Production Equipment Virtualization Technology by Integrated Development Environment for Factory Automation

SHIMAKAWA Haruna and IWAMURA Shintaro

Rapidly changing market needs have resulted in the demand for shorter launching times for production facilities. In conventional methods where only physical machines were used, production facility launches took longer because of stand-by time and teaching positions and orientations of robots. A virtual launching environment where multiple independent software programs work together has been proposed to resolve this issue, but complex configurations for data coordination and difficulty in time synchronization between software programs are challenges.

To overcome those challenges, we have defined system data that can be used throughout the system for linking the data in each function on an factory automation integrated development environment, and then we have developed Virtual Modules to seamlessly interlink and synchronize the data.

We have launched actual production facilities with our virtualization technology for the purpose of validation. The launching time decreased by 56%, which proves the effectiveness of our virtualization technology.

1. Introduction

Product life cycles are becoming shorter these days because of rapidly changing market needs. The manufacturing industry must launch a production line and start production as quickly as possible to meet such market trends, and it will enhance competitiveness.

In launching production line, debugging of the control program has generally been conducted using the actual equipment[1]. However, such debugging procedures using the actual equipment are generally not efficient because it is necessary to wait until assembly, wiring work of the equipment is completed, and it is also necessary to operate the equipment slowly to check and avoid collisions.

To solve the problems associated with inefficiency in the launch of production line using actual equipment, virtualization technology is used for debugging and teaching of production line without using the actual line. However, to virtualize the entire production line, it is necessary to link multiple simulation software, but setting up them to align and synchronizing times are difficult.

We virtualize a production line with Sysmac Studio, the integrated development environment for factory automation offered by OMRON to solve those challenges. The feature of Sysmac Studio is that it consists of a single software program, which makes automatized data coordination and time synchronization possible. We have defined the system data that can be used throughout the system for coordination of the data in each function and developed virtual modules that can seamlessly coordinate and synchronize such data.

In Section 2 of this paper, details of the problems encountered in production line development using the conventional method are explained. In Section 3, details of the virtualization technology implemented are explained, and in Section 4, validation results of the virtualization technology are discussed. Section 5 provides a summary of this paper and future vision and challenges.

Contact : *SHIMAKAWA Haruna* haruna.shimakawa@omron.com

2. Problems

2.1 Inefficiency in launch of production line using actual equipment only

Fig. 1 shows the conventional development process in the launch of production line.

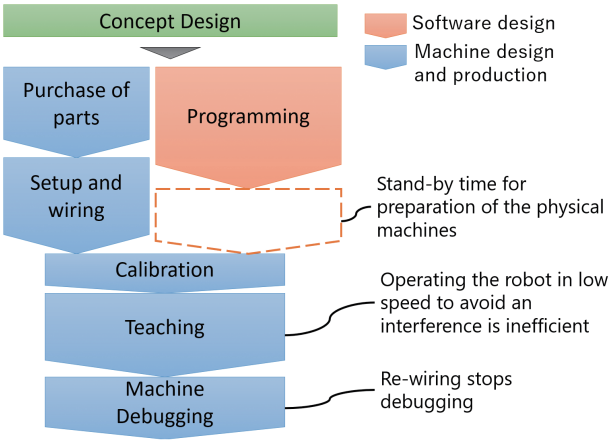

Fig. 1 Conventional launch process of production line

Production lines have been generally developed with the actual equipment. Control programs for robots and peripherals were created to control such a system which Fig. 2 illustrates, and validating their operations required actual equipment to control. As a result, there is often a waiting period after programming until the actual equipment is ready, which reduced the efficiency of the production line launch. Debugging with the actual equipment sometimes requires a change in a mechanism position or wiring. While the position of the mechanism or wiring is being reviewed, debugging stops, which will decrease efficiency. Teaching a robot also takes time since an operator must slowly move the robot to avoid damage caused by colliding with other objects.

