FA 統合開発環境における生産設備の仮想化技術

島川 はる奈，岩村 慎太郎

近年は市場ニーズの変化が速く、製品ライフサイクルの短命化が進んでいる。このような市場動向に対応するため、より短い期間での生産設備の立ち上げが必要である。従来は実機のみで立ち上げていたため、作業の待ち時間の発生やロボットの作業位置、姿勢を教示するティーチングに時間がかかっていた。これを解決するため、独立した複数のソフトウェアを連携させて仮想立ち上げ環境の実現が提案されてきたが、データ連携のための設定の複雑さと各ソフトウェア間の時間の同期が難しいという課題があった。

これらの課題を解決するために、FA 統合開発環境において各機能でデータを連携するためにシステム全体で使えるシステムデータを定義し、このデータをシームレスに相互連携・同期できる仮想モジュール群を開発した。

実現した仮想化技術の有効性を検証するため、仮想化技術を使用して生産設備の立ち上げを行った。その結果、立ち上げ時間を 56%削減でき、仮想化技術の有効性を確認できた。

Production Equipment Virtualization Technology by Integrated Development Environment for Factory Automation

SHIMAKAWA Haruna and IWAMURA Shintaro

The rapidly changing market needs have demanded shorter launching time for production facilities. In conventional methods where only physical machines used, production facility launching took longer time due to stand-by time and teaching positions and poses to robots. A virtual launching environment where multiple independent software work together has been proposed to solve this issue, but complex configurations for data coordination and difficulty in time synchronization between software are challenges.

To solve those challenges, we have defined system data that can be used throughout the system for linking the data in each function on an FA integrated development environment and developed virtual modules that can 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%, and it proves the effectiveness of our virtualization technology.

1. まえがき

近年は市場ニーズの変化が速く、製品ライフサイクルの短命化が進んでいる。このような市場動向に対応するには、より短い期間で生産設備を立ち上げて早期に製品を生産することが製造業において競争力を高めるには必要不可欠である。

生産設備の立ち上げの際、従来は一般的に実機を用いて制御プログラムのデバッグを行っていた[1)]。しかし、実機でデバッグを行う場合は組立や配線が終わるまでデバッグできないなどの待ち時間が多いことや、干渉を避けるために低速で動かす必要があるなど非効率な点が多い。

実機による生産設備の立ち上げにおいて効率を落とす課題を解決するため、実機を用いない生産設備のデバッグやティーチングを行う仮想化技術が使用されている。しかし、生産設備全体を仮想化するためには複数のシミュレーションソフトウェアを連携する必要があり、シミュレーションソフトウェア間での連携の設定と時間の同期が難しいという課題がある。

これらの課題を解決するために、オムロンが提供する FA 統合開発環境 Sysmac Studio で生産設備の仮想化を実現した。Sysmac Studio は一つのソフトウェアで構成されるという特徴を持っており、これによりデータ連携の自動化

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

や時間の同期を実現した。各機能でデータを連携するためにシステム全体で使えるシステムデータを定義し、このデータをシームレスに相互連携・同期できる仮想モジュール群を開発した。

本稿では、2 章で従来の生産設備開発の課題の詳細を説明する。3 章では実現した仮想化技術の詳細を説明し、4 章では仮想化技術の有効性の検証結果について述べる。そして、5 章にて本稿での総括及び今後の展望と課題を述べる。

2. 課題

2.1 実機のみで行う生産設備立ち上げの非効率性

従来の生産設備立ち上げ時の開発プロセスを図 1 に示す。

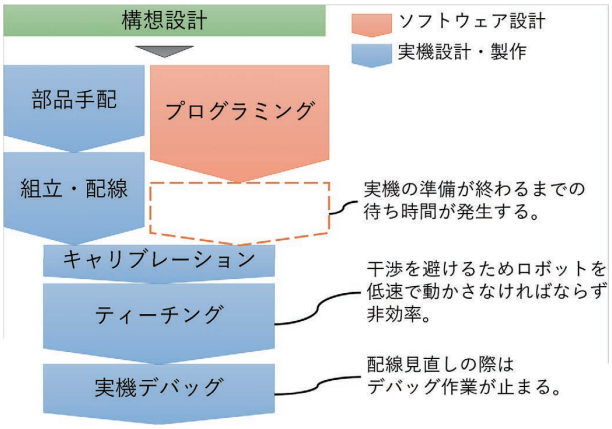

図 1　従来の生産設備立ち上げプロセス

従来、生産設備の開発は一般的に実機を用いて行なっていた。ロボット及び周辺機器の制御プログラムは図 2 に示すシステムを制御することを目的に作られており、制御対象がないと動作を検証できなかったためである[1)]。このため、プログラミングの後、実機の準備が終わるまでの作業待ち期間が生じることが多く、生産設備立ち上げの効率を落とす要因となっていた。また、実機でデバッグする際は機構位置や配線が見直されることがある。見直す間はデバッグが止まってしまうことも効率を落とす要因となっていた。ロボットのティーチング作業においても干渉による破損を避けるため、ロボットを低速で動かさなければならず作業に時間を要していた。

