Design and Prototype of Reusable Software Library for Controlling Space Robots

Hiroaki Hattori*, Masaaki Hatsuta*, Mitsushige Oda**, Hiroshi Ueno**, Takeshi Nishimaki**

* Mitsubishi Space Software Co., Ltd
1-11-3 Sengen Tsukuba, Ibaraki, 305-0047, Japan
hhattori@tkb.mss.co.jp, max@tkb.mss.co.jp
** National Space Development Agency of Japan
2-1-1 Sengen, Tsukuba, Ibaraki, 305-8505, Japan
oda.mitsushige@nasda.go.jp, ueno.hiroshi@nasda.go.jp, nishimaki.takeshi@nasda.go.jp

Keywords Space Robotics, Object-orientated, Control Software, Testbed

Abstract

While the role on space robot becomes more important (e.g. construction of LSS), it is difficult to develop reusable software to control space robot with conventional development approach. In this paper we propose a design pattern of reusable software library for controlling space robots. The features of this design pattern are as follows.

- Architecture based on object-oriented method.
- Multithreading.
- Macro Commands which realize complex operation with simple instruction.

Additionally, we have developed prototype software library for controlling space robots based on this design, and confirmed validity and reusability of the design.

1. Introduction

As the International Space Station is being constructed on orbit by using robotic arms, future large-scaled space structures (LSS) will require fairly amount of robotic assembly and maintenance operation. Therefore, space robotics will play a more important role to provide services of assembling structures and repairing components on LSS. Moreover, it is expected to be applied to the various field of space development such as space utilization or planetary exploration.

Robot tasks are versatile and often complicated. For example, in the case of autonomous or semi-autonomous robotic assembly of LSS, it is expected that sequences of operations are quite complicated, because each task may include unique operation even if on repeated tasks. If conventional software development approaches are applied to such mission, the software becomes one package of complex system and it will become difficult to verify reliability and quality of software system. Besides, in case we try to reuse the software for another mission, it may require more time to rebuild the system.

To solve such problems, it is important to (1) modularize software, (2) describe complex operation flow as a combination of simple operation flows, and (3) combine necessary modules for designated task efficiently. To accomplish these, we propose a design pattern with the feature of ‘Object-orientated Method’ (corresponds to (1)), ‘Multithreading’ (corresponds to (2)), and ‘Instruction with Macro Command’ (corresponds to (3)). This paper describes concept and architecture of the software design of reusable software library for controlling space robots, and prototype of designed software to control ground-based assembly operation testbed [1].

2. Design of software library for controlling space robots
The details of the design pattern of reusable software library for controlling space robots we propose, are described in the following sections.

2.1. Architecture based on object-oriented method

By modularizing a software program, it is possible to add or modify a module, leading to efficient modification of the software program. We introduce object-oriented method as means to modularize a software program.

It is popular to abstract real-world thing as a class object (referred to as ‘object’ herein) and construct class structure, in case we design software architecture based on object-oriented method.

‘Robot’ is abstracted as object, at first, by applying this method. The objective of ‘Robot’ object is to maintain the information about the robot (position, attitude, etc.) and to carry out operation command. Control routines supplied with the robot is wrapped in this class. In general, a robot consists of several components such as arm, end effector, camera, etc. Each component is defined as a subclass of ‘Robot’ class, since they have different way of operation. We can use inheritance association for components which have similar configuration (e.g. arms with different DOF).

Secondary, ‘Target’ which robots transport or observe, is abstracted. This object maintains the information about the target (position, attitude, etc.). In case several robots are at one target, there is a possibility that position information of the target may not be consistent with real position since other robots may have moved the target. Thereat, we add reliability value to ‘Target’ object. The reliability value is high when the robot is holding the target and there is no other robots nearby. On the contrary, it is low when the target is far from the robot. If reliability value of necessary information is low, the robot collects the information from other robots around and adopts most reliable one.

Above two objects are abstracted from real things, other various functions concerning robot control are also necessary to be defined as object. Those are the functions which observe robot’s operation and evaluate it on some criterion. For instance, the function which evaluates achievement of operation or checks whether joint angle is within the allowable range is indispensable. Each observation function should be an independent object, for the reason that each mission needs different functions to observe. That is, these objects work independently for one purpose. However, some observation functions may produce contradictory results owing to the difference in criterion of each observation object. Consequently, we introduce an object to produce final decision based on all observation objects’ results (referred to as ‘Comprehensive Decision’ object herein).

‘Comprehensive Decision’ object has a priority for each observation result, producing final decision in accordance with the priority, and provides foregoing ‘Robot’ object with an operation command. We define the object, which consists of ‘Comprehensive Decision’ object and observation objects, as ‘Scheduler’ object.

All foregoing objects are aggregated to ‘Control’ object. ‘Control’ object activates robots by using operation command execution service of ‘Robot’ object, and controls a process of the sequence (i.e. start, stop, pause, resume of sequence) by using observation and decision services of ‘Scheduler’ object.