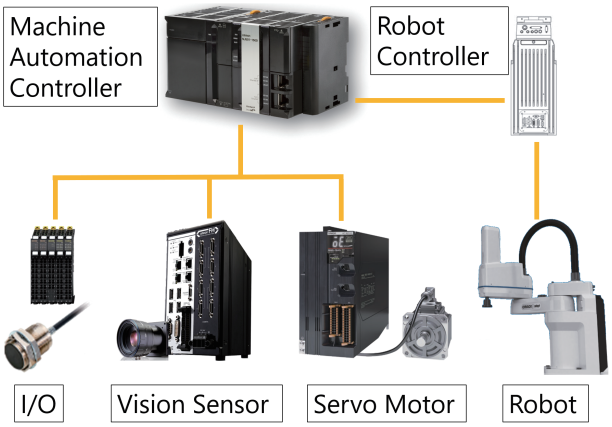

Fig. 2 Production equipment control system typical architecture

2.2 Issues associated with virtualization using multiple software

The technology to virtualize an entire production line eliminates the inefficiency in production line launching described in Subsection 2.1 since virtualizing the whole production line allows us to debug or teach robots in the virtual space while assembling the actual equipment and wiring. Production line development using virtualization technology is more effectively planned than using the actual equipment only.

Virtualization of the entire production line demands virtualizing all equipment in the line, which requires different simulators, respectively. As time synchronization between multiple simulators is difficult, it is also difficult to reproduce the identical operation of the actual equipment as a result[2].

3. Virtualizing an entire production line with single software

We integrate multiple simulators into OMRON's Sysmac Studio, the integrated development environment for factory automation, to virtualize a whole production line. Sysmac Studio can virtualize all elements of the production line: robots, peripherals, I/O devices, vision sensors, and parts assembled in the production line.

Specifically, we have developed the Robot Integrated CPU Units[3], which combines a machine automation controller and robot controller to integrate controls of robots and control devices. Then we have developed the Robot Integrated Simulator to virtually operate a Robot Integrated CPU Unit on a computer to integrate into Sysmac Studio. The single simulator can link and synchronize the I/O data of robots and control devices in real-time.

Next, we have developed modules for virtualizing all elements in a production line (Virtualization Modules). Subsection 3.1 describes solutions for Sysmac Studio's software architecture consists of different virtualization modules. It is also necessary to realize the programming environment for virtualization to define virtual operations. Details of the programming environment for virtualization are explained in Subsection 3.2. Subsection 3.3 and the following sections provide details about virtualization modules defined in Subsection 3.1.

3.1 Software architecture to virtualize an entire production line with single software

Fig. 3 shows the software architecture for production line virtualization with Sysmac Studio. Fig. 4 illustrates the sample production line configuration with a Robot Integrated CPU Unit. The production line consists of the I/O device to control the sensor, vision sensor, servo motor, and robot.

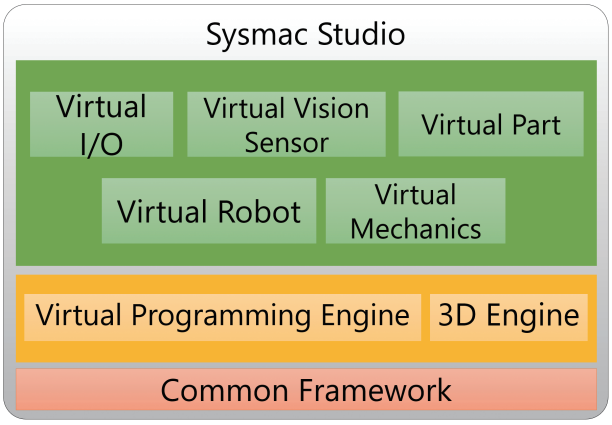

Fig. 3 Software architecture for virtualization with Sysmac Studio

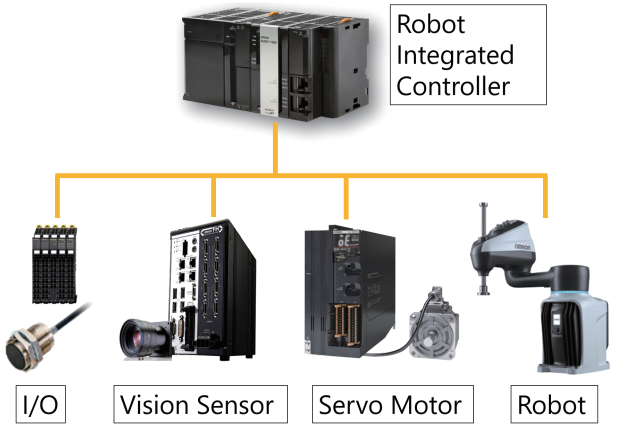

Fig. 4 Example of production line configuration including Robot Integrated CPU Unit

