Virtual Commissioning for an Overhead Hoist Transporter   
in a Semiconductor FAB

**Sang H. An, Sang C. Park**

Dept of IE, Ajou University, San 5,

Woncheon-dong, Yeongtong-gu, Suwon, Korea

**Abstract**: Presented in the paper.

**Key words:** HILS, OHT, Virtual commissioning, Control software verification

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\*Correspondence: Sang C. Park ([scpark@ajou.ac.kr](mailto:scpark@ajou.ac.kr))

Department of Industrial Engineering

Ajou University

San 5, Woncheon-dong, Yeongtong-gu

Suwon 443-749, Korea

**1. INTRODUCTION**

Manufacturing meets new threshold after industry 4.0. Most people need unique things themselves. So many manufacturers are considering mass customization. On the other hand, COVID-19 shocked the world. Manufacturing is no exception. Supply network is unstable. And supply chain resilience is one of important things in manufacturing industry. For these reasons, manufacturer have to make diverse products based on mass production environment. Making diverse products effort manufacturing complexity. And optimized transportation is one of methods solve production efficiency and reducing costs. **AGV (Automated Guided Vehicle) can**

**+ AGV 스케줄링과 디스패칭에 대한 설명**

✁ Figure 1. Material transport system in a FAB ✁

✁ Figure 2. Commissioning configurations ✁

Dispatching rule typically classified a vehicle-initiated task and a workcenter-initiated task assignment (Tanchoco 1984). Depending on the situation, traditional dispatching rules are used to dispatch AGVs using only single method or multi mixed methods. These rules are First Come First Served (FCFS), Shortest Travel Distance (STD), Earliest Due Date first (EDD), Longest Waiting Time (LWT), Nearest Vehicle First (NVF), Maximum Queue Size (MQS), etc. To solve dispatching problem, some cases adopted reinforcement learning. **Reinforcement learning is a machine learning method that can constantly adjust agent’s behavior through trial and error** (Kaelbling, Littman & Moore, 1996). **[Scheduling problem]** An reinforcement learning based approach for a multiple-load carrier scheduling problem (Chen, Xia, et al. 2015), and they proposed Q() model improve throughput and reduce travel cost. Vehicle-initiated task assignment approach production scheduling problem using Q-learning algorithm (Wang and Usher, 2005). In dynamic job shop scheduling problem approach using reinforcement learning. **The other effective cases**,

**2. PROBLEM FORMULATION**

In this paper, t-step simulation environment clip image as input approach real-time AGV dispatching problem using multi-agent method of reinforcement learning conclude CNN and GNN.

2.1. State representation

**State raw observation full state**

State at t-step is separable term represented multi-matrix form meaning 3-channel image (127 by 127). Environment clip image is program screen of human level. And Feature Image show AGVs, Current Job, and Simulation attributes as color image. conclude Red channel image is represented job information. Green Channel image is represented each link’s driving constraints such as direction, velocity, and rotation. Blue channel image is represented sequence information about time-horizon AGV routing left.

Finally, mask make block unnecessary region. It’s predefined static matrix.

2.2. Reward representation

Many researcher are trying well-made reward function, several good reward function design cases, ‘Deep Mimic’ and ‘GAIL(**dd**)’ are used exponential form into kinimetic models. If demonstrate agent get valuable experience by non-linear functions. Also , we refer multi attributes rule, selected throughput, waiting time, and mileages. We make non-negative reward function.

2.3. Action representation

Action represents the dispatching rule of the AGV’s system and is defined by a encoding value Dispatching rules in action are First Come First Served (FCFS), Shortest Travel Distance (STD), Earliest Due Date first (EDD), Longest Waiting Time (LWT), Nearest Vehicle First (NVF), and Maximum Queue Size (MQS).

The state represents separable matrix form at time t . Each observation merged to state in AGV. Critic network adjust weights from value function. And Actor network give action strategy to Environment. Mix-up method is one of feature extract methods. And select RasNet-50 fine-tune method transfer learning

2.3. Policy representation

Policy is a approximator estimated future action by decision boundary in reinforcement learning. Generally, policies are adjusted by value function.