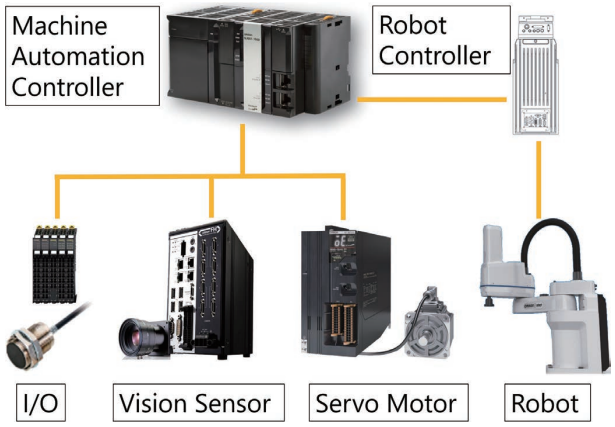

図 2　生産設備の制御システム構成例

2.2 複数のソフトウェアによる仮想化の課題

2.1 で述べた生産設備立ち上げの非効率性の解消を目的として、生産設備全体の仮想化技術が使用されている。生産設備全体を仮想化すれば、実機の組立や配線を行う間に仮想空間上でデバッグやティーチングを進めることができるため、実機のみで行うよりも効率的に設計を進められるためである。

しかし、生産設備全体の仮想化を行うためには、生産設備を構成する全ての機器を仮想化する必要がある。それぞれの機器は異なるシミュレータで仮想化を実現する。複数のシミュレータ間で時間の同期を取ることは難しいため、結果として実機と同じ動作を実現することは困難である[2)]。

3. 生産設備全体の仮想化を一つのソフトウェアで実現

生産設備全体の仮想化を実現するため、複数のシミュレータをオムロンが提供する FA 統合開発環境 Sysmac Studio に統合した。これによって生産設備を構成する全ての要素であるロボット、周辺機器、I/O 機器、画像センサの他、生産設備で作業対象となるワークの仮想化を実現した。

具体的にはロボットと制御機器の制御を一体化するためにマシンオートメーションコントローラーとロボットコントローラーを 1 つに統合した機器、ロボット統合コントローラー[3)] を開発した。そして、ロボット統合コントローラーを PC 上で仮想的に動作させるロボット統合シミュレータを実現し、Sysmac Studio に統合した。これによって一つのシミュレータでロボットと制御機器の I/O データをリアルタイムに連携・同期できるようになった。

次に、生産設備を構成する全ての要素を仮想化するためにモジュール（仮想化モジュール）を開発した。各仮想化モジュールで構成される Sysmac Studio のソフトウェア構造を実現するための課題解決策を 3.1 章で述べる。また、

仮想動作を定義するために仮想化用プログラミング環境を実現する必要がある。仮想化用プログラミング環境の詳細を 3.2 章で述べる。3.3 章以降は、3.1 章で定義した各仮想化モジュールの詳細を述べる。

3.1 生産設備全体の仮想化を一つのソフトウェアで実現するためのソフトウェア構造

Sysmac Studio で生産設備全体を仮想化するために実現したソフトウェア構造を図 3 に示す。また、図 4 にロボット統合コントローラーの生産設備全体の構成例を示す。その構成要素にはセンサ制御に使用する I/O、画像センサ（Vision Sensor）、周辺機器の制御に用いるサーボモータ（Servo Motor）、及びロボット（Robot）が含まれる。