The foundation required to implement all virtualized modules by a single software is Common Framework. Common Framework is a module that provides the common functions and interfaces to the modules and controls lifecycles of the modules from creation to deletion. Virtual Programming Engine executes control and simulation programs for production lines. Virtual Programming Engine includes the simulator of the Robot Integrated CPU Unit and the programming environment for simulations. 3D Engine is the module that manages the shape data of 3D models for an entire production line, performs the 3D visualization function, and controls the models not to collide. Virtual Robot, Virtual Peripheral, Virtual I/O, Virtual Vision Sensor, and Virtual Part are the virtualized elements of a production line on 3D Engine. Virtual Robot and Virtual Peripheral are broken down into movable parts, and kinematics calculation models according to robot type are assigned to the movable parts. Inputs of motor command values obtained from calculations of the control program replicate the operations of Virtual Robot and Virtual Peripherals on 3D Engine.

Virtualizing the entire production line system with a single software requires sharing the data of individual functions so that the system can use the data as the system data for the whole system, and the data in each module must be shared with other modules. However, sharing the module data increases interdependence among modules and complexity and results in low operation performance and high memory usage. To avoid such issues, we manipulate the module data as the system data, shared with the entire system. The system data for virtualization must share the data that identify a 3D model's position and the state variable data storing the 3D model's state. For this purpose, we define an abstract interface common with different position data, e.g., the 3D position data for 3D models and the operational position data of peripherals. We have developed the function to display these data on the 3D Engine in the absolute coordinates automatically to manipulate. Furthermore, this function allows managing the position data as the common data from any module. We have realized the system data to enable data sharing while preventing direct linkage between modules using the abstracted state variables given the unique ID information with the hierarchy structure shared in the whole system. The number of system datasets is usually over 1,000 in a general production line, and accordingly, increasing data volume causes issues like performance degradation and high memory consumption. We have solved the issues by using the ID with the hierarchy structure reduces the search volume to access the data at high speed, compared with the data transmission with the simple parallel data structure. The ID is a relative ID and it decreases memory consumption when a hierarchy of the data increased. These system datasets achieve the virtualization of an entire production line to enable real-time monitoring on a large production facility.

3.2 Realization of simulation programming environment for virtualization of entire production line

For virtualization of the operation of an entire production line, programming the virtual operations of the object (part) that is not directly controlled by an actuator is necessary, since generating conditions and indication position to define virtual operation of a part are not included in the control program. It is also necessary to define virtual operations of the I/O device, such as sensor, since we cannot connect actual I/O devices in the virtual space.

Because the operations in the virtual space must be defined flexibly depending on the state of the control program rather than defined in a standardized and straightforward way, a virtualization-dedicated program written in a specialized language is generally used for definition. Generally, a virtualization program is written in a specific programming

language. However, it will take time to familiarize such a special language[4]. Accordingly, we have developed the script language (Shape Script) for Sysmac Studio, which based on the C#, the all-purpose programming language.

Acquiring the control data of I/O operations in real-time to change position indications of objects such as a part is required while the Shape Script is running. The Shape Script must be executed in the same process where the Shape Script shares the memory space with Sysmac Studio. The process is a unit of program execution in the Windows environment. Including the Shape Script in the same process enables sharing the memory space and exchanging information in real-time. We have built the structure to compile a Shape Script and generate assembly to load it into the same process before execution. However, when the Shape Script is solely loaded into the same process, the shared memory cannot be released if the Shape Script stops. To resolve this issue, we have created a system to separate the memory space in a process.

Shape Script and Sysmac Studio are applications run on the Windows environment, and use the .NET Framework, the common library provided by Microsoft. The .NET Framework is one of the common language runtimes (CLR), the operating environment of the Windows program. A CLR has the "application domain" as the control unit of the execution code. Dividing the application domain allows to divide the memory space. Therefore, we have separated the application domains of the Shape Script and Sysmac Studio as shown in Fig. 5. This makes Sysmac Studio unaffected even when the Shape Script assembly is loaded or unloaded.