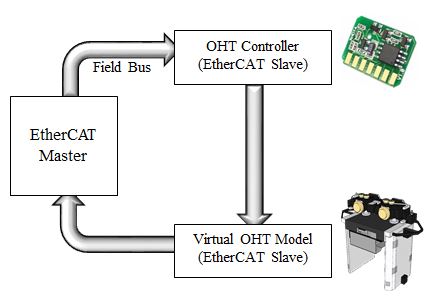
Value function 수식

GraphMix

Q-learning

✁ Figure 3. OHT design & production procedures ✁

To cope with the problems of the conventional procedure, we propose a concurrent procedure of mechanical and electrical designs including the virtual commissioning phase, as shown in Figure 3-(b). To achieve the concurrency, we separate a virtual OHT model into two parts, a physical model and a logical model. The mechanical engineer and the electrical engineer can do their jobs concurrently without interfering with each other, since the physical model and the logical model can be defined independently. Both the physical model and the logical model are defined; then, we can simply define a virtual OHT model by combining the two sub-models. For the virtual commissioning of an OHT, it is necessary to perform a simulation involving a virtual OHT model and a real controller, connected through a fieldbus system (EtherCAT). Then, the virtual commissioning can be performed to fix various errors caused by mechanical models not being properly adjusted, and faults in the control programs. We can minimize the stabilization time before the production phase, since most of errors are fixed through the virtual commissioning.



✁ Figure 4. Communication structure on the EtherCAT environment ✁

For the virtual commissioning with high fidelity, it is necessary to observe the communication mechanism in an OHT. For the real-time motion control of an OHT, it is necessary to have a fast fieldbus system through which the OHT controller to communicate with sensors and actuators (servo motors). The real-time motion control of an OHT requires very short cycle time (≤ 100 µs) with low communication jitter (≤ 1 µs) for precise synchronization. EtherCAT protocol (<https://www.ethercat.org/default.htm>) is standardized in IEC 61158, and it has been known to be the fastest Ethernet-based fieldbus system which may synchronizes with nanosecond accuracy. Since the rapid reaction times are very essential for the precise synchronization, it is common to use the EtherCAT protocol for the communication fieldbus system of an OHT.

As shown in Figure 4, the EtherCAT environment consists of one ‘master’ and two ‘slaves’; 1) ‘OHT controller slave’ representing the real OHT controller, and 2) ‘OHT model slave’ representing all actuators and sensors belonging to the OHT. For the full verification of the OHT controller though the virtual commissioning, it is essential to have a virtual OHT model which operates exactly the same with the real OHT device on the EtherCAT environment. By doing so, it is possible to make the controller assume that it is controlling the real OHT device instead of the virtual OHT model (Drath et al., 2008; Hibnio et al., 2006; Hoffman et al., 2019; Huang & Yeh, 1999).

As shown in Figure 3-(b), the logical model of an OHT should be able to communicate with the real controller on the EtherCAT environment. This paper employs Zeigler’s DEVS (Discrete Event Systems Specifications) formalism (Zeigler, 1984; Kim, 1994; Ham et al., 2019; Chang & Park, 2018) to build such a logical model. To make this paper self-contained, a brief explanation on the DEVS formalism is given bellow. Within the DEVS formalism, one must specify two types of sub-models: 1) the atomic model, the basic models from which larger models are built, and 2) the coupled model, how atomic models are connected in a hierarchical manner. Formally, an atomic model ***M*** is specified by a 7-tuple:



***X***: input events set;

***S***: sequential states set ;

***Y***: output events set;

: ***S*** 🡪 ***S***: internal transition function;

: ***Q*** \* ***X*** 🡪 ***S***: external transition function

***Q*** = {(*s*,*e*)| s ∈ ***S***, 0 ≤*e*≤ (*s*)}: total state of ***M***;

: ***S*** 🡪 ***Y***: output function;

: ***S*** 🡪 ***Real***: time advance function.

The four elements in the 7-tuple, namely  and, are termed the characteristic functions of an atomic model. The second form of the model, termed a coupled model, shows a method to couple several component models together to form a new model. Formally, a couple model ***DN*** is defined as:



