Lab-scale Models of Manufacturing Systems for Testing Real-time Simulation and Production Control Technologies

Lugaresi, G.; Alba, V. V.; Matta, A.

This is a post-peer-review, pre-copyedit version of an article published in JOURNAL OF MANUFACTURING SYSTEMS. The final authenticated version is available online at: http://dx.doi.org/10.1016/j.jmsy.2020.09.003

Lab-scale Models of Manufacturing Systems for Testing Real-time Simulation and Production Control Technologies

Giovanni Lugaresi[1], Vincenzo Valerio Alba[1], Andrea Matta*[1]

[1] *Dipartimento di Meccanica, Politecnico di Milano*
Via la Masa 1, 20156 Milan, Italy
**Corresponding Author:* andrea.matta@polimi.it

Abstract

In the last years, the increase of data availability together with enhanced computation capabilities empowered researchers to conceive production planning and control methods with real-time inputs. Literature is rich with techniques for using simulation to take production planning and control decisions online. However, it is generally impractical to test these approaches on real systems, and experiments on digital instances are limited because they do not capture the physical aspects. This work proposes to test Real-time Simulation approaches using lab-scale models of manufacturing system and a software architecture aligned with industrial standards. Such models allow to reproduce material flows and the production control logic of real factory environments. By exploiting this setting to test new approaches and tools, it is possible to increase their own achievable Technology Readiness Level (TRL). The laboratory has been used to set a real-time rescheduling problem on a Flexible Manufacturing System (FMS) model. The test involves simulation models aligned with the current system state for the online identification and implementation of a production scheduling rule that decreases the expected makespan. The results testify that the proposed lab-scale models can be used successfully to test production planning and control approaches.

Keywords: Real-time Simulation; Re-scheduling; Lab-scale Models; Flexible Manufacturing Systems.

1. Introduction

The current industrial scenario has been defined by radical changes in the methodologies used to manage and control manufacturing systems. The Industry 4.0 phenomenon provided a set of new technologies for production environments such as Internet of Things (IoT), cloud computing, big data analytics, augmented and virtual reality, Radio Frequency Identification (RFID), artificial intelligence and machine learning [1]. As a consequence, several innovative solutions have been developed, such as digital twins [2] and cyber-physical systems [3]. Thanks to the aforementioned developments in industry and research, it is possible to imagine a situation in which the shop-floor status in manufacturing companies can be retrieved anytime. Such capabilities can enhance decision-making and reaction time. Performance evaluation tools traditionally used for long term decisions (e.g., simulation) can now be exploited in the very short term [3].

An established standard for production management is the ANSI-ISA95 [4]. It is based on five hierarchical levels as shown in Figure 1. Level 0 is associated with the physical process; level 1 refers to the governance of actuators and sensors; level 2 represents the control logic and production supervision; level 3 to manufacturing operations (i.e. Manufacturing Execution System); level 4 to the management of the entire firm (i.e. Enterprise Resource Planning). The ISA95 levels are often depicted as a pyramid, although recent research claims that this hierarchical view of the production system has been unsettled by the data sharing capabilities of IoT and cloud computing [5]. The increased integration of functionalities brings benefits to production systems such as increased flexibility and the capability to promptly respond to unexpected events [6]. In fact, production policies defined a-priori may be optimal within a precise set of boundaries, but turn out to be detrimental on system performances when applied in actual scenarios because real systems are always subject to some degree of stochasticity. A possible countermeasure is to design solutions which are robust against many different scenarios, although never truly optimal for any particular condition rather for the average performance of the system. Distinctively, online decisions can be tailored exactly considering the system status in the very moment they are examined.

The automated data acquisition and communication paved the way for an exploitation of simulation for short-term decision making [7]. In Real-time Simulation (RTS) applications, data are acquired from a production plant and feed a simulation model of the system. Then, alternative scenarios are simulated and the one that leads to the best performances is applied online [8]. Alternatively, the simulation models are used as offline learning tools, by exploring the performance obtained for the input parameter values which have not been implemented in the real system yet. The literature on Real-time Simulation approaches for manufacturing systems is rich and we may expect several contributions in the future [1]. Meanwhile, it is hard for researchers and practitioners to provide a realistic environment to test their architectures and algorithms. Typically, the proposed methods would either be tested on a real system or within a digital setting. In the former case, it is necessary to allocate an entire manufacturing system and to change the already established production strategies for a considerable time. The risk is to invest a lot of resources in arranging the system to reproduce the desired behavior, rather than iterating the proposed logic. On the other hand, the theorized algorithms and policies could be compared and validated exploiting digital models of the production systems. In this case, several issues related to the physical system might be underrated. For instance, data collection devices must be designed properly to retrieve the desired information (e.g., current number of parts in a queue, machine status) and to guarantee the alignment between the real system and its digital counterpart. Further, the absence of a physical substrate impedes to use hardware suitable for the target production environment.