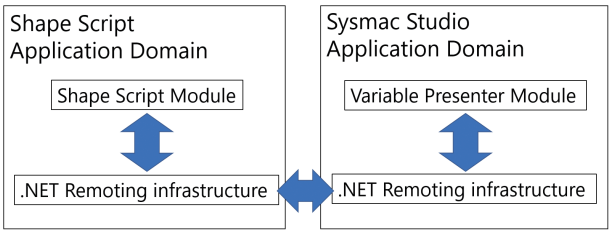

Fig. 5 Communications between Shape Script and Sysmac Studio

Execution conditions for an operation to define will vary in many cases depending on the Robot Integrated CPU Unit state. Such state is stored in a variable. Therefore, a Shape Script needs to obtain variable values from the simulator of the Robot Integrated CPU Unit.

Sysmac Studio has a function to obtain the values of the designated variables from the simulator of a Robot Integrated CPU Unit. Since the Shape Script and Sysmac Studio application domains are divided, Sysmac Studio can acquire variable values through communications between both application domains.

For the communications between the application domains as shown in Fig. 5, we have developed the dedicated module (Variable Presenter Module) that abstracts the state variables obtained in the application domain of Sysmac Studio and shares them among modules. This module realizes inter-module marshaling (data exchange) of the state variables. Also, the variable presenter module achieves marshaling of the system data, making the assembly of the Shape Script the subject of marshaling. In marshaling, the data are tentatively serialized and stored in the memory space, then exchanged between application domains and deserialized before data exchange. Therefore, the data structure affects the performance and memory capacity directly. In order to ensure the performance, we have eliminated the information that can be re-constructed outside the application domain from the marshaling targets and included only the data of positions or states necessary for virtualization to achieve a high-speed state variable marshaling of the Robot Integrated CPU Unit.

3.3 Realization of virtual part for virtualization of entire production line

As explained in Subsection 3.2, virtualization of the part is impossible by executing a simulation of the control program only. We define the state of a virtual part according to the simulation of the control program with a Shape Script describing operations in the virtual space.

Operating the virtual part requires defining the operation conditions a Shape Script. The conditions are determined depending on the Robot Integrated CPU Unit's state. For example, the operation that a robot hand grips the virtual part can be defined by the script describing to detect a gripping signal input to the robot hand.

Secondly, the Shape Script must define the state of the virtual part when the conditions are satisfied. The state to be defined is the visibility or the indication position (coordinates) of a virtual part. When the virtual part is fed to the production line, the state is visible; when the virtual part is removed from the production line, the state is invisible. With respect to the indication position, it should be defined so that the virtual part follows the robot hand when the gripping signal input to the robot hand is detected. To actualize these settings, we generate instances of a virtual part with a Shape Script to toggle the visibility and the indication positions of each instance.

The position information of virtual parts is critical when a problem related to it occurs. For example, if the robot fails to pick up a virtual part from the conveyor, it is necessary to investigate where the virtual part is on the conveyor. We have

achieved tracing the indication position of the virtual part in the virtual production line and showing the tracing result in a graph to identify the problem regarding parts and debug readily. Fig. 6 shows examples of indication position graphs.

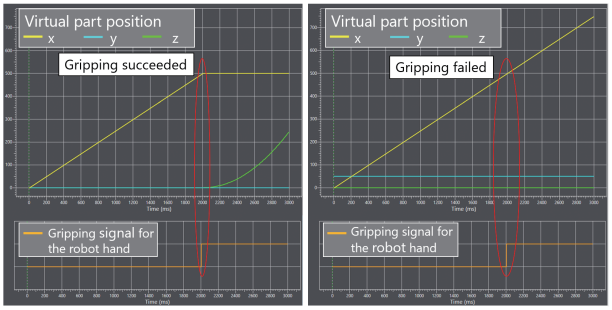

Fig. 6 Tracing results of the virtual part indication position

Fig. 6 shows the changes to the x, y, and z coordinates when a virtual part carried on the conveyor in the x-axis is gripped and picked up by the robot hand in the virtual space. The graph on the left indicates that the virtual part moving in the x-axis stops and starts moving upward in the z-axis immediately after the gripping signal is input to the robot hand. The graph on the right shows that the virtual part does not stop moving in the x-axis at the input of the gripping signal, and the robot hand failed to pick up. Comparing the two graphs suggests the position of the virtual part shifts in the y-axis direction, and an operator should adjust the virtual part position or the pick-up position of the robot hand in the y-axis direction. Tracing the position of the virtual part will allow identifying the point where a problem occurs readily.