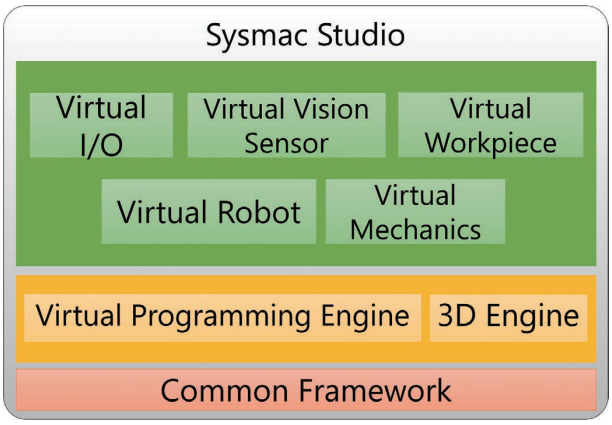

図 3　Sysmac Studio による仮想化のソフトウェア構造

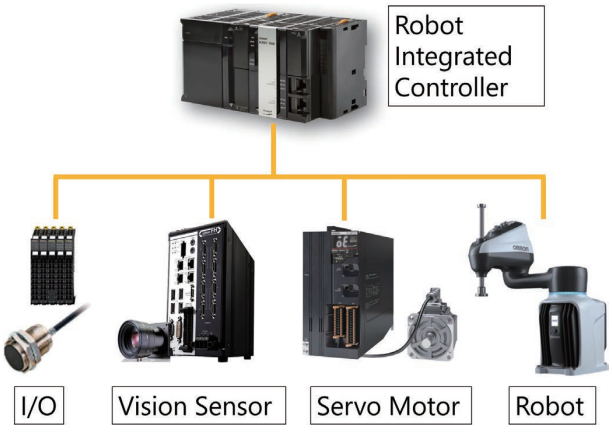

図 4　ロボット統合コントローラーによる生産設備の構成例

仮想化した全てのモジュールを一つのソフトウェアで実現するための基盤となるのが、「Common Framework」である。Common Framework は各モジュールに対して共通機能や共通インタフェースを提供し、モジュールの生成・消滅のライフサイクルを管理するモジュールである。次に、生産設備の制御プログラムやシミュレーション用のプログラムを実行するのが、「仮想プログラミング環境（Virtual Programming Engine）」である。仮想プログラミング環境にはロボット統合コントローラーのシミュレータ及びシミュレーション用プログラム構築環境が含まれる。そして、生産設備全体の 3D モデルの形状データを管理し、3D 表示機能やモデル同士の干渉チェックを実現するのが「3D Engine」である。この 3D Engine 上で、生産設備の要素であるロボット、周辺機器、I/O 機器、画像センサ及びワークを仮想化したモジュールが、それぞれ「仮想ロボット（Virtual Robot）」、「仮想周辺機器（Virtual Mechanics）」、「仮想 I/O（Virtual I/O）」、「仮想画像センサ（Virtual Vision Sensor）」、「仮想ワーク（Virtual Part）」である。このうち、仮想ロボットと仮想周辺機器は 3D モデルを可動部毎に分離し、ロボットの種別に応じたキネマティクスの計算モデルを与えた。そして、制御プログラムの演算結果であるモータの指令値を入力することで、仮想ロボット及び仮想周辺機器の動作を 3D Engine 上で再現した。

一つのソフトウェアで生産設備のシステム全体を仮想化するためには個別機能の持っているデータをシステム全体で使えるシステムデータとして共有することが必要であった。この為には各モジュール間でのデータを、モジュールを超えて共有する必要があった。そうするとモジュール間の相互依存が増えて、複雑度が上がり結果として動作性能の低下やメモリ使用量の増大といった課題が発生する。こうした課題を回避するため、モジュールで扱っているデータを共通データであるシステムデータとして扱えるようにした。具体的には仮想化におけるシステムデータは 3D モデルの位置を表すデータとその状態を記憶する状態変数のデータをモジュール間で共有することが必要である。このため 3D モデルの 3D 位置データや周辺機器の動作位置データなどの異なる位置データを共通の抽象インタフェースを定義して扱えるようにした。これらのデータを 3D Engine 上で表示するときに自動的に 3D Engine 上の絶対座標で扱えるような機能を開発しどのモジュールからも共通の扱いで位置データを管理できるようにした。状態変数についてもシステム全体で一意に扱える階層構造を持った ID 情報を付加して抽象化することでモジュール間の直接結合を防ぎながらデータを共有するシステムデータを実現した。このシステムデータの数は通常の生産設備では 1000 を超える。このためデータの増加に伴う性能低下やメモリ使用量の増大といった課題が生じた。この課題に対して階層構造を持った ID を用いることで単純に並列にデータを並べてデータの受け渡しを行うのに比べて検索量を減らし高速にアクセスできるようにすることで解決した。この ID は相対的な ID となっておりデータの階層が増えてもメモリ使用量の増大も抑えることができた。これらのシステムデータを用いることで大規模な生産設備であってもリアルタイムの動きを確認できる生産設備全体の仮想化を実現した。

3.2 生産設備全体の仮想化のためのシミュレーション用プログラム構築環境の実現

生産設備全体の動作を仮想化するには、ワークのようなアクチュエータによって直接制御されない物体の仮想動作をプログラムする必要がある。なぜならワークの仮想動作を定義するための生成条件や表示位置は制御プログラムに含まれないためである。また、センサなどの I/O 機器についても仮想空間では実際の I/O が接続されていないため、仮想的に I/O の動作を定義する必要がある。

このような仮想空間上における動作は、単純かつ定型的な定義よりも制御プログラムの状態によって柔軟に定義できることが求められるため、専用言語を使用した仮想化用のプログラムによって定義されることが多い。一般的に、仮想化用のプログラムには専用言語が使用される。しかし、専用言語を習得するには時間を要する[4]。そこで、Sysmac Studio には汎用プログラム言語である C# 言語をベースとしたスクリプト言語（シェイプスクリプト）を開発した。

シェイプスクリプトの実行時は I/O の動作などの制御情報をリアルタイムに取得し、ワークなどの表示位置を制御に合わせて変更することが必要となる。このリアルタイム性を実現するには Sysmac Studio とメモリ空間を共有できる同一プロセスで実行されなければならない。ここでプロセスとは、Windows におけるプログラムの実行単位である。シェイプスクリプトを同一プロセスにすることでメモリ空間を共有し、リアルタイムでの情報交換が可能となる。これを実現するために、シェイプスクリプトを実行前にコンパイルしアセンブリを生成して同一プロセスにロードする仕組みを構築した。しかし、単純にシェイプスクリプトを同一プロセスにロードすると、シェイプスクリプト停止時に共有されたメモリを開放できないという問題が発生する。そこで、同一プロセスにおいてメモリ空間を分ける仕組みを構築した。

