screw a rail (1st)	2018-09-04 15:47:44	2018-09-04 15:47:44
screw a rail (2nd)	screw a rail (2nd)	2018-09-04 15:47:55	2018-09-04 15:47:55

Fig. 5 Perspective for User

information presented in the workflow view. In order to alleviate this problem, we introduce a function to switch the visibility of parameters.

Basically, the design of workflows consists of two phases. Control parameters used for an assembly operation between specific parts are not often changed, once the adjustment is made. When the layout of the objects and the robot is changed, setting these control parameters invisible makes the other adjustment such as the design of the working environment easy.

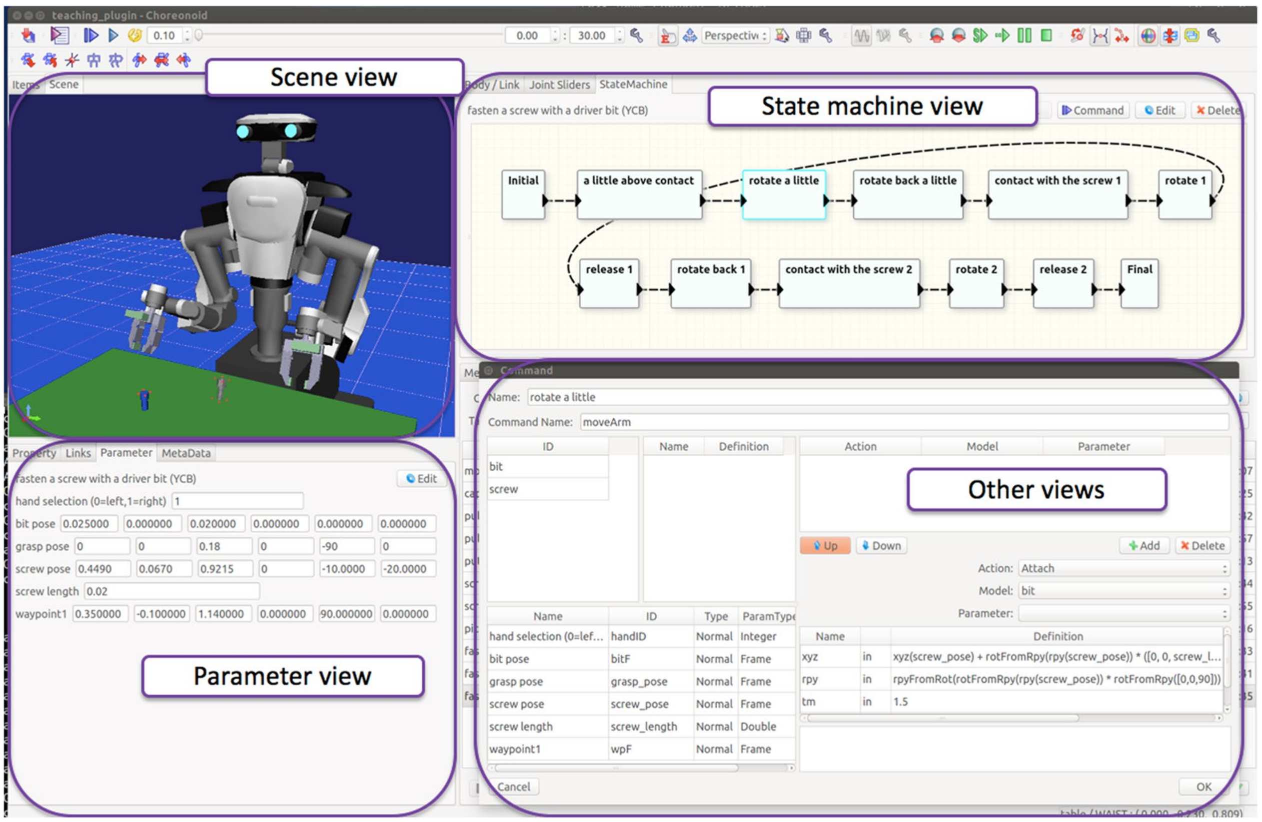
To enable this, task parameters can be set to invisible in the workflow view. In addition, in order to improve the operability, the GUI has functions of enlargement/reduction, the shift of display area, switching of edit mode. By setting the edit mode off, it is possible to prevent mistakenly changing parameters or structure of workflow during the operations such as search and execution of motions.

Perspective for Expert

Figure 6 shows the perspective for *Expert*. *Expert* mainly designs task models. For this purpose, views for defining the information necessary for the task model is prepared. The state machine view is used to design the structure of state machines. The command list defined by the low-level controller is shown. A command node is added by dragging them into the state machine view. The parameter view is used to define task parameters and changing values of them. When a command node is selected in the state machine view, information of the node is displayed in a separate dialog. Expressions to calculate command arguments from task parameters are edited in the dialog. The task model can also be defined in a text file. Currently, YAML format is supported. Some people prefer to import the task model after exporting and editing it in YAML format once at a time.