As for another necessary function for software to control robots, there is a communication function between robots. This communication function consists of the function which sends operation command or request for information to other robots (= Client function), and the function which receives and processes the request from other robots (= Server function). The client function is realized by using ‘Communication’ object in ‘Control’ object when needed. The server function , however, is an independent object which doesn’t belong to ‘Control’ function, since it should reply to other robots in any case, even if robot has control system failure.
Lastly, we discuss the object which gives operator’s request to ‘Control’ object. This object should have an interface that operator can designate robot operation easily. We call this object ‘Macro Command’. The detail of ‘Macro Command’ is described in Sec.2.3.

What we described above is represented in figure.2-1. This is the architecture of software library for controlling space robot we propose.

2.2. Multithread and communication between threads

A software program with simple processing flows is reusable and easy to verify its reliability. In order to develop such software, we define the simultaneous processes as multithread. Hereby, the description of operation by multiple objects becomes simple.

Firstly, we design robot control part in ‘Robot’ object (function which activates robot described in Sec.2.1) as an independent thread with respect to each component such as arm, end effector, and so on. Similarly, observation object in ‘Scheduler’ object is designed as an independent thread with respect to each purpose.

Necessary robot components and observation items are different between tasks, therefore only necessary objects are created in the sequence. Incidentally, we don’t allocate each thread to time slot directly and leave it to OS (real-time OS). Owing to this way, all we have to do is to set the execution cycle of each thread.

In case one task is performed by multiple threads, these threads need to communicate each other. For this purpose we add ‘Information provide service’ to each object. The system of communication by ‘Information provide service’ is explained below (See Figure.2-2).

- Each thread has private data area, and writes its own information to the area periodically.
- If necessary, each thread obtains the position data, result of observation, or operation command of other threads using ‘Information provide service’, which their objects offer (this service is executed in requestor’s thread).
- Each thread can access its own private data area by priority over other threads in order to keep the cycle.
In case other robot’s information is necessary, as described in Sec.2.1, each thread requests server thread of other robot to provide its information using ‘Communication’ object. Server thread accepts the request from other robot, collects necessary information using ‘Information provide service’, and provides the requestor with the information.

2.3. Instruction with macro command

In case we combine the software modules corresponding to the task, it is convenient to define frequently-used combination of the modules as a set (macro command). This leads to increase of reusability of the software program. Hence, we introduce user interface with macro command which realizes complicated operation with simple instruction. Two key points in the making macro command are described below.

(1) Rough instruction of task.

When robot arm captures some target at a distance, the following operations are probably necessary.

1) Approach the target.
2) Slow down near the target.
3) Control the hand to keep relative position and attitude to the target near the capturing point for a while.
4) Capture the target with the hand.

When operator wants the robot to capture the target, it is not efficient and not reusable to instruct consciously on every detail, every time. In solving this problem, ‘CAPTURE’ macro command, which includes all operation above, should be created. In this macro command, operator can easily alter hand velocity or grasp timing by modifying macro command parameters.

(2) Using not specific number but abstracted name.

When operator wants the robot to move to ‘Workplace A’, if he instructs the place with coordinate value such as ‘(x, y, z) = (0, 5, 10)’, he must instruct again accordingly whenever the position of ‘Workplace A’ is changed. Similarly, in case that operator wants the robot to carry the target ‘slowly so that robot can avoid collision’ or ‘rapidly so that operation finishes sooner’, the way he designates velocity of the robot directly is obscure.

For that reason, we should use not number directly but abstracted name as macro command.
parameter, if possible. As for above examples, we should use 'slowly' or 'rapidly' to designate movement speed of the robot.

These abstracted names are translated into definite number when macro command is interpreted.

If there is no information concerning 'Workplace A' in the robot, it inquires other robots. In case the robot receives multiple replies, it adopts the information with the highest reliability parameter.

3. Prototype software for controlling ground-based assembly operation testbed

We took the ground-based assembly operation testbed as an example, which is for testing on-orbit experimental technologies by NASDA (National Space Development Agency of Japan), and developed prototype libraries for controlling space robots based on the design mentioned in Sec.2. The development environment is as follows.

- Development Tool: Visual C++ Ver.6.0
- Development Language: C++
- OS: Windows 95, Windows NT

At first, we developed prototype software for the experiment in capturing satellite using visual information. After that, we modified the software program for the experiment in handling flexible structure using two robot arms. As for both experiments, our control software worked successfully and the desired objective of the experiments were achieved. Additionally, in modifying the control software, the architecture of the software was not altered but only slight modification was necessary. The details are described below.

3.1. Experiment in capturing satellite using visual information

3.1.1. Experiment overview

The objective of this experiment is to capture the simulated satellite moving with low constant rate by the three-finger hand at the end of arm. The visual information of hand camera is used while the hand approaches capturing point of the satellite. The capturing point is the flange part of the satellite, and after grasping, compliance control using force-torque sensor is performed so that the satellite may be stabilized.