シェイプスクリプトと Sysmac Studio は Windows 上で動作するアプリケーションであるため、Microsoft の提供する共通ライブラリである .NET Framework を活用している。.NET Framework は、Windows プログラムの動作環境である共通言語ランタイム（CLR: Common Language Runtime）の 1 つである。CLR にはアプリケーション・ドメインという実行コードの管理単位がある。アプリケーション・ドメインを分離すると、メモリ空間を分けることが可能になる。そこで、図 5 に示すようにシェイプスクリプトと Sysmac Studio のアプリケーション・ドメインを分離した。これにより、シェイプスクリプトのアセンブリがロードまたはアンロードされても、Sysmac Studio には影響を与えないようにした。

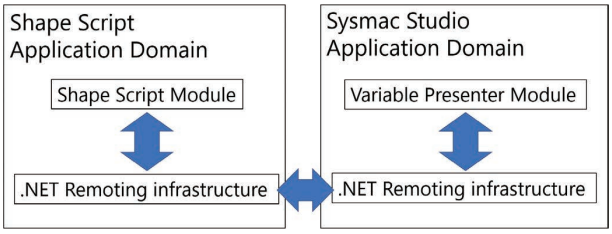

図 5　シェイプスクリプトと Sysmac Studio 間の通信

定義したい動作を実行する条件は、ロボット統合コントローラーの状態によって変わることが多い。これらの状態は変数に格納される。このため、シェイプスクリプトはロボット統合コントローラーのシミュレータから変数の値を取得する必要がある。

Sysmac Studio には、指定した変数の値をロボット統合コントローラーのシミュレータより取得する機能を備えている。シェイプスクリプトと Sysmac Studio のアプリケーション・ドメインを分離したため、アプリケーション・ドメイン間での通信を行うことでシェイプスクリプトから変数の値を取得できるようにした。

図 5 に示したように、アプリケーション・ドメイン間の通信を行うために、Sysmac Studio のアプリケーション・ドメイン内で取得した状態変数を抽象化し、各モジュール間で共有できる仕組みを提供する専用モジュール（Variable Presenter モジュール）を開発した。これを用いることでモジュールを超えた状態変数のマーシャリング（データ交換）を実現した。同様に、シェイプスクリプトのアセンブリをマーシャリングの対象とすることで、システムデータのマーシャリングができる仕組みを実現した。マーシャリングにおいてはデータ交換をする上でデータを一旦シリアライズしてメモリ空間に格納し、このメモリをアプリケーション・ドメイン間で交換した後でデシリアライズすることでデータ交換を実現しているため、データ構造が性能やメモリサイズに直接影響を与える。この性能を確保するためにアプリケーション・ドメインを超えて再構築できる情報は除き、仮想化を実現するための位置や状態を持つデータなどのみをマーシャリングの対象とすることでロボット統合コントローラーのシミュレータが持つ状態変数の高速なマーシャリングを実現した。

3.3 生産設備全体の仮想化のための仮想ワークの実現

3.2 で述べた通り、仮想ワークは制御プログラムのシミュレーションを実行するだけでは仮想化を実現できない。そこで仮想化空間における動作を記述するシェイプスクリプトを使用することで、制御プログラムのシミュレーションに合わせて仮想ワークの状態を定義できるようにした。

仮想ワークを動作させるには、まずはその条件をシェイプスクリプトに定義する必要がある。条件は、ロボット統

合コントローラーの状態から決める。例えば、仮想ワークがロボットハンドに把持される動作を定義する場合、ロボットハンドに対して把持信号が入力されたことを検知するスクリプトを記述することで定義できる。

次に、条件成立時の仮想ワークの状態をシェイプスクリプトに定義する必要がある。設定する状態は、仮想ワークの表示・非表示状態または表示位置（座標）である。表示・非表示状態は、仮想ワークが生産設備に搬入されたら表示、搬出されたら非表示にする。表示位置は、例えば、ロボットハンドに対して把持信号が入力されたことを検知したら、仮想ワークがロボットハンドに追従するように定義する。これらの設定ができるように、シェイプスクリプトで仮想ワークのインスタンスを生成し、各インスタンスの表示・非表示状態及び表示位置を変更できるようにした。

また、ワークに関連する不具合が発生したとき、ワークの位置情報が重要になる。例えば、ロボットがコンベア上を流れるワークを把持できなかった場合、そのワークはどの位置を流れていたかを調査する必要がある。そこでデバッグを容易に行えるように、仮想化された生産設備における仮想ワークの表示位置をトレースし、結果をグラフに表示することで、仮想ワークに生じた問題を簡単に確認・デバッグできるようにした。表示位置のグラフの例を図 6 に示す。