Implementation

The system architecture is shown in Fig. 7. The framework was implemented as a plugin for Choreonoid [8], which is an integrated development environment for robot tools with a powerful plugin-based extension mechanism. The plugin implementing the framework is shown as *Teaching core plugin* in the figure. All the data including the motion patterns designed by *Expert* and associated meta-data are managed by a relational database. SQLite [19] is used as a database management system. The stored data can be exported to YAML files and



fasten a screw with a driver bit (YCB)

fasten a screw with a driver bit (YCB)

hand selection (0=left,1=right)	1					
bit pose	0.025000	0.000000	0.020000	0.000000	0.000000	0.000000
grasp pose	0	0	0.18	0	-90	0
screw pose	0.4490	0.0670	0.9215	0	-10.0000	-20.0000
screw length	0.02					
waypoint1	0.350000	-0.100000	1.140000	0.000000	90.000000	0.000000

Name: rotate a little

Command Name: moveArm

ID
bit
screw

Name	ID	Type	ParamType
hand selection (0=lef...	handID	Normal	Integer
bit pose	bitF	Normal	Frame
grasp pose	grasp_pose	Normal	Frame
screw pose	screw_pose	Normal	Frame
screw length	screw_length	Normal	Double
waypoint1	wpF	Normal	Frame

Action: Attach

Model: bit

Parameter:

Name		Definition
xyz	in	xyz(screw_pose) + rotFromRpy(rpy(screw_pose)) * ([0, 0, screw_l...
rpy	in	rpyFromRot(rotFromRpy(rpy(screw_pose)) * rotFromRpy([0,0,90]))
tm	in	1.5

Fig. 6 Perspective for Expert

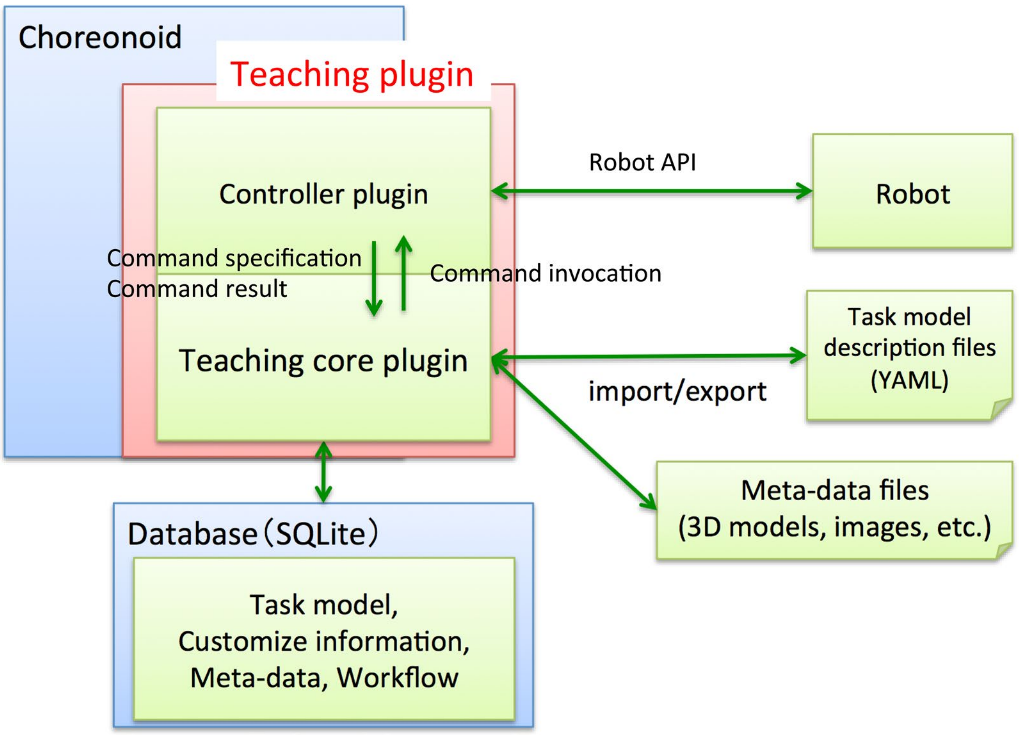


Fig. 7 System architecture of the framework

imported after editing with a text editor. YAML files are also useful for the cooperation with other tools. One possible cooperation is to import a task model generated by a tool to analyze human demonstrations.

Controllers used in the task models are implemented as separate plugins. This is shown as *Controller plugin* in the figure. *Controller plugin* needs to implement a common interface so that the commands defined by it are called by *Teaching core plugin*. The controllers can be implemented by directly mapping standard API of robots or by using different abstractions. *Teaching core plugin* obtains the list of the commands when the *Controller plugin* is loaded, calls the commands based on the command specification, and branches based on the result of the command execution.

Experiment

In this section, we show how a robot motion sequence is created using an example of YCB airplane [20] assembly. The robot used in this example is UR3 made by Universal Robots with a dual-arm configuration. This task consists of four parts: a partially assembled airplane body, a main wing, and two screws. The goal of this task is to assemble these four parts into one. Figure 8 shows these parts and how they are assembled.

The assembly procedure is as follows. First, the main wing is assembled on the airplane body. The wing needs to be holded down with moderate force to be fitted in the body. Then the wing is fixed with a screw. Finally, another screw is inserted at the rear of the airplane body.

Assemble the main wing

Assembling the main wing to the airplane body is not simple pick-and-place but the robot needs to push the wing into the body. If such a task is found in the database, it can be used as it is. Here we adopt an approach of putting the main wing once on the body and performing the task of holding down with appropriate force from above, since the main wing can be placed on the body in a statically stable manner. Since the airplane body moves easily, it needs to be fixed in some way. The use of another jig is possible, but we fix it with the other hand of the robot since the robot has two hands. The work procedure on our software to make this workflow is summarized as follows.