***X***: input events set;

***Y***: output events set;

***M***: set of all component models in DEVS;

***EIC*** ⊆ ***DN.IN*** \* ***M.IN***: external input coupling relation;

***EOC*** ⊆ ***M.OUT*** \* ***DN.OUT***: external output coupling relation;

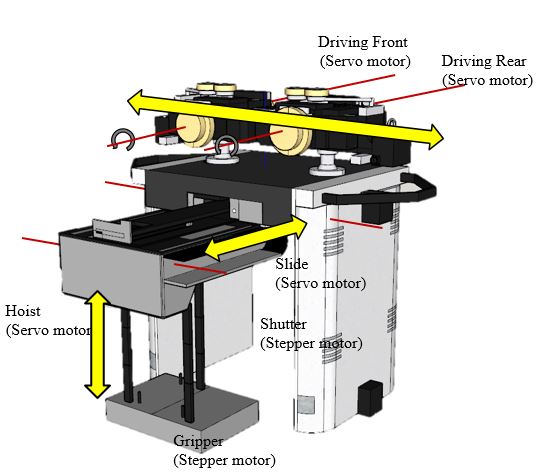
***IC*** ⊆ ***M.OUT*** \* ***M.IN***: internal coupling relation;

***SELECT***: 2M −Ø 🡪 ***M***: tie-breaking selector,

where the extensions .IN and .OUT represent the input port set and the output port set of the respective DEVS models. The detailed OHT model construction methodology is addressed in the following section.

**3. VIRTUAL OHT MODEL CONSTRUCTION**

As shown in Figure 3-(b), the virtual OHT model consists of a physical model and a logical model. As shown in Figure 5, the physical OHT model may be represented in the form of a geometric model with kinematics. While the physical OHT model includes the inherent functionalities of actuators and sensors, the logical OHT model needs to communicate with the real controller on the EtherCAT environment. To build the logical OHT model slave, this paper employs the DEVS formalism. Firstly, it is necessary to identify tasks assigned to the OHT model slave. Once tasks are identified, we can use an atomic model of the DEVS formalism to construct the slave model. At this time, every task becomes an internal transition () of the slave model.

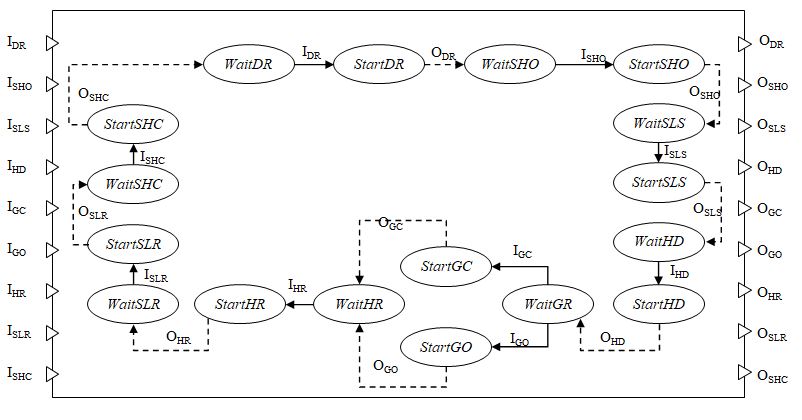


✁ Figure 5. Task identification of an OHT model ✁

The main mission of an OHT is to move a FOUP from a starting point to a destination point, and it consists of four major steps; 1) Move to the starting point in an empty state, 2) Load a FOUP at the starting point, 3) Move to the destination point with the FOUP, and 4) Unload the FOUP at the destination point. Among the four steps, the FOUP loading and unloading steps require many tasks from shutter, slide, hoist and gripper. An OHT device has multiple motions requiring actuators such as ‘servo motors’ and ‘stepper motors’. While servo motors are used for precise control requiring feedback sensors (closed-loop control), stepper motors are suitable for less precise control without feedback sensors (open-loop control). Typically, an OHT has four servo motors (two for driving, one for slide, and one for hoist) and two stepper motors (one for shutter and one for gripper). Each motor has corresponding tasks. By analyzing those tasks, we identify 9 tasks, as shown in Table 1.

[Table 1] Tasks of an OHT

|  |  |  |
| --- | --- | --- |
| **Task** | **Corresponding actuator** | **Function** |
| Drive (DR) | Two servo motors | Drive OHT on the overhead track (OHT network) |
| Shutter open (SHO)  Shutter close (SHC) | One stepper motor | Open shutter before slider stretches  Close shutter after slider retrieves |
| Slider stretch (SLS)  Slider retrieve (SLR) | One servo motor | Stretch slider before host drops  Retrieve slider after host raises |
| Hoist drop (HD)  Hoist raise (HR) | One servo motor | Drop hoist to access a load port  Raise hoist with or without a FOUP |
| Gripper close (GC)  Gripper open (GO) | One stepper motor | Close gripper to load a FOUP  Open gripper to unload a FOUP |



✁ Figure 6. DEVS model representing the logical OHT model ✁

As mentioned earlier, a DEVS atomic model consists of three sets and four characteristic functions (), and it is possible to construct a DEVS model representing the logical OHT model based on the identified tasks, as shown in Figure 6. The construction methodology can be described as follows.