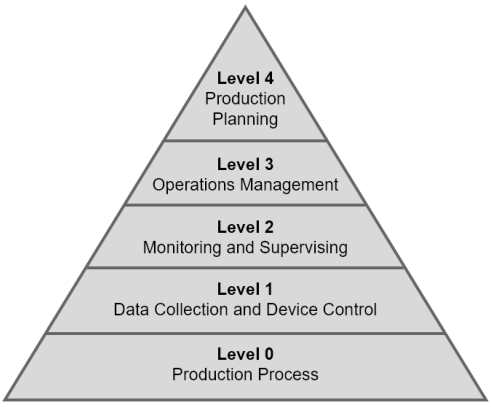


Figure 1: ISA95 levels [9].

Hence, without a fully integrated prototype the test of both individual and combined features can only be done virtually. As a consequence, it is not possible to prove the functional compatibility of the components in the intended operational system.

In this paper, we propose a lab-scale environment for testing Real-time Simulation research and digital technologies for production systems. The proposed laboratory reproduces the ISA95 architecture, hence we developed both physical and digital levels and exploited IoT-compatible devices to connect them [10]. The adoption of such models enables to examine approaches that involve information loops through real industrial components (e.g., gateways, sensors), which is not possible if validation is performed only on digital models. For instance, hardware-related issues such as data sharing between the various layers of a Cyber Physical Production System (CPPS) can be investigated, while remaining capable to implement and iterate the proposed production planning and control logic in reasonable times. Further, the realization of physical models with components such as LEGO, Arduino, fischertechnik, requires lower investments and provides higher flexibility for tests compared to similar settings in real systems. Additionally, we assess the impact of exploiting the proposed environment to test software tools, hardware devices, and production management methods with a discussion on the obtainable Technology Readiness Level (TRL).

The rest of the paper is organized as follows. Section 2 identifies the main contributions in the literature related to RTS. Section 3 outlines a general Real-time Simulation procedure for online decision making in production and section 4 presents the proposed lab-scale testing environment. Section 5 discusses on the TRL achievable by components tested in the proposed laboratory. Section 6 presents the test cases that have been done and summarizes the numerical results. Section 7 presents the numerical results. Our final remarks are in section 8.

2. Related Literature

One of the first classification methods for RTS-related research can be derived from Manivannan and Banks [11], who proposed a framework for real-time control of a manufacturing cell using simulation which can be considered as one of the first comprehensive analysis on the application of Discrete Event Simulation (DES) as short-term decision making tool. The authors have identified the **challenges** to overcome for the successful implementation of RTS models: (1) data handling, (2) model generation and validation, (3) model synchronization and initialization, and (4) efficient proactive scheduling models. Monostori et al. [3] described the **operation modes** for DES

models in production system, dividing them in three categories: (1) offline simulation, used for sensitivity analysis and robustness evaluation of production schedules before their execution, (2) proactive simulation, used with the aim of defining online short term actions in response to expected deviations from the plan, and (3) reactive simulation, used for an online evaluation of alternatives after a disturbance has occurred. Mousavi and Siervo [12] proposed a framework with the aim to provide three main **features**: (1) flexibility: the framework has to be general in terms of input data and utility function model (for example, the sample frequency can be determined dynamically); (2) real-time readiness: the framework aims to make an effective use of data available at timenow for simulation modeling; (3) fast-forwardness: it is always possible to feed the simulation model with different types of inputs to perform *what-if* analyses; historical data may also be used to gradually improve the response time of the system. The next sections elaborate on the significant contributions from the literature, which have been organized in Table 1 according to the three aforementioned classification criteria. The following sections adhere to the classification over the RTS challenges.