1. Create a new workflow and add initial and final states.
2. Search "pick-and-place" and add it to the workflow.
3. Replace the 3D models connected to the pick-and-place with the main wing and the airplane body respectively (Fig. 9). This is performed by finding the desired 3D model from the model masters. Then, create a link between the wing frame of the airplane body model node and the place base frame of the pick-and-place task, since the placing motion needs to be executed with respect to the wing frame.
4. Next, search "press" and add the result to the workflow. After that, change the associated objects in the same way.
5. The pressing force or the control parameter is changed if necessary. When the same assembly data exists, the value of the parameter may be used without change. If there is no such data, an appropriate value is set beforehand and some adjustment is carried out later in the real environment. If there is data dealing with similar parts, the recorded data is expected to give a good initial value at the time of adjustment even if it is not optimal.
6. Continue to add motions to hold and release the airplane body with the other hand before and after pick-and-place and press operations. Adjust the holding

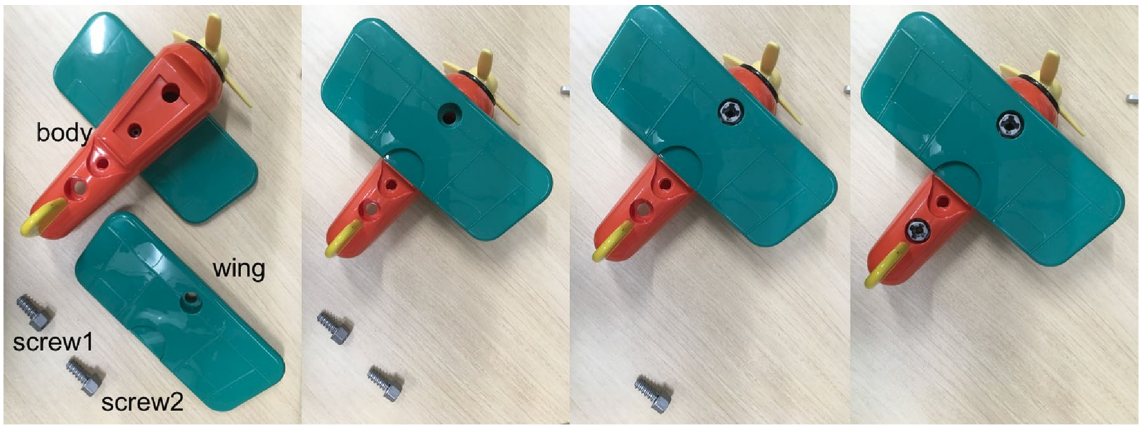


Fig. 8 A part of YCB airplane assembling

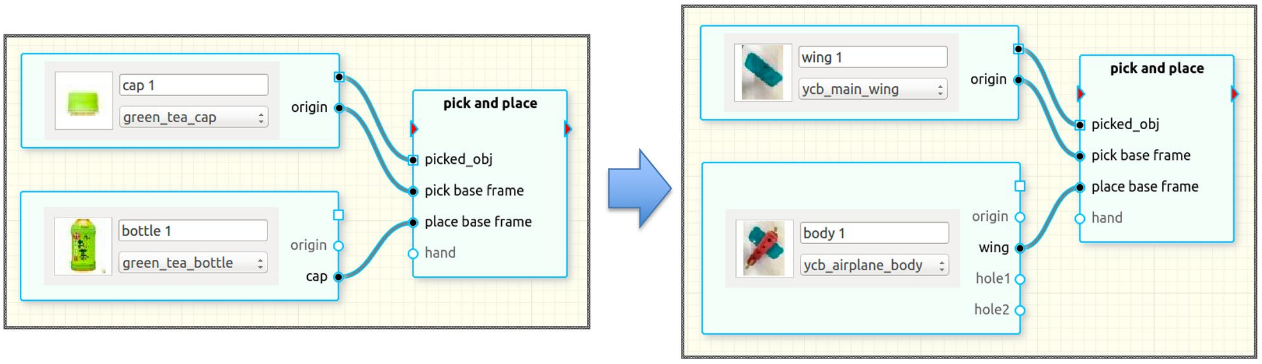


Fig. 9 Add pick-and-place and replace 3d models

position, gripper opening/closing width for the holding. Implementation of the main wing assembly has been completed (Fig. 10).

Since the force value for the pressing is not shared with other tasks, it is not bound to a flow parameter. Likewise, the grasping and hold positions, which are the relative position from the origin of the wind, are not necessarily bound to flow parameters. In the following figures, the selection of the hands has been completed, and the flow parameter for hand selection and the related links are set invisible.

Adding recognition

We assume that the supply position of the main wing is roughly given, but it is not exactly determined. This is a common problem setting for assembly tasks. To deal with this situation, we add a recognition task. The procedure is shown below.

1. Search for recognition task for the wing and add it to the workflow. Keywords such as "airplane" and "recognition" are expected to hit.
2. Recognition is also defined in the same framework as motion patterns. Recognition outputs to a speci-