3.4 Realization of virtual I/O for virtualization of entire production equipment

It is necessary to define the virtual operation of the I/O since an actual I/O is not connected to the virtualized production line, as mentioned in Subsection 3.2. The sensors to detect parts, such as the photo sensor, and the sensors to detect the actuating position of the mechanism, such as the limit switch on an air cylinder, are commonly used in production lines. Therefore, we have developed the Virtual Part Detection Sensor and Virtual Actuation Position Sensor as virtual I/O devices.

The common requirement for both sensors is to switch I/O variables to True or False, when the condition to detect the virtual part is satisfied or not. As the I/O variables are defined in the Robot Integrated CPU Unit, I/O variables can be set through the Variable Presenter module from the Shape Script.

The I/O variable of the Virtual Part Detection Sensor switches when the virtual part enters or exits the predetermined area. The 3D Engine displays the detection area for the Virtual Part Detection Sensor to define these conditions. We determine the presence or absence of collision of the 3D model of the virtual part to the Virtual Part Detection Sensor's detection area on the 3D Engine as the condition to switch the IO variable. Fig. 7 shows the 3D models of the Virtual Part Detection Sensor and virtual parts displayed on the 3D Engine.

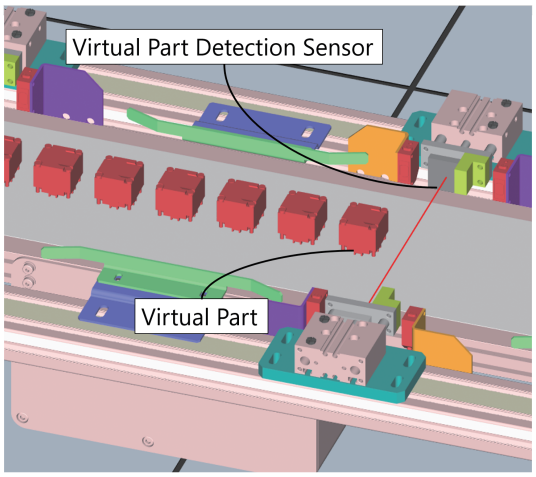

Fig. 7 3D models of the Virtual Part Detection Sensor and virtual parts

The 3D Engine detects an collision between 3D models based on the collision check explained later in Subsection 3.6. A Shape Script can acquire the collision detected by the 3D Engine by referencing the instances of the 3D models, and a designer can create a script to compensate the operation of an actual I/O device only by describing the Shape Script to change the I/O variable when collision between the 3D models occurs.

When the movable part of a peripheral moves to the predetermined position, the I/O variable of the Virtual Actuation Position Detection Sensor toggles. In an air cylinder, the I/O variable corresponding to the piston extension complete signal should switch after a certain period passed since the I/O variable for starting air charge becomes True because the I/O variable switches when the piston extends and retracts.

The Shape Script using the Variable Presenter module can detect changes in the I/O variable. In the same way as the I/O variable, the Shape Script can acquire the time information to measure time lapse since the simulator of the Robot Integrated CPU Unit contains the time information in the variable. As explained above, it is easy to make the Shape Script to switch the I/O variable of the Virtual Actuation Position Detection Sensor after a certain period since the I/O variable for starting air charge becomes True.

Some production lines have 100 or more sensors to detect parts and actuation positions of peripherals, and considerable person-hours are necessary to define virtual operations for all

these sensors in a Shape Script. However, the virtual operation of an I/O is represented with a pattern code, unlike the virtual part. We have automatized defining the virtual operations using a Shape Script with simple settings in the I/O variables corresponding to the respective virtual I/O and shortened the time for defining I/O virtual operations with the Shape Script.