2.1. Data Handling

Data collection is one of the most time consuming practices at the beginning of a simulation project, and the optimization of this phase regards both static and real-time approaches. Robertson and Perera [13] proposed to automatically collect input data for a simulation model to save modeling time. The authors discussed on how to revisit the data input methodologies for simulation and proposed to group the data-input phases exploiting an intermediate database between an Enterprise Resource Planning (ERP) system and the simulation input data. Hanisch et al. [14] contemplated two characteristics for data in real-time digital models: availability and quality. Availability means that all the necessary inputs are guaranteed either by automatic collection from the system or by successive elaboration from the existing datasets. Quality regards the degree of exactness of data. Mousavi and Siervo [12] proposed a flexible data input management system as solution for quick-response decision making. The aim is to generalize the input procedure and make it applicable to a wide variety of manufacturing environments. Similarly, Blum and Schuh [15] defined a three-layer architecture for data analytics in real-time frameworks, composed by (1) a data layer, (2) an integration layer, and (3) a visualization layer. Since online decisions must be made as soon as the problems are identified in the system, RTS frameworks are based on real-time input data acquired while the real system is evolving. Kitazawa et al. [16] used RTS to estimate the completion time of a flow shop manual assembly. The data is collected by Bluetooth-based beacons to record the proximity of operators to their workstation. The beacon position is used to infer the operator current working state. Altaf et al. [17] showed the integration of simulation with RFID collected data to infer the status of a wood-frame panel prefabrication plant. Similarly, Luo, Fang, and Huang [18] introduced an approach for real-time scheduling using RFID technologies to facilitate shop-floor conditions visibility.

2.2. Model Generation and Validation

For a successful RTS implementation, the digital counterparts of production systems have to reflect the current configuration of the system at any moment in time [19]. In flexible and reconfigurable manufacturing systems these updates can be done in an automated way. Model generation deals with the recognition of the logical relationships between components of the manufacturing system (e.g., part routes, precedences, spacial contraints) [20, 21]. For instance, if an automated model generation procedure is established along the life cycle of the production plant, a plugin of

a new machining tool or the deterioration of a machine can be detected online and the simulation model can be updated coherently [22, 23]. Planning and control activities using RTS typically rely on the assumption that a simulation model for the real manufacturing system exists and has withstood the validation process [24]. Hence, if the model is generated online, validation has to be performed with respect of the data representing the current state of the system. Autovalidation is the practice through which the simulation model ability to predict the system performances is checked online before using the model to take operative decisions. Model Structure Update deals with the validation of an already available model, that is compared with the system structure by means of comparison of data and the process logical layout [25]. Recent approaches proposed to use signal processing theory for the online validation of digital models [26].

2.3. Model Synchronization and Initialization

The exploitation of a digital model of a factory beside a continuous information exchange with the real system allows to make realistic simulation-based predictions referred to the current system status [27, 28]. In general, the synchronization between the system and its digital alter can be carried out in three main ways [29]: (1) by continuously collecting data and connecting the data acquisition devices to an input data processor with the simulation software; (2) by developing a simulation model for each of the parts and resources, and restricting data collection activities to those altering temporal information; (3) by making use of past, future and current event lists. Kadar et al. [30] identified the following issues for synchronization: (1) the acquisition and validation of the input data, (2) the responsiveness of the analysis, and (3) the capability of quickly gathering the real system state to initialize the simulation model. Talkhestani et al. [31] proposed to obtain model integration in a product life-cycle management platform using an anchor-point-based method to systematically detect variations in the data structure between the digital models and the physical system. Initialization refers to guaranteeing that alternative simulation models refer to the same initial point in time. The goal is to assure that alternative production policies can be effectively compared. Namely, in order for the statistics of the alternative policies performance to be comparable to the ones of the real system, the corresponding models must be initialized to the current real system state, or at least to the same state occurred in a certain time-frame. Therefore, RTS models start from a state in which all the variables of the model are set to the values of the physical quantities at timenow [14].