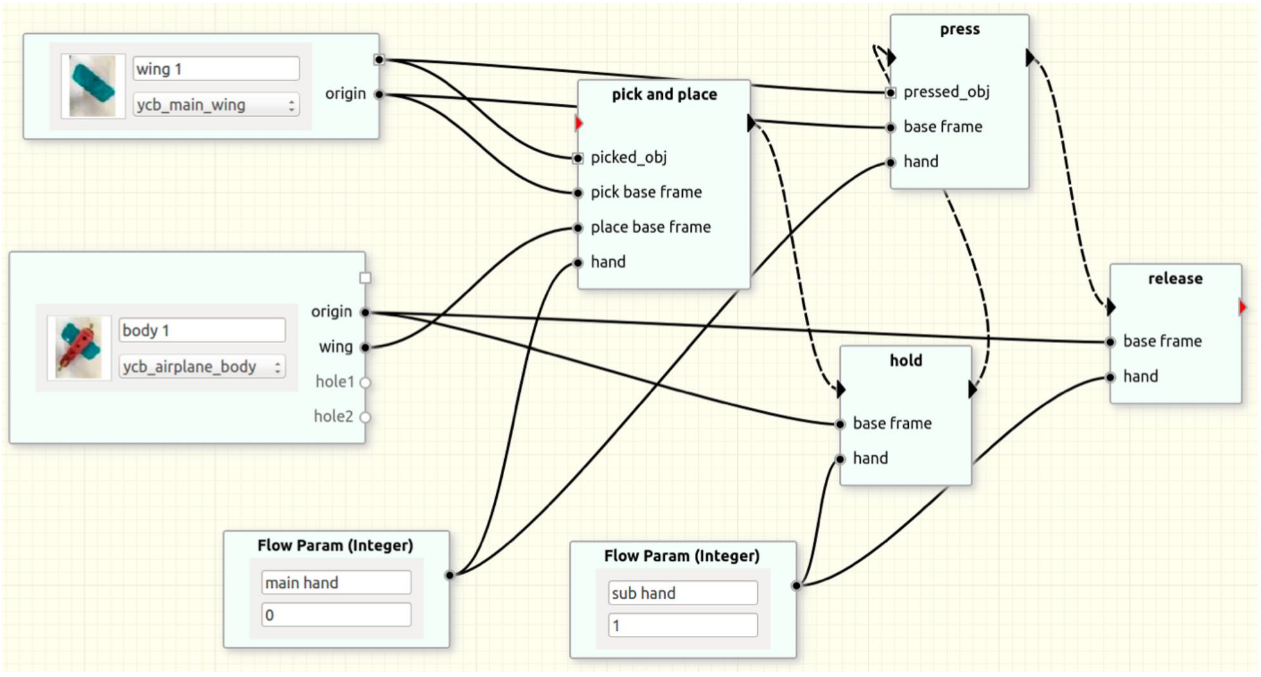


Fig. 10 Assemble main wing

fied task variable. By adding a flow parameter to the workflow and binding it to the task variable, the position and orientation of the recognized object are output to the flow parameter. Bind the flow parameter to a task parameter of the pick-and-place task. As a result, the pick-and-place motion is executed based on the recognized position (Fig. 11).

Screw1

The procedure for assembling the first screw (screw 1) is as follows.

1. First, we add the task of pick-and-place to the workflow and replace the picked object with the screw of the YCB airplane by selecting it from the model masters. Screw 1 is inserted into a hole of the main wing, which is already added to the workflow. This not a separate object of the same type. Thus, place base frame of the pick-and-place is connected to the hole1 frame of the existing model. Since the grasp pose of screw1 is defined with respect to the origin of the screw, pick base frame of the task is connected to the origin of the screw1.
2. Next, motion to tighten the screw placed in the hole with the driver is added. This is a task of picking and transporting the driver and applying force and torque to the screw with the driver. This task consists of 5 steps: pick and transport the driver, screwing, transport back, place the driver to the original position.
3. If the screw tightening torque is different from the registered value, add a flow parameter and connect them with the corresponding task parameter and set some new value.
4. Grasping the driver placed in a free posture requires additional recognition or re-grasping of the driver. This makes the work more complicated than necessary. Therefore, we introduce a tool station for easy grasping of the driver. The 3D model of the tool station is added from the model masters and the hole position of it is connected to the last place task.
5. The screw1 is also supplied using another slot of the tool station so that it can be easily picked. After adding the sequence for screw1, we get the workflow in Fig. 12.

Screw2

Most of the procedure to add the second screw (screw 2) is the same as screw 1. The difference is that the screws associated with the pick and screwing tasks are screw 2 instead of screw 1. Screw 1 and screw 2 are the same type of screws. They are generated from the same model master. However since they are different entities, they are not shared by pick tasks. Since the way of grasping screws and the tightening torque are expected to be the same, the parameters representing them are shared. This prevents from forgetting to change some of the task parameters.

Figure 13 shows the integrated 3D model environment for this workflow. In the integrated 3D model