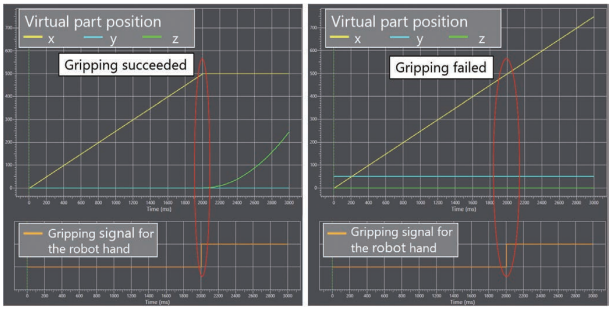

図 6　仮想ワークの表示位置のトレース結果

図 6 では、仮想空間上を x 軸方向にコンベアで搬送される仮想ワークをロボットハンドが把持して真上に移動した時の x、y、z 座標の変化を示している。図 6 の左のグラフでは、ロボットハンドの把持信号が入力されると同時に、仮想ワークの x 方向の動きが止まり、z 方向に上昇していることが分かる。それに対し、図 6 の右のグラフでは、把持信号が入力されても x 方向の動きが止まらないため、把持に失敗したことが分かる。左右のグラフを比較すると、仮想ワークの位置が y 方向にずれていることが分かる。以上の情報から仮想ワークの位置またはロボットが把持する位置を y 方向に補正すれば良い。このように仮想ワークの位置をトレースすることで、問題発生箇所を簡単に特定できる。

3.4 生産設備全体の仮想化のための仮想 I/O の実現

3.2 で述べた通り、仮想化された生産設備には実際の I/O は接続されていない。このため、I/O の仮想動作を定義する必要がある。生産設備で多用される I/O には光電センサのようなワークを検出するセンサと、エアシリンダに取り付けられたリミットスイッチのような機構の動作位置を検出するセンサがあるため、「仮想ワーク検出用センサ」と「仮想動作位置検出用センサ」を仮想 I/O として実現した。

両センサに共通することは、仮想ワークの検出等の条件が成立したら、対応する I/O 変数の値を True または False に切り替えることである。I/O 変数はロボット統合コントローラーに定義されているため、シェイプスクリプトから Variable Presenter モジュールを通して設定することができる。

仮想ワーク検出用センサの I/O 変数が切り替わる条件は、仮想ワークがあるエリアに入った時、またはあるエリアから出た時である。この条件を設定できるように、仮想ワーク検出用センサの検出エリアを 3D Engine 上に表示できるようにした。そして、仮想ワーク検出用センサの検出エリアと仮想ワークの 3D モデルが 3D Engine 上で干渉しているかどうかを I/O 変数を切り替える条件とした。3D Engine 上に表示した仮想ワーク検出用センサ（Virtual Part Detection Sensor）と仮想ワーク（Virtual Part）の 3D モデルの例を図 7 に示す。

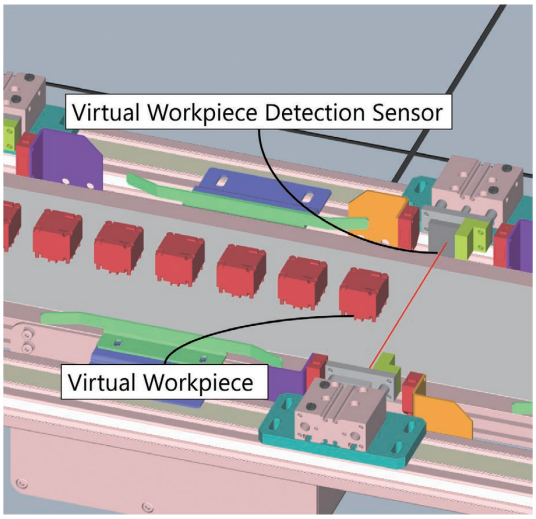

図 7　仮想ワーク検出用センサと仮想ワークの 3D モデル

3D モデル同士の干渉は、3.6 にて後述する干渉チェックにより 3D Engine が検出する。3D Engine が検出した干渉を、シェイプスクリプトが 3D モデルのインスタンスを参照することで取得できるようにした。これにより、設計者が仮想ワーク検出用センサと仮想ワークの 3D モデルが干渉したときに I/O 変数を切り替えるシェイプスクリプトを記述するだけで、実際の I/O 機器の動作を補うスクリプト

を簡単に作成できるようになった。

次に、仮想動作位置検出用センサの I/O 変数が切り替わる条件は、周辺機器の可動部がある場所に移動した時である。エアシリンダの場合、ピストンの押出時及び引込時に I/O 変数が切り替わるため、エアの注入を開始する I/O 変数が True になってから一定時間経過した後にピストンの押出完了信号に対応する I/O 変数を切り替える必要がある。

I/O 変数の変化は、Variable Presenter モジュールを使用したシェイプスクリプトを記述することで検知できる。また、ロボット統合コントローラーのシミュレータは時刻情報を変数に格納しており、I/O 変数と同様にシェイプスクリプトを記述することで時刻情報を取得し、時間経過を計測できる。以上より、エア注入開始 I/O 変数が True となった後、一定時間経過後に仮想動作検出用センサの I/O 変数を切り替えるシェイプスクリプトを簡単に作成できるようになった。