2.4. Proactive Policies

Typically, the Manufacturing Execution System (MES) functions are included in the CPPS, hence also rescheduling and dispatching algorithms [5]. These tasks can be accomplished online exploiting information coming from both the shop-floor and any other management software. Recently, several frameworks exploiting RTS to improve online production policies have been proposed [32, 33, 34, 35]. A production re-planning strategy describes when a new production plan for a certain manufacturing system has to be generated. This can happen in two main forms [36, 37]: (1) the plan can be adapted periodically based on the present production trend, (2) the re-planning is triggered by specific events that can have an impact on the system performances, such as machine failures, urgent orders, quality issues [38]. Cardin and Castagna [19, 39] explored the decisional component of a job-shop with six workstations connected with transporters. The authors exploited a *base model* aligned with the real system and *variant models* for proactive decision making such as the routing of parts on transporters. These decisions depend on many factors, such as the number of available transporters in the system or the expected machine breakdowns. The multitude of

decision possibilities is reflected in the number of variant simulation models that are initialized to the system state and used to run experiments for a time horizon of interest. Harmonosky et al. [40] developed an heuristic approach to manage the queue of unfinished jobs at a failed machine in a flow-shop system. The main idea is to compare the expected waiting time between the failed machine and an alternative machine including a penalty term for accounting rerouting time. Simulations are done offline before any actual system breakdown occurs. With reference to manufacturing systems characterized by stochastic processing times, Framinan et al. [41] suggested that if real-time data measured on the shop-floor were exploited as a rescheduling contour condition, it would be possible to lower the expected makespan. This advantage is greater if the variability in system parameters is fairly low, while a high variability translates into high uncertainty of the results and may hinder the improvement. Mirdamadi et al. [42] described a procedure for exploiting simulation to determine the best production control alternatives. Events in the real system are clustered and tagged to identify the necessity and to assign priority to the interventions. Rao et al. [43] described a Real-time Simulation architecture for shop-floor control. The system can collect data from the shop-floor and communicate with a scheduling controller through the MES. It is also shown how the software infrastructure of a MES may incorporate RTS functionalities.

3. Real-time Simulation Procedure

In this section, we outline a procedure for testing Real-time Simulation applications. Let us consider a manufacturing system on which a production policy π is implemented (e.g. priority rule). Further, define $X(t)$ a vector that describes the system state at time t (e.g., current buffer levels), and T is a time horizon of interest. Let us define $\Theta(\pi, X(t), T)$ a generic Key Performance Indicator (KPI) achieved in the time period $[t, T]$ by implementing π on a production system that is in state $X(t)$ at time t.

In general, we can consider a moment $t_e \in [0, T]$ in which a simulation-optimization cycle is launched to determine the production policy to be implemented in the remaining part of the production planning horizon $[t_e, T]$. For instance, t_e may define the instant in which a disruption occurs (e.g., machine failure). Indeed, disruptive modifications may detriment system performances, and the current production policy might not be optimal any further. Therefore, an evaluation is needed to search for a better reaction strategy. Define Π a finite set of $N + 1$ policies $\{\pi_0, \pi_1, \ldots, \pi_N\}$ defined a-priori: π_0 is the original policy implemented on the system and the remaining N policies define alternative reaction scenarios. Further, assume that a discrete event simulation model of the manufacturing system is available and validated. The simulation model is a digital instance of the manufacturing system and represents the *base model*. At t_e, the digital model is synchronized with the physical system status $X(t_e)$. The performances obtained by each policy are evaluated with digital models derived from the base simulation model. Namely, N *variant models* are created, each implementing the i-th alternative policy π_i. All simulation experiments start from the same time and system state $X(t_e)$. The solution corresponds to the new optimal system management policy π^*, which satisfies:

$$\pi^* = \arg\max_{\pi_i \in \Pi} \{\Theta(\pi_i, X(t), T)\}. \tag{1}$$

Notice that $\pi^* = \pi_0$ is allowed, hence this procedure contemplates also the *do-nothing* possibility. Finally, π^* is implemented in the physical system at the time $t'_e \geq t_e$ and the interval $[t_e, t'_e]$