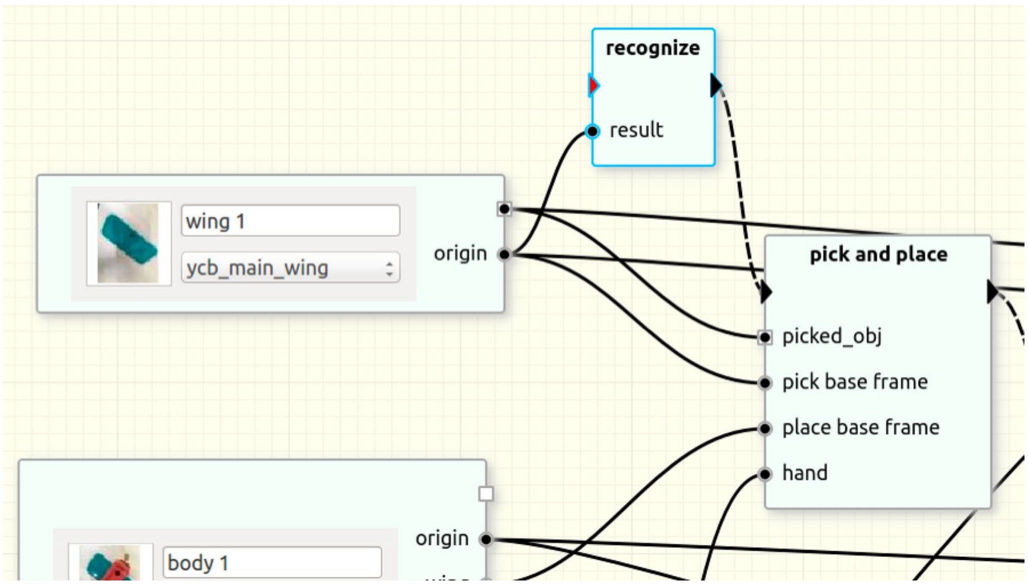


Fig. 11 Add recognition

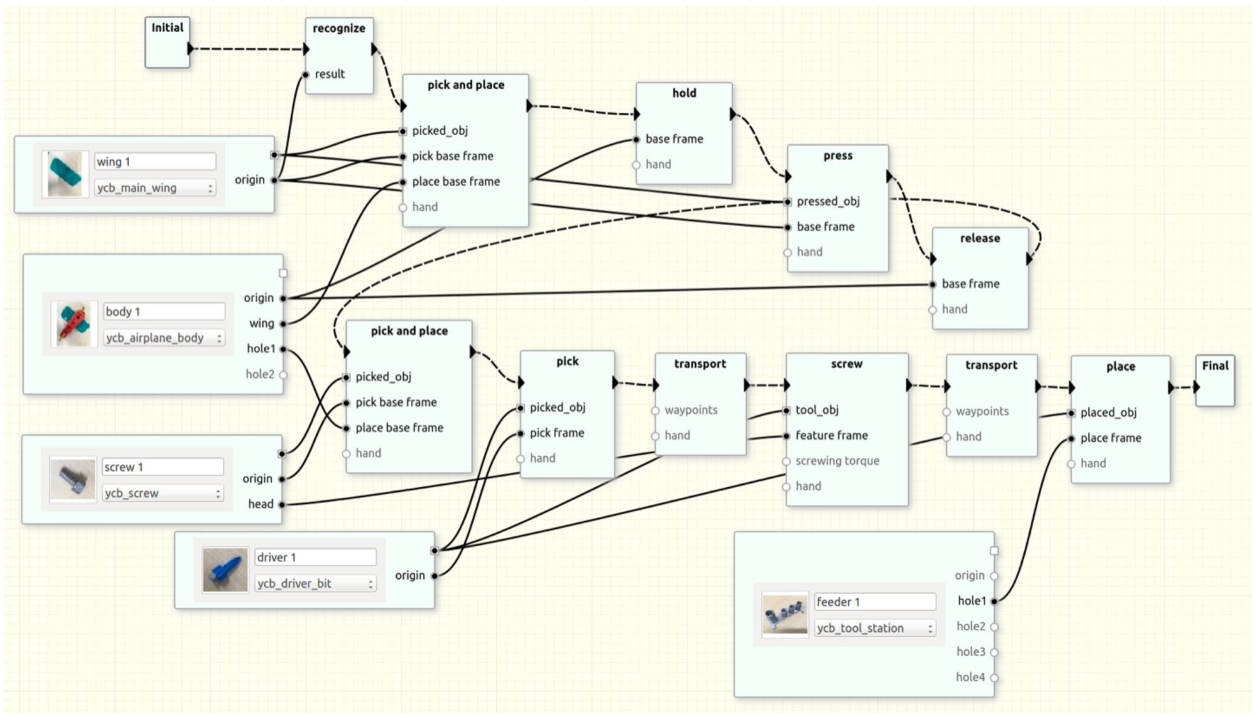


Fig. 12 Add screwing

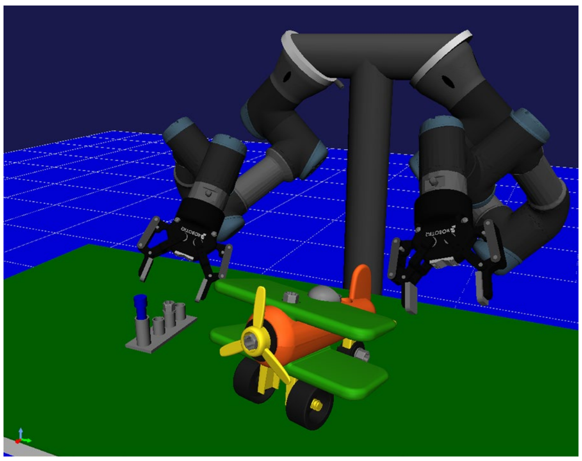


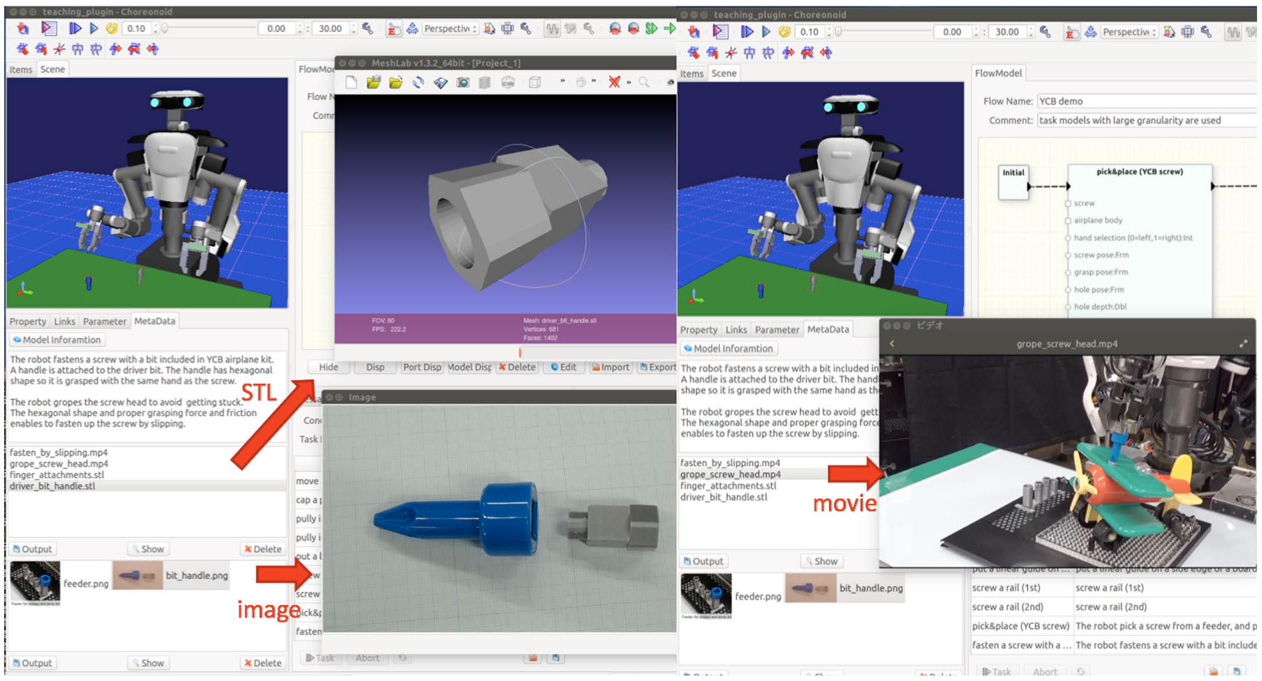
Fig. 13 Integrated 3D models for the workflow