Hardware configuration of this experiment is shown in Figure.3-1(a).

3.1.2. Class structure

Figure.3-2 shows class diagram of control software library we developed for this experiment.

We created the following classes as subclasses of ‘Robot’ class. Summaries of them are mentioned below.

(1) RobotArm
Maintains the information about seven-degree-of-freedom robot arm and executes operation command.

(2) Camera
Imports the image data of camera and processes it. Note that this class is a virtual base class and ‘ThreePointMarker’ class, which is a subclass of ‘Camera’ class, has an actual image processing part.

(3) RobotEE
Maintains the information about three-finger hand attached to the end of arm, and executes operation command.

(4) FTSensor
Maintain the information about force-torque sensor attached between three-finger hand and arm, and performs compliance control.

Additionally, we created the following classes as component classes of ‘Scheduler’ class.

(1) JudgeTargetAchievement
Evaluates achievement of each robot component’s operation.

(2) AcceptKeyboardInput
Checks operator’s interruption command (pause/resume/terminate) by keyboard input.

(3) JudgeStart
Determine whether or not to start
operation (includes only a function which generates a command for closing hand after arm keeps the position and attitude for capturing the satellite).

(4) JudgeOverall

Obtains the results above three objects estimate, and generates operation commands for each robot component.

What is more, ‘Server’ class, which has a service to send visual information to the arm/hand control computer, is created. Finally, we created ‘Macro Command’ class and added necessary macro commands for this experiment (See Sec.3.1.3). In this prototype, We omitted ‘Target’ class which manages target object.

3.1.3. Macro command

Main macro commands we created for this experiment, are as follows.

(1) GUIDE_WITH_CAMERA

Manipulates the arm by visual information (relative distance and attitude between satellite and hand) from the image processing computer, so that the hand may approach capture point (flange of satellite).

User-configurable parameter is name of the robot from which the control computer obtains visual information.

(2) MOVE_FINGER

Manipulates the fingers of hand. In case of grasp, string ‘CLOSE’ is used as macro command parameter instead of joint angle of finger. Similarly, in case of spread, string ‘OPEN’ is used.

(3) CAPTURE

Manipulates the arm so that the hand may approach and capture the target (Combination of macro command (1) and (2) above).

User-configurable parameters are name of the robot from which the control computer obtains visual information, approach speed, and duration of keeping the position and attitude for the target before capturing.

(4) PROC_IMAGE

Imports the image data of camera, and performs designated image processing.

User-configurable parameters are camera ID and type of image processing.

3.2. Experiment in handling flexible structure

3.2.1. Experiment overview

The objective of this experiment is to grasp the flexible structure by the three-finger hand at the end of arm, and to move it to the attachment. At this time, both grip point of the flexible structure and mobile base of the arm are vibrating. The arm/hand control computer estimates relative motion between arm and flexible structure using visual information from the hand camera and another arm’s camera, and makes a plan to approach and grasp the flexible structure. As a result, if possible, the arm executes the plan and moves the flexible structure to the attachment along designated path.

Hardware configuration of this experiment is shown in Figure.3-1(b).

3.2.2. Class structure

We modified control software library mentioned in Sec.3.1. for this experiment. ‘TrackGen’ and ‘MultipleThreePointMarker’ class were added. ‘TrackGen’ object has the role to provide ‘Ctrl’ object with path data loaded from user’s file when the arm handles the flexible structure. ‘MultipleThreePointMarker’ class is a subclass of ‘Camera’ class, and processes image data of two three-point markers simultaneously.

‘RobotArm’ and ‘Ctrl’ class were modified.

The algorithm, which makes a plan to approach and grasp the flexible structure, was added to these classes.

And then, we added two macro commands to ‘MacroCommand’ class (See Sec.3.2.3). Basically, other classes hardly needed to be modified.

3.2.3. Macro command

The following two macro commands were
added for this experiment.

(1) APPROACH_FLEXIBLE_STRUCTURE

Obtains visual information from the hand camera and another arm’s camera, and makes a plan to approach and grasp the flexible structure. As a result, if possible, carry out the plan.

User-configurable parameters are name of the robot from which the control computer obtains visual information and approach speed.

(2) HANDLE_FLEXIBLE_STRUCTURE

Manipulates the arm to move the flexible structure captured by the hand along the path data loaded from user’s file.

User-configurable parameters are path data name and movement speed.
4. Conclusions

We proposed the design pattern of software library for controlling space robots which is easy to use and reusable, and developed a prototype of control software library for ground-based assembly operation testbed to validate the design. As a result, we could effectively modify the library for different missions with a little effort. Furthermore, it seems possible to apply it to the complicated mission such as assembly of LSS.

In the future, we intend to expand this prototype of software library for controlling space robots, and apply it to the control of more complicated missions, for instance, cooperative task by several robots. We are also considering the modifying it for real-time OS, supposing employment on embedded system.

5. References