* ***DEVS model construction methodology based on the tasks of an OHT model***

// Input: {*DR, SHO, SLS, HD, GC, GO, HR, SLR, SHC*} (a set of OHT tasks) and *tDR*, *tSHO, tSLS, tHD*, *tGC, tGO* , *tHR, tSLR* , *tSHC* (task durations)

// Output: a logical model (DEVS atomic model)

Step 1) Define the input event set (***X***): For each task of the OHT model, it is necessary to define a triggering input event from the OHT controller.

***X*** ={*IDR, ISHO, ISLS, IHD, IGC, IGO, IHR, ISLR, ISHC*};

Step 2) Define the states set (***S***): Each OHT task needs to have two corresponding states; one is waiting for a triggering event from the OHT controller; the other, starts the corresponding task.

***S***={*WaitDR, StartDR, WaitSHO, StartSHO, WaitSLS, StartSLS, WaitHD, StartHD, WaitGR, StartGO, StartGC, WaitHR, StartHR, WaitSLR, StartSLR, WaitSHC, StartSHC*};

Step 3) Define the output events set (***Y***): Whenever a task is done, it is necessary to inform the OHT controller. Define an output event for each task to notify the OHT controller of the end of the task.

***Y*** (output events set) = {*ODR, OSHO, OSLS, OHD, OGC, OGO, OHR, OSLR, OSHC* };

Step 4) Define internal transition functions (): For each task, it is necessary to define an internal transition function.

(*StartDR*) = *WaitSHO;* (*StartSHO*) = *WaitSLO;*

(*StartSLS*) = *WaitHD;* (*StartHD*) = *WaitGR;*

(*StartGC*) = *WaitHR;* (*StartGO*) = *WaitHR;*

(*StartHR*) = *WaitSLC;* (*StartSLR*) = *WaitSHC;*

(*StartSHC*) = *WaitMove;*

Step 5) Define external transition functions (): Each OHT task is triggered by an input event given by the OHT controller. Define an external transition function for each external event.

(*WaitDR, IDR*) = *StartDR;* (*WaitSHO, ISHO*) = *StartSHO;*

(*WaitSLS, ISLS*) = *StartSLS;* (*WaitHD, IHD*) = *StartHD;*

(*WaitGR, IGC*) = *StartGC;* (*WaitGR, IGO*) = *StartGO;*

(*WaitHR, IHR*) = *StartHR;* (*WaitSLR, ISLR*) = *StartSLR;*

(*WaitSHC, ISHC*) = *StartSHC;*

Step 6) Define output functions (): The OHT model needs to inform the OHT controller that a task is done. Define an output function for each state having an internal transition function.

 (*StartDR*) = *ODR;*  (*StartSHO*) = *OSHO;*

 (*StartSLS*) = *OSLS;*  (*StartHD*) = *OHD;*

 (*StartGC*) = *OGC;*  (*StartGO*) = *OGO;*

 (*StartHR*) = *OHR;*  (*StartSLR*) = *OSLR;*

 (*StartSHC*) = *OSHC;*

Step 7) Define time advance functions (): Define a time advance function for every state. Observe that the duration for a state having no internal transition functions becomes infinite (∞), because it cannot escape the state without an external trigger from the OHT controller.