environment, you can change the position where the assembly is performed or the supply location of the tool while confirming the positional relationship of all the related objects. The transfer part of pick-and-place can be designed to perform motion planning internally when designing the task model so as to generate collision-free motion in the integrated 3D model environment.

Figure 14 shows a use case of the meta-data. The text describes the content of work, notes when using each motion pattern, adjustable parameters and so on. The driver is attached a handle for stable grasp by a parallel gripper and slippage prevention during screwing. By 3D-printing the STL file and fitting the printed handle as shown in the picture, it is possible to prepare the same tool as the original data. This accelerates the reproduction of the similar environment. The operation of screw tightening includes an exploratory motion to fit the driver tip robustly in the screw thread. To understand such minute motion, real movie is often effective than 3D models. Actually, there is information that can not be reproduced by the simulator, such as the movement of the screw caused by the contact of the tool with the screw at the time of exploration, which is useful information for understanding what kind of environment interaction was supposed to occur when the original data was prepared.

Discussion

As you can see from Fig. 12, the workflow tends to be complicated. In this figure, the choice of robot hands used for individual tasks is not displayed, but the figure is already complex. Making the diagrams easy to see is one of the important future works. The complexity of control flow and data flow is a common problem in visual languages. We would like to discuss possible approaches with reference to modeling software for SysML [21].


Model Inforamtion

The robot fastens a screw with a bit included in YCB airplane kit.
A handle is attached to the driver bit. The handle has hexagonal shape so it is grasped with the same hand as the screw.

The robot gropes the screw head to avoid getting stuck.
The hexagonal shape and proper grasping force and friction enables to fasten up the screw by slipping.

fasten_by_slipping.mp4
grope_screw_head.mp4
finger_attachments.stl
driver_bit_handle.stl

Flow Name: YCB demo
Comment: task models with large granularity are used

put a linear guide on ...	put a linear guide on a side edge of a board
screw a rail (1st)	screw a rail (1st)
screw a rail (2nd)	screw a rail (2nd)
pick&place (YCB screw)	The robot pick a screw from a feeder, and p
fasten a screw with a ...	The robot fastens a screw with a bit include

Fig. 14 An example of meta-data usage

Firstly, since tasks are compatible with hierarchy ("Related work" section), by defining a partial sequence of tasks as a separate state machine, the entire sequence is expressed with a higher degree of abstraction. Figure 15 represents the overall workflow after hierarchization. Figure 16 shows the sub state machine corresponding to "assemble screw 1". In the case where a large number of local nodes are connected to one parameter, this approach is effective to localize the links in the sub state machine. An example is the parameterization of the hand used for a sequence of tasks.

Figure 15 shows only control flow and does not display data flow. A typical use case of data flow is to see which tasks share a model or parameter and which parameters and models are used by a task. Therefore, we separate the view of data flow from the view of control flow and further suppress the complexity on the data flow by using a view restricted to the data flow connected to selected