ワークを検出するセンサや周辺機器の動作位置を検出するセンサは、生産設備によって 100 を超えることがあり、これら全ての仮想動作をシェイプスクリプトに定義するにはかなりの工数を要してしまう。しかし、I/O の仮想動作の場合、仮想ワークとは異なり定型的なコードになる。そこで、各仮想 I/O に対応する I/O 変数等の簡単な設定をするだけでシェイプスクリプトによる仮想動作の定義を自動で行えるようにした。これにより、シェイプスクリプトでの I/O の仮想動作を短時間で定義できるようになった。

3.5 生産設備全体の仮想化のための仮想画像センサの実現

生産設備には、コンベア上をランダムに流れてくるワークの位置の検出や、製品の外観検査を行う等の目的で画像センサが使用される。そこで、画像センサのエンジンを PC で仮想的に実行できる仮想画像センサを開発した。あらかじめ用意した画像を仮想画像センサに入力することで、ワークの位置の検出や外観検査のシミュレーションを実行できるようになった。

画像センサを使用する際、実機の場合は画像センサ、ロボット及び周辺機器の位置を合わせるキャリブレーション作業が必要である。仮想空間上では画像センサ、周辺機器、ロボットを配置するだけで自動的にキャリブレーションできるようにした。更に、シェイプスクリプトを使用することで複数の画像を用いた画像センサのワークの仮想化が可能となった。これは仮想ワークを自動的に生成し、画像を用いた複数種類の仮想ワークの位置検出を現実の生産設備と同様に再現できるようになったことを示している。これにより、仮想空間上でワークのコンベアトラッキングを行う場合、配置するだけでロボットが正確に仮想ワークを把持できるようになった。仮想画像センサにより位置を検出し、複数種類の仮想ワークのコンベアトラッキングを行う例を図 8 に示す。図 8 の（a）はワークを把持する例、（b）はワークを置く例を示している。仮想ワークの種類ごとに 3D 上でも把持された順番を表示することでシステム全体としてのシステムデータワーク管理を実現し、その可視化も行えるようにした。

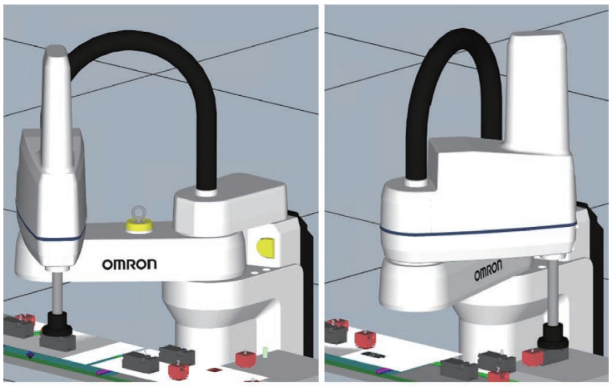

図 8　仮想画像センサによるコンベアトラッキング

3.6 生産設備全体の仮想化のための干渉チェックの実現

生産設備の立ち上げにおいて最も時間がかかる作業の一つがロボットのティーチングである。ティーチングではロボットの TCP（Tool Center Point）を覚えさせたい位置まで移動させ、その場所を記録することでティーチングポイントを作成する。そして、記録したティーチングポイントを繋ぐロボットプログラムを作成することで、ロボットのパスを生成する。

実際のロボットを使用したティーチングでは、干渉を回避するためロボットを低速かつ慎重に動かさなければならない。それに対し、仮想空間上でのティーチングは、仮想空間上で干渉しても機器が壊れる恐れがなく、実際のロボットよりも高速で動かすことができるため、短い時間で行うことができる。仮想空間上でオフラインティーチングを事前に行うことで、実際のロボットでティーチングする際はティーチングポイントの微調整を行うだけで良いため、ロボットのティーチング作業時間を短縮することができる。

オフラインティーチングで重要となるのが、干渉しないパスの生成である。仮想空間上でロボットと周辺機器が干渉した場合、ティーチングポイントを設定し直すことで干渉を回避することができる。したがって、ロボットがパスを描く上で、どのタイミングでどこが干渉したかを検出する必要がある。

そこで、3D Engine に干渉チェックを導入し、干渉した 3D モデルの色を変更して通知するようにした。一般的に 3D 空間上での干渉チェックでは、表示されている 3D モデルを使って干渉チェックを行うと、チェックに要する計算時間が長くなり、かつメモリの消費量が大きくなってしまうため、3D モデルを簡略化した境界ボックスを内部的に

生成して干渉をチェックすることが多い。Sysmac Studio の 3D シミュレーションでは、凸包（Convex Hull）または近似凸分解（Approximate Convex Decomposition）[5)6)] を用いて境界ボックスを生成することで干渉チェックを実現した。以下に凸包および近似凸分解によって境界ボックスを生成した例を図 9 に示す。