 (*StartDR*) = *tDR;*  (*StartSHO*) = *tSHO;*

 (*StartSLS*) = *tSLS;*  (*StartHD*) = *tHD;*

 (*StartGC*) = *tGC;*  (*StartGO*) = *tGO;*

 (*StartHR*) = *tHR;*  (*StartSLR*) = *tSLR;*

 (*StartSHC*) = *tSHC;*  (*WaitMove*) = ∞*;*

 (*WaitSHO*) = ∞*;*  (*WaitSLO*) = ∞*;*

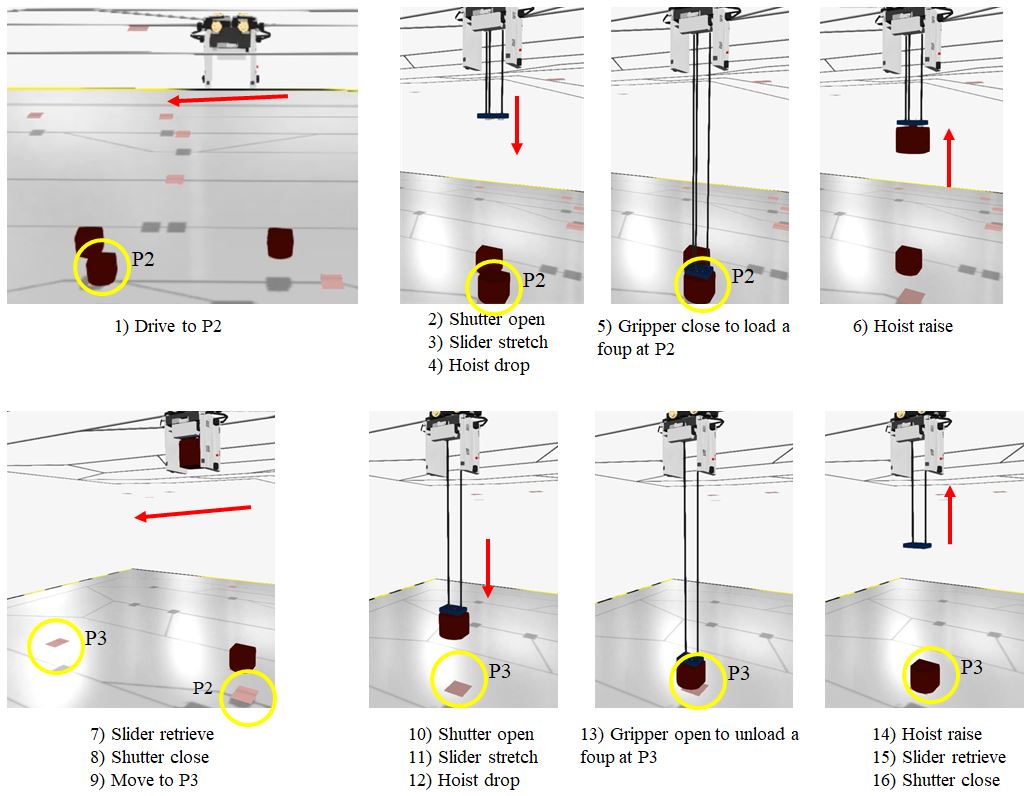
 (*WaitHD*) = ∞*;*  (*WaitGR*) = ∞*;*

 (*WaitHR*) = ∞*;*  (*WaitSLC*) = ∞*;*

 (*WaitSHC*) = ∞*;*

As mentioned earlier, the logical OHT model becomes the ‘OHT model slave’ on the EtherCAT environment (Figure 4). By doing so, we are able to make the controller (OHT controller slave) assume that it is controlling the real OHT device instead of the virtual OHT model. The prototype of the proposed virtual commissioning environment was implemented and tested with several examples. The C++ language in a Visual Studio environment was used, with OpenGL for the graphical rendering.

Figure 7 shows the prototype performing the virtual commissioning of an OHT. An OHT stays at P1, and it needs to move a FOUP from P2 to P3. The scenario consists of sequential tasks; 1) Drive to P2, 2) Shutter open, 3) Slider stretch, 4) Hoist drop, 5) Gripper close to load a FOUP at P2, 6) Hoist raise, 7) Slider retrieve, 8) Shutter close, 9) move to P3, 10) Shutter open, 11) Slider stretch, 12) Hoist drop, 13) Gripper open to unload the FOUP at P3, 14) Hoist raise, 15) Slider retrieve, and 16) Shutter close.



✁ Figure 7. Virtual commissioning of an OHT ✁

**4. DISCUSSION AND CONCLUSIONS**

In a large FAB, the travelling rail of an OHT network can extended up to a total length of 10km with up to several hundred OHTs. On the network, each OHT needs to perform the autonomous driving by recognizing its local environment interactively. Since the behavior of each OHT is controlled by its own controller, it is important to design the control software by considering various situations, such as collision avoidance, switching for branching or merging of rails, and the sudden failures of sensors and actuators.