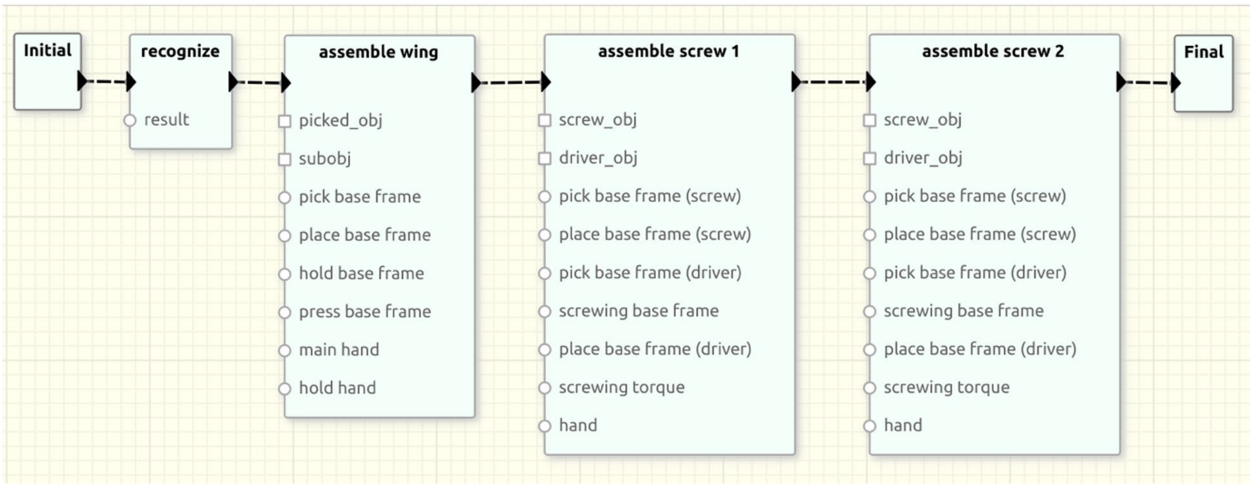


Fig. 15 The whole workflow

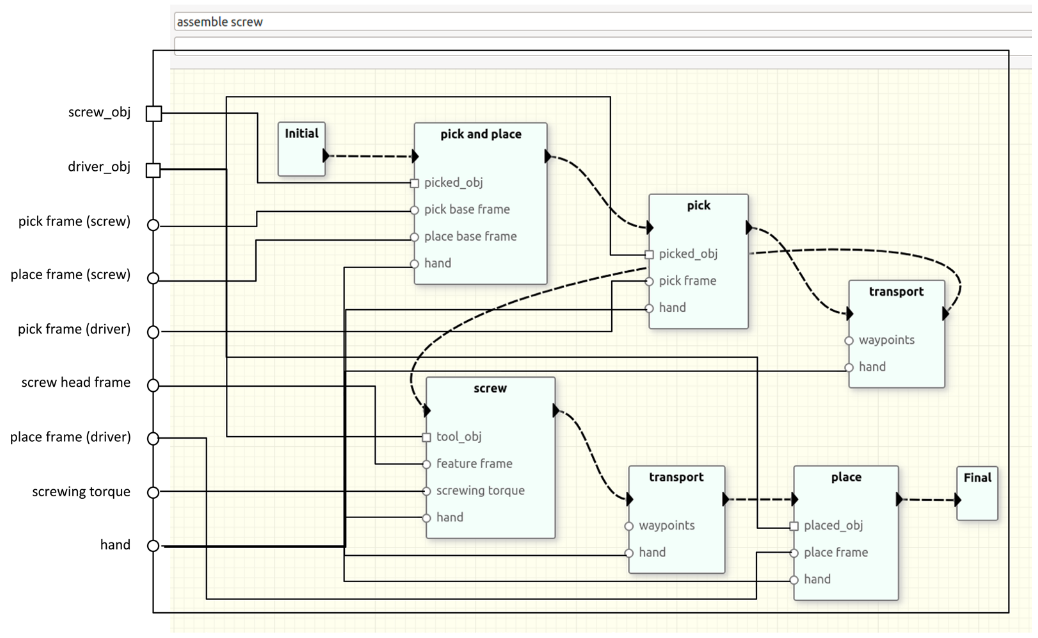


Fig. 16 Sub state machine for assembling a screw

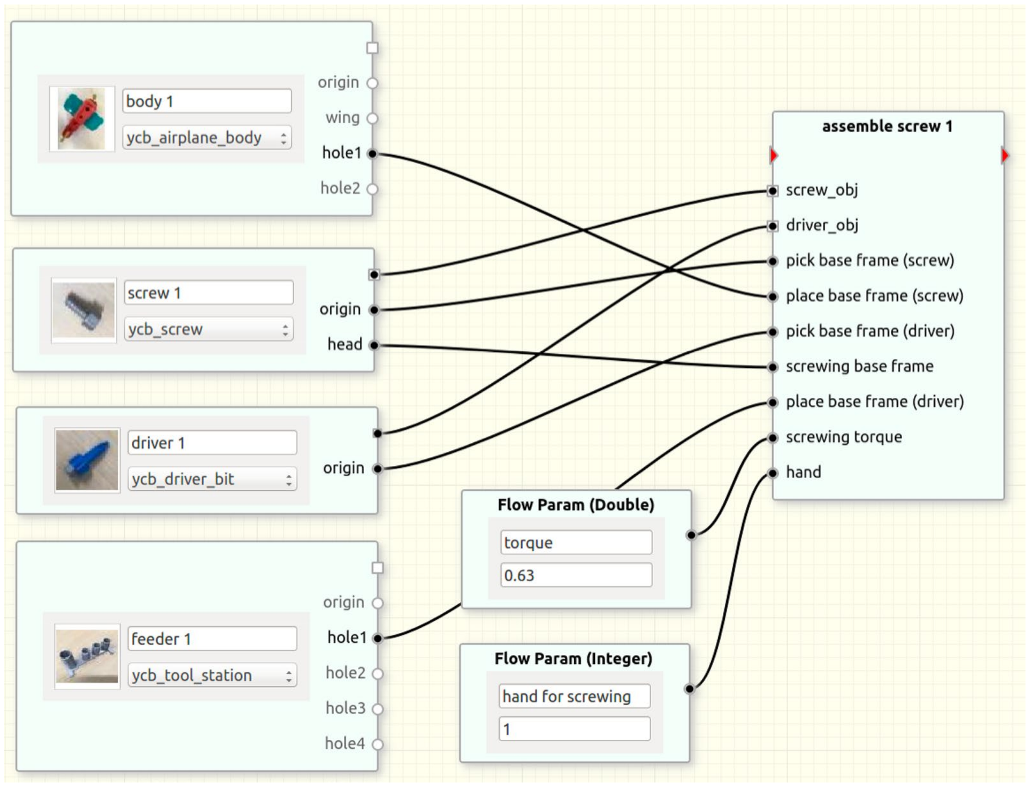


Fig. 17 Dataflow view for a selected task

flow parameter nodes, 3D model nodes, or task nodes. Figure 17 is a view showing the data flow connected to "assemble screw 1" task in Fig. 15.