3.5 Realization of virtual vision sensor for virtualization of entire production line

Vision sensors in a production line detect the position of the parts randomly fed onto the conveyor or visually inspect products. We have developed the Virtual Vision Sensor that virtually performs the vision sensor engine on a computer. The virtual part position detection and visual inspection simulations are available by inputting the prepared image data into the Virtual Vision Sensor.

An actual vision sensor requires calibration to adjust the positions of the vision sensor, robot, and peripherals. We have realized automatic calibration by arranging the vision sensor, peripherals, and robot in the virtual space. Additionally, we have virtualized parts using multiple images with the Shape Script. These achievements suggest that automatic generation of virtual parts and replicating position detection of different kinds of virtual parts with images in the same way as in an actual production line are possible. Arranging the objects on the virtual space enables the robot to precisely pick up the virtual parts in tracking the virtual parts on the conveyor. Fig. 8 shows the Virtual Vision Sensor allows position detection and tracking of different kinds of virtual parts. The robot picks up the part in Fig. 8 (a) and places the part in Fig. 8. Displaying the order of the parts gripped by each kind of the virtual parts in 3D realizes the system data part control and visualizes it.

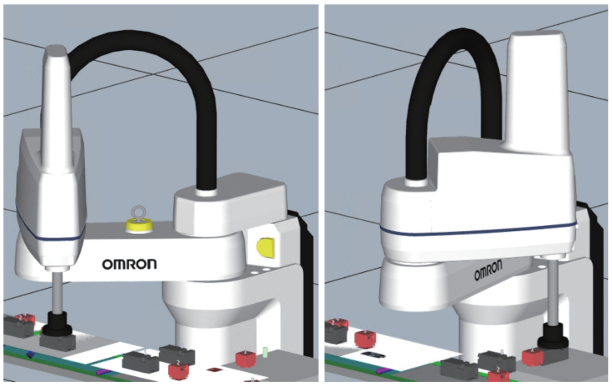

Fig. 8 Part tracking by Virtual Vision Sensor

3.6 Realization of collision check for virtualization of entire production line

One of the most time-consuming operations in the launch of a production line is teaching a robot. An operator creates teaching points by moving the Tool Center Point (TCP) of the robot to positions to be learned and recording. Then the operator makes a robot program connecting the recorded teaching points to generate a robot path.

In teaching with an actual robot, the operator must slowly and carefully move the robot to avoid a collision. On the other hand, teaching in the virtual space can save time since there is no risk of physical damage to devices even when a collision occurs in the virtual space, and the operator can move the robot at a higher speed than the actual robot. Also, preparative offline teaching in the virtual space can reduce actual robot teaching time because the operator does not have to do tasks than adjusting the teaching points.

Generating a robot path that will not cause a collision is vital in offline teaching. When the robot collides with peripherals in the virtual space, the collision can be avoided by modifying the teaching points. Therefore, it is required to detect when and where a collision occurs in the robot path.

For this purpose, we have introduced the Collision Check function into the 3D Engine to indicate a colliding 3D model by color change. As the collision check using 3D models in the 3D space generally requires long computation time and large memory consumption, a box indicating borders, which simplifies a 3D model, is generated internally in many cases. In the 3D simulation of Sysmac Studio, an collision check is implemented by generating border-indicating boxes using convex hull or approximate convex decomposition[5,6]. Fig. 9 shows border-indicating boxes generated by convex hull and approximate convex decomposition.

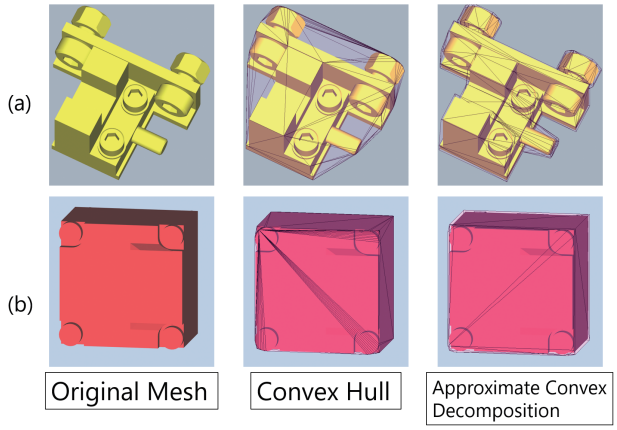

Fig. 9 Border-indicating boxes generated by convex hull and approximate convex decomposition

The characteristics of a 3D model determine the selection between convex hull or approximate convex decomposition. Approximate convex decomposition can represent 3D models with complex shapes as (a) better, but convex hull is suitable for relatively simple shapes like (b) since the calculation cost is low. Depending on the 3D models, users can select a suitable border-indicating box.