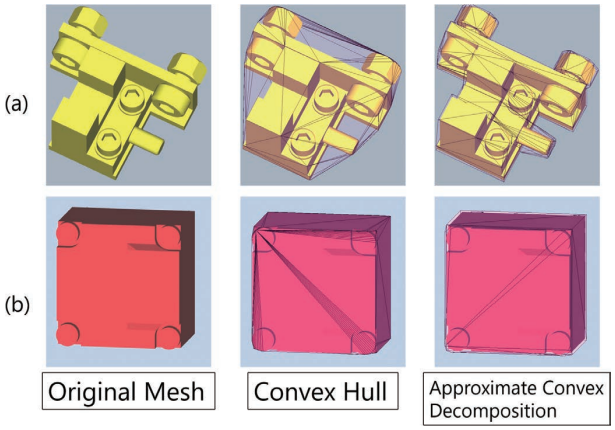

図 9　凸包と近似凸分解による境界ボックス

凸包または近似凸分解は、3D モデルの特性によってどちらを選択すれば良いかが異なる。(a) のように複雑な形状の場合は近似凸分解の方が表示上の 3D モデルに近似されるが、(b) のように比較的単純な形状だと計算コストの少ない凸包の方が良い。このように 3D モデルによってユーザが最適な境界ボックスを選択できるようにした。

オフラインティーチングの後、仮想空間上で生産設備全体のシミュレーションを行う際、周辺機器や仮想ワークの位置がオフラインティーチング時とは異なることがある。このため、生産設備全体のシミュレーションの際にロボットが周辺機器やワークと干渉することがある。干渉が検出された場合、再度オフラインティーチングが必要となる。この時、シミュレーション中にどのタイミングでどこが干渉したかが正確に分からないと、どのティーチングポイントを修正すれば良いか特定できなくなり、再オフラインティーチングに時間を要してしまう。

シミュレーション中、干渉したときに 3D モデルの色を変えて通知するだけでは干渉を見逃してしまう可能性がある。干渉を見逃さないようにするために、干渉タイミングと干渉箇所の情報をシェイプスクリプトの実行によりログに出力できるようにした。3.4 で述べた通り、シェイプスクリプトは 3D Engine 上の 3D モデル同士の干渉を取得できる。したがって、3D モデル同士の干渉時、シェイプスクリプトに干渉した時のタイムスタンプと干渉したインスタンス名をログに出力するコードを記述することで、干渉タイミングと干渉箇所の確認が可能となり、再オフラインティーチングの手間を省くことができた。

4. 生産設備全体の仮想化による有効性の検証

生産設備全体を仮想化することによる有効性を検証するため、生産設備の立ち上げを行った。検証に使用した生産設備の上面図を図 10 に示す。図 10 の生産設備は、ワーク供給トレイで搬入されたワークを垂直多関節ロボット Viper650 によって組み立てる検証用の生産設備である。

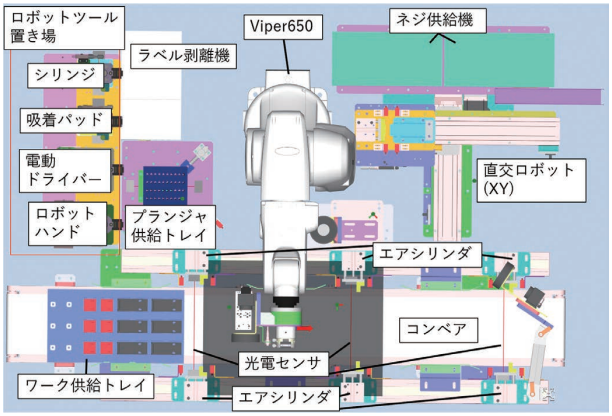

図 10　検証用の生産設備の上面図

検証は組立・配線が既に行われた生産設備に対して制御プログラムの作成やデバッグ、キャリブレーション及びティーチングを行い、各工程の作業に要した工数を記録し、仮想化技術未使用時と使用時の工数を比較した。比較結果を図 11 に示す。

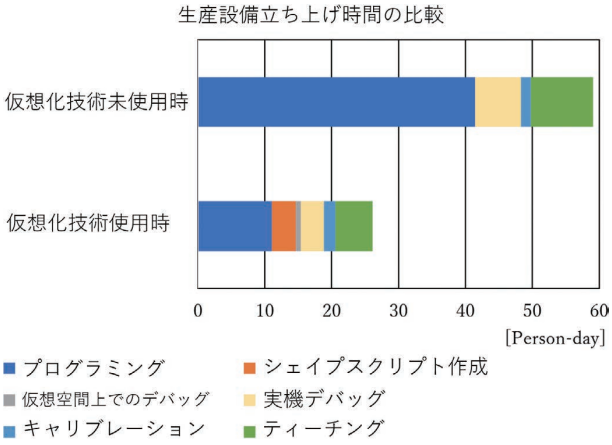

図 11　生産設備の仮想化による有効性の検証結果

図 11 に示した通り、仮想化技術を使用しなかった場合、立ち上げには約 59 人日要した。それに対し、仮想化技術を使用した場合は約 26 人日となり、仮想化技術を使用することで立ち上げ工数を約 56%削減することができた。

削減効果が最も大きかったのはプログラミング工程である。仮想化技術を使用しなかった場合、プログラミング工程には約 41 人日要したが、仮想化技術を使用した場合はシェイプスクリプト作成工数と合わせて約 15 人日となり、