Conclusion

This paper presented the design of new teaching software based on the reuse of previous teaching data. In this teaching software, a combination of parameters adjustable by users, motion pattern, 3D models as an environment expression and meta-data used for search keys and content understanding by users is the unit of data reuse. A mechanism for systematically building longer motion sequences by combining these data was also explained. This mechanism enables not only to simply arrange motions in line but also to replace 3D models, adjust control parameters over task models and manage the sharing relationship of the parameters and the 3D models when the data are reused in a workflow.

By using this software, it is possible to reuse motion patterns adjusted in the real environment, while adjusting the overall layout of the working environment with integrated parameters and 3D models. Effective support by kinematic motion planning can be expected with the integrated 3D models in the simulator. After that, the implementation as a Choreonoid plug-in are mentioned.

We also presented how each function of the software can be utilized, taking a part of YCB airplane assembling as an example, and showed that a comparatively complex, practical assembly workflow can be implemented using the proposed software. As is discussed in "Discussion" section, introducing mechanisms to suppress the complexity of diagrams is one of important future works.

Authors' contributions

RH designed and implemented the software tool. KH, IH and NA discussed the specification of the software. All authors read and approved the final manuscript.

Author details

[1] Robot Innovation Research Center, National Institute of Advanced Industrial Science and Technology, 1-1-1, Umezono, Tsukuba, Japan. [2] Graduate School of Engineering Science, Osaka University, 1-3 Machikaneyama, Toyonaka, Japan.

Acknowledgements

This paper is based on results obtained from a project commissioned by the New Energy and Industrial Technology Development Organization (NEDO).

Competing interests

The authors declare that they have no competing interests.

Publisher's Note

Received: 2 April 2018 Accepted: 30 August 2018
Published online: 06 September 2018

References

1. Wan W, Harada K (2016) Integrated assembly and motion planning using regrasp graphs. Robot Biomim 3(1):18. https://doi.org/10.1186/s40638-016-0050-2
2. Ramirez-Alpizar IG, Harada K, Yoshida E (2014) Motion planning for dual-arm assembly of ring-shaped elastic objects. In: 2014 IEEE-RAS international conference on humanoid robots. pp 594–600. https://doi.org/10.1109/HUMANOIDS.2014.7041423
3. Pinto L, Gupta A (2016) Supersizing self-supervision: learning to grasp from 50k tries and 700 robot hours. In: 2016 IEEE international conference on robotics and automation (ICRA). pp 3406–3413. https://doi.org/10.1109/ICRA.2016.7487517
4. Levine S, Wagener N, Abbeel P (2015) Learning contact-rich manipulation skills with guided policy search. In: 2015 IEEE international conference on robotics and automation (ICRA). pp 156–163. https://doi.org/10.1109/ICRA.2015.7138994
5. KUKA LBR Iiwa. https://www.kuka.com/en-de/products/robot-systems/. Accessed 31 Mar 2018
6. Rethink Robotics Sawyer. http://www.rethinkrobotics.com/sawyer/. Accessed 31 Mar 2018
7. Universal Robots UR5. https://www.universal-robots.com/products/ur5-robot/. Accessed 31 Mar 2018
8. Nakaoka S (2012) Choreonoid: extensible virtual robot environment built on an integrated gui framework. In: IEEE/SICE international symposium on system integration (SII). pp 79–85. https://doi.org/10.1109/SII.2012.6427350
9. Intera 5.0. http://www.rethinkrobotics.com/intera/. Accessed 31 Mar 2018
10. Ogawa M, Honda K, Sato Y, Kudoh S, Oishi T, Ikeuchi K (2015) Motion generation of the humanoid robot for teleoperation by task model. In: 2015 24th IEEE international symposium on robot and human interactive communication (RO-MAN). pp 71–76. https://doi.org/10.1109/ROMAN.2015.7333619
11. Huckaby JO (2014) Knowledge transfer in robot manipulation tasks. Ph.D. thesis, Georgia Institute of Technology
12. Szynkiewicz W (2012) Skill-based bimanual manipulation planning. J Telecommun Inf Technol 2012(4):54–62
13. Miura J, Iwase K, Shirai Y (2005) Interactive teaching of a mobile robot. In: Proceedings of the IEEE international conference on robotics and automation. pp 3378–3383. https://doi.org/10.1109/ROBOT.2005.1570632
14. Wakita Y, Nagata K, Ogasawara T (2007) Picking instruction with task model for a robot in a daily life environment. In: The 16th IEEE international symposium on robot and human interactive communication RO-MAN. pp 576–581. https://doi.org/10.1109/ROMAN.2007.4415151
15. Thomas U, Hirzinger G, Rumpe B, Schulze C, Wortmann A (2013) A new skill based robot programming language using UML/P Statecharts. In: 2013 IEEE international conference on robotics and automation (ICRA). pp 461–466
16. Hanai R, Suzuki H, Nakabo Y, Harada K (2016) Modeling development process of skill-based system for human-like manipulation robot. Adv Robot 30(10):676–690. https://doi.org/10.1080/01691864.2016.1156575
17. Nagai T, Aramaki S, Nagasawa I (2007) Representation and programming for a robotic assembly task using an assembly structure. In: 7th IEEE international conference on computer and information technology (CIT 2007). pp 909–914
18. Evernote. https://evernote.com/. Accessed 28 Feb 2017
19. Allen G, Owens M (2010) The definitive guide to SQLite, 2nd edn. Apress, Berkely
20. Calli B, Singh A, Walsman A, Srinivasa S, Abbeel P, Dollar AM (2015) The ycb object and model set: towards common benchmarks for manipulation research. In: 2015 international conference on advanced robotics (ICAR). pp 510–517. https://doi.org/10.1109/ICAR.2015.7251504
21. Friedenthal S, Moore A, Steiner R (2011) Practical guide to SysML. The systems modeling language, 2nd edn. The MK/OMG Press, Needham