Sometimes positions of the virtual peripherals and parts in a production line simulation may differ from the positions in offline teaching, and the difference may make the robot collide with peripherals or parts in the simulation in the virtual space. If such collision is detected, offline teaching is necessary again. Suppose it is impossible to identify when and where the collision occurs during simulation precisely. In that case, it will be impossible to find which teaching point needs adjustment, and the offline re-teaching will demand extra time.

Collision may be left unnoticed if collision notification is given only by color changes in the 3D model during simulation. To prevent missing the collision, a Shape Script can take a log of the information on the timing and location of the collision and output it. As addressed in Subsection 3.4, the Shape Script can obtain a collision between 3D models on the 3D Engine. Therefore, the code for outputting the timestamps and instance names at a collision between 3D models on the 3D Engine to a log enables checking the timing and location of the collision and eliminates the extra work of offline re-teaching.

4. Validation of virtualization of entire production line

To validate a virtualized whole production line, we launched an actual production line. Fig. 10 illustrates the layout drawing of the production line for the validation. In the production line in Fig. 10, the vertical articulated robot Viper 650 assembles parts supplied from the feeding tray.

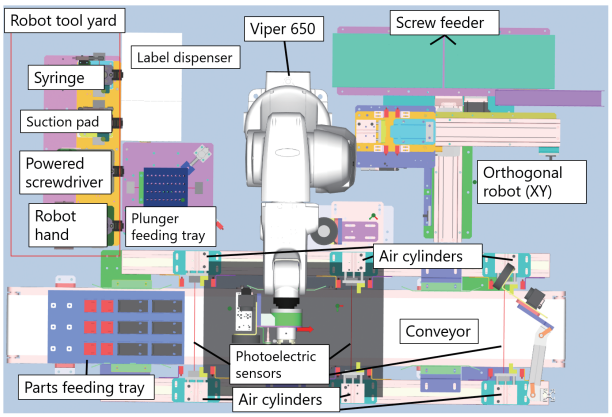

Fig. 10 Layout drawing of the production line used for validation

We prepared the control program, debugged, calibrated the devices, and taught the robot for the assembled and wired production line. Then, we recorded person-days for each process to compare the person-days with and without the virtualization technology. Fig. 11 shows the comparison results.

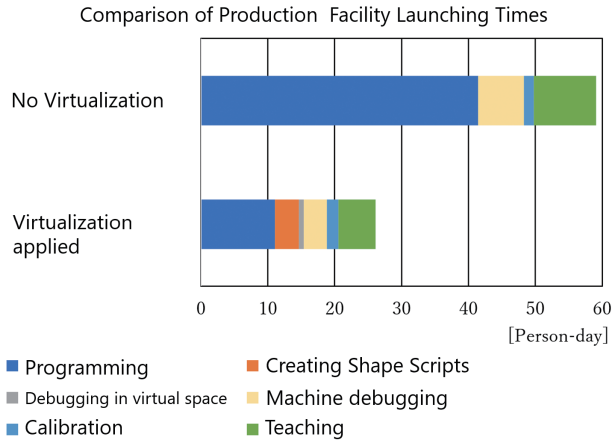

Fig. 11 Validation results of production line virtualization effect

Fig. 11 shows that launching the production line required about 59 person-days without the virtualization technology. On the other hand, launching the production line with the virtualization technology took about 26 person-days, and the person-days for launching were reduced by approximately 56%.

We have the most effective person-day saving effect in the programming process and reduced about 63% person-days: it took about 41 person-days without the virtualization technology but about 15 person-days with the virtualization technology, including person-days for Shape Scripts. In programming without the virtualization technology, each assembly process requires designs and implementations for each individual function to avoid collision. For the state variable used in all processes and the functions covering multiple processes, additional designs and implementations are necessary to merge all processes. In contrast, the validation proves that programming with the virtualization technology eliminates the collision issue, and it is possible to work on the whole production line from the beginning without designs and implementations of individual functions. Substantially, removing additional designs and implementations reduces the programming person-days for the entire production line.