In a conventional implementation procedure of an OHT, the mechanical and electrical design phases are performed sequentially, which is a main cause of delays in time to market. Another problem is that the conventional procedure does not include the virtual commissioning. Without virtual commissioning, an OHT will have to be stabilized solely by real commissioning, which is very expensive and time consuming. To cope with the problems, we propose a concurrent procedure of mechanical and electrical designs including the virtual commissioning phase. To achieve the concurrency, we separate a virtual OHT model into two parts, a physical model (mechanical part) and a logical model (electrical part). The mechanical engineer and the electrical engineer can do their jobs concurrently without interfering with each other, since the physical model and the logical model can be defined independently.

For the virtual commissioning, it is necessary to perform a simulation involving a ‘virtual OHT model’ and a ‘real OHT controller’, connected through a fieldbus system (EtherCAT). Then, the virtual commissioning can be performed to fix various errors caused by mechanical models not being properly adjusted, and faults in the control programs. We can minimize the stabilization time before the production phase, since most of errors are fixed through the virtual commissioning.

**ACKNOWLEDGEMENT**

**This work was supported by the technology innovation program (20002772) funded by the Ministry of Trade, Industry & Energy (MOTIE, Korea).**

**REFERENCE**

Anglani, A., Grieco, A., Pacella, M., Tolio, M. (2002). Object-oriented modeling and simulation of flexible manufacturing system: a rule-based procedure, Simulation Modeling Practice and Theory, 10, 209-234.

Chang, DS., Park, SC. (2018). Configuration space-based discrete event system specification formalism for a smart factory with real-time flexibility, Concurrent Engineering: Research and Applications, 26(3): 265-275

Drath, R., Weber, P. and Mauser, N. (2008). An evolutionary approach for the industrial introduction of virtual commissioning, *IEEE International Conference on Emerging Technologies and Factory Automation*: 5-8.

Gans, N. R., Dixon, W. E., Lind, R., Kurdila, A. (2009). A hardware in the loop simulation platform for vision-based control unmanned air vehicles, Mechatronics, 19, 1043-1056.

Ham, WK., Oh, JW., Cho, KH., Park, K., Park, SC. (2019). New modeling formalism for the energy simulation of conveyor systems, Computers & Industrial Engineering, 128, 180-191.

Hibnio, H. Inukai, T. and Fukuda, Y. (2006). Efficient manufacturing system implementation based on combination between real and virtual factory. *International Journal of Production Research*, **44** (18): 3897–3915

Hoffman, P., Maksoud, T. M. A., Schuman, R. and Premier, G.C. (2010). Virtual Commissioning of Manufacturing Systems a review and new approaches for simplification*, Proceedings 24th European Conference on Modeling and Simulation*.

Hsieh, CH., Cho, C., Yang, T., Chang, TJ. (2012). Simulation study for a proposed segmented automated material handling system design for 300-mm semiconductor fabs, Simulation Modeling Practice and Theory, 29, 18-31.

Huang, H., Yeh, C. (1999) Development of a virtual factory emulator based on three-tire architecture, *IEEE Int. Conf. On Robotics & Automation*, Detroit Michigan, 2434-2439.

Kim, J., Yu, G., Jang YJ. (2016). Semiconductor FAB layout design analysis with 300-mm FAB data: “Is minimum distance-based layout design best for semiconductor FAB design?”, Computers & Industrial Engineering, 99, 330-346.

Kim, T. G. (1994). *DEVSIM++ User’s Manual*, Department of Electrical Engineering, KAIST, Korea.

Klingstam, P., Gullander, P. (1999). Overview of simulation tools for computer-aided production engineering, Computers in Industry, 38, 173-186.

Kong, SH. (2007). Two-step simulation method for automatic material handling system of semiconductor fab, Robotics and Computer-Integrated Manufacturing, 23(4), 409-420.

Lee, CG., Park, SC. (2014). Survey on the virtual commissioning of manufacturing systems, Journal of Computational Design and Engineering, 1(3), 213-222

Park, SC. (2005). A methodology for creating a virtual FMS model, Computers in industry, 56(7), 734-746

Park, SC., Chang M. (2012). Hardware-in-the-loop simulation for a production system, International Journal of Production Research, 50(8), 2321-2330

Park, SC., Park, CM., Wang, G. (2008). A PLC programming environment based on a virtual plant, International Journal of Advanced Manufacturing Technology, 39, 1262-1270

Seo, JC., Chung, YH., Kim, BH., Park, SC. (2016). Backward capacity-filtering for electronic Fabs, Production Planning & Control, 27(11), 925-933.

Zeigler, B. P. (1984). *Multifacetted modeling and discrete event simulation*, Academic Press, Orland.