工数を約 63%削減することができた。仮想化技術を使わず実機のみでプログラミングを行う場合、干渉を避けるため組立手順ごとに単機能の設計・実装作業を実施する必要があった。そのため手順全体で使う状態変数や手順をまたぐ機能に関しては手順全体の結合時に追加設計・実装が必要であった。それに対し、仮想化技術を用いた場合は干渉を気にする必要がないため単機能の設計・実装作業を行わずに最初から組立手順全体で作業を進められることが検証を通して分かった。このため追加設計・実装無しで作業を行うことができ、生産設備全体のプログラミング工数を大幅に削減できた。

次に削減効果が大きかったのは実機デバッグ工程で、仮想化技術を使用しなかった場合は約 7 人日要したが、仮想化技術を使用した場合は仮想空間上でのデバッグ工程と合わせて約 4 人日となり、工数を約 43%削減することができた。仮想空間上でのデバッグ工程は、ロボットや周辺機器を高速で動かすことができたため実機でのデバッグより短時間で行うことができた。また、一度実施したデバッグを再度行う場合、実機ではロボットや周辺機器、ワークの状態を全て初期位置に戻す必要があるが、仮想空間ではそれらの状態をリセットするだけで良いため工数を削減できた。仮想空間上でのデバッグが完了すれば実機でのデバッグはワークの吸着・把持の調整など実機での調整が必須の作業のみ行うため、総じてデバッグ工数を削減できた。

今回の検証では生産設備の組立・配線時間およびその待ち時間が含まれていない。したがって、実際の生産設備の立ち上げではプログラミング及びデバッグ工程に要する時間のより大きな削減が期待できる。

3 番目に削減効果が大きかったのはティーチング工程である。仮想化技術を使用しなかった場合は約 9 人日要したが、仮想化技術を使用した場合は約 5.5 人日となり、工数を約 39%削減することができた。2.1 に記載した通り、実機でティーチングを行う場合、干渉による機器の破損を避けるためロボットを低速で動かさなければならない。それに対し、オフラインティーチングでは仮想ロボットを高速に動かすことができる。したがって、オフラインティーチングを行い実機では微調整のみ行うことで実機での作業時間を削減でき、結果としてティーチング工数を削減できた。

5. むすび

生産設備の立ち上げ工数の削減を目指し、FA 統合開発環境である Sysmac Studio に生産設備の仮想化技術を導入した。仮想化技術によって I/O、画像センサ、周辺機器やロボットを仮想空間上で動作させることを可能にした。また、仮想ワークの動作を柔軟に定義するシミュレーション用プログラム構築環境を実現した。更に、全ての機器及び仮想ワーク間の干渉チェックを導入することでオフラインティーチングを可能とした。

仮想化技術の有効性を検証するために生産設備の立ち上げを行った。検証の結果、仮想化技術を使用することで生産設備の立ち上げ工数を約 56%削減できることが分かった。

プログラミング及びデバッグ工程は仮想化技術によって工数を十分削減できた。今後は仮想化技術以外の手法でプログラミングエディタの改善等による更なる工数削減を検討する。また、プログラミング及びデバッグの次に工数を要する工程はロボットのティーチングである。今後はオフラインティーチングの工数削減を目指し、始点と終点のみを指定するだけでパスを自動で生成するパスプランニング技術の開発に取り組んでいく。

参考文献

1）宮内孝，小林大輔，藤田和明．設備制御ソフトウェア開発の効率を向上させる実機レス デバッグシステムの適用．東芝レビュー．2009, Vol. 64, No. 5, p.10−13.

2）日本ロボット工業会．“ロボットシステムインテグレータのスキル読本［第一版］”．経済産業省．2018−5−31．https://www.meti.go.jp/press/2018/05/20180531008/20180531008−1.pdf,（参照 2021−01−04）.

3）オムロン株式会社．“ロボット統合システム”．2020−02−09．https://www.fa.omron.co.jp/product/robotics/lineup/integratedcontroller/,（参照 2021−01−13）.

4）Hosseinpour, F.; Hajihosseini, H. “Importance of Simulation in Manufacturing”. World Academy of Science, Engineering and Technology 27, 2009, p.285−288.

5）Mamou, K.; Ghorbel, F. “A simple and efficient approach for 3D mesh approximate convex decomposition”. 16th IEEE International Conference on Image Processing（ICIP）, 2009, p.3501−3504.

6）Lien, J.-M.; Amato, N. M. “Approximate convex decomposition of polygons”. Proceedings of the 20-th Annual Symposium on Computational Geometry, 2004, p.17−26.

執筆者紹介

島川 はる奈　SHIMAKAWA Haruna
インダストリアルオートメーションビジネス
カンパニー
商品事業本部　コントローラ事業部
ソフトウェア開発部
専門：ソフトウェア工学
所属学会：日本ロボット学会

岩村 慎太郎　IWAMURA Shintaro
インダストリアルオートメーションビジネス
カンパニー
商品事業本部　コントローラ事業部
ソフトウェア開発部
専門：ソフトウェア工学
所属学会：日本ロボット学会