The second most person-days saved process is debugging of actual devices: it took about seven person-days without the virtualization technology, but the debugging with the virtualization technology reduced about four person-days, which accounts for about 43%, including person-days required

for debugging in the virtual space. We debugged in a shorter period on the virtual space than using the actual equipment since we could operate the robot and peripherals faster. Furthermore, we succeeded in reducing the person-days for re-debugging: in the virtual space, only resetting of the conditions is required, but as for the actual equipment, we need to set the robot, peripherals, and parts to the initial conditions. After the debugging on the virtual space, the debugging in the actual equipment requires necessary tasks only: e.g., adjustments on suction and gripping a part. Therefore, we have decreased the person-days for debugging in total.

The validation did not include the time required for assembly, wiring work, and associated waiting time. Therefore, we expect a more substantial reduction in programming and debugging in launching an actual production line.

The third most person-days saved process is robot teaching: the teaching with the virtualization technology took about 5.5 person-days, but the teaching without the virtualization technology took about nine person-days. The validation addresses that the virtualization technology reduced about 39% of person-hours. As Subsection 2.1 describes, the operator must move the robot slowly to avoid damage to the equipment caused by a collision during teaching with the actual equipment, but in offline teaching, the operator can operate the virtual robot faster in the virtual space. Thus, offline teaching and minor adjustments for the actual equipment can save the working time on the actual equipment and reduce the teaching person-days.

5. Conclusion

We aim to reduce person-days for launching a production line and have implemented the production line virtualization technology into the factory automation integrated development environment, Sysmac Studio. Our virtualization technology enables I/O devices, vision sensors, peripherals, and robots to operate in the virtual space. Furthermore, we have built the simulation programming environment that defines the virtual part's operations flexibly. The clearance check for all virtual devices and parts realizes offline teaching.

We have validated the virtualization technology using the actual production line and proved that the virtualization technology could reduce the launching person-days by about 56%.

This study presents that the virtualization technology can reduce satisfactory person-days in programming and debugging. We will discuss the further reduction using a method other than virtualization technology, such as improving a program editor.

The robot teaching process involves most person-days, following programming and debugging. We will develop a path planning technology that allows automatic path generation by only designating the start and end points to reduce person-days in offline teaching.

References

1) T. Miyauchi, D. Kobayashi, and K. Fujita, "Virtual Debugging System for Manufacturing Equipment Control Software Development," (in Japanese), *TOSHIBA Review*, vol. 64, no. 5, pp. 10–13, 2009.
2) Japan Robot Association, *Reference Book of Robot System Integrator's Skills* [1st Edition], Ministry of Economy, Trade and Industry (2018–5–31), (in Japanese), https://www.meti.go.jp/press/2018/05/20180531008/20180531008–1.pdf (accessed on Jan. 4, 2021)
3) OMRON Corporation, "Robot Integration System" (2020–02–09), https://www.fa.omron.co.jp/product/robotics/lineup/integrated controller/ (accessed on Jan. 13, 2021)
4) F. Hosseinpour and H. Hajihosseini, "Importance of Simulation in Manufacturing," *World Academy of Science, Engineering and Technology* 27, 2009, pp. 285–288.
5) K. Mamou, F. Ghorbel, "A simple and efficient approach for 3D mesh approximate convex decomposition," in *16th IEEE Int. Conf. on Image Processing (ICIP)*, 2009, pp. 3501–3504.
6) J.-M. Lien and N. M. Amato, "Approximate convex decomposition of polygons," in *Proc. 20th Annual Symposium on Computational Geometry*, 2004, pp. 17–26.

About the Authors

SHIMAKAWA Haruna

Software Development Dept., Controller Div.
Product Business Division H.Q.
Industrial Automation Company
Specialty:Software Engineering
Affiliated Academic Society:RSJ

IWAMURA Shintaro

Software Development Dept., Controller Div.
Product Business Division H.Q.
Industrial Automation Company
Specialty:Software Engineering
Affiliated Academic Society:RSJ