Reference		Data Handling	Generation and Validation	Initialization and Synchronization	Proactive Policies	Offline	Proactive	Reactive	Flexibility	Real-time Readiness	Fast-forwardness
Altaf et al.	[17]	•					•			•	
Aydt et al.	[44]				•		•		•		
Bergmann, Stelzer, and Strassburger	[45]			•			•	•		•	
Biesinger et al.	[20]		•			•			•		
Blum and Schuh	[15]	•					•			•	
Bohlmann et al.	[38]				•			•			•
Cardin and Castagna	[39]				•		•			•	
Cardin and Castagna	[19]			•	•		•			•	
Damiani et al.	[32]				•	•			•		
Framinan et al.	[41]	•			•		•			•	
Fujihara and Yoneda	[23]		•	•		•			•		
Hanisch, Tolujew, and Schulze	[14]	•	•	•			•	•			•
Harmonosky, Farr, and Ni	[40]				•	•			•	•	
Kadar et al.	[30]		•	•	•			•		•	
Katz and Manivannan	[27]			•				•		•	
Khan et al.	[25]		•			•			•		
Kitazawa et al.	[16]	•			•	•					•
Low et al.	[46]	•						•			•
Lugaresi et al.	[26]		•				•			•	
Luo, Fang and Huang	[18]	•			•			•	•	•	
Manivannan and Banks	[11]	•		•	•	•				•	
Martinez et al.	[28]		•	•			•			•	
Martinez et al.	[22]		•			•	•	•		•	
Mirdamadi, Fontanili and Dupont	[42]	•	•	•	•			•	•		
Mousavi and Siervo	[12]	•					•				•
Nasiri, Yazdanparast and Jolai	[33]	•						•			•
Pfeiffer et al.	[34]				•			•		•	
Rao et al.	[43]	•						•			•
Robertson and Perera	[13]	•				•			•		
Son and Wysk	[21]	•	•			•			•		
Suresh, Wassick, and Ferrio	[35]				•		•			•	
Talkhestani et al.	[31]			•		•				•	
		Challenges				**Operation Modes**			**Features**		

Table 1: RTS literature items classified according to the three criteria: challenges, operation modes, features.

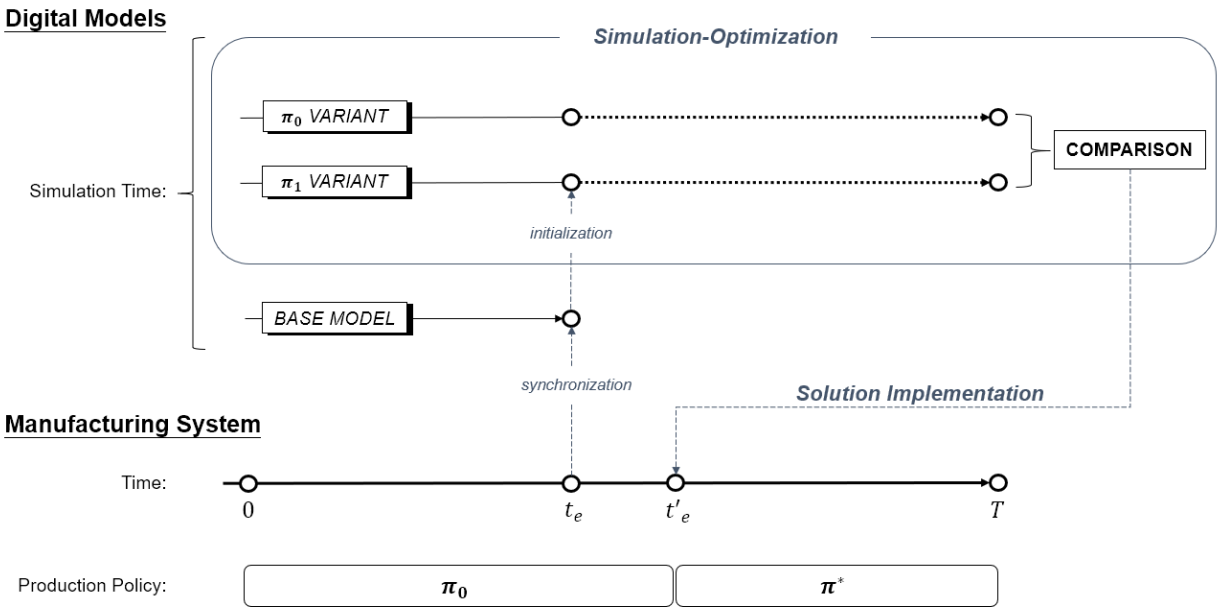


Figure 2: Real-time Simulation procedure illustration with two alternative production policies.

is proportional to the computation effort required to identify the new policy. Figure 2 summarizes the temporal evolution of the procedure.

4. Proposed Models and Software Architecture

The proposed lab-scale models are part of a cyber-physical architecture with four hierarchical levels which is shown in Figure 3. The first level is a physical model of a production system built with structural components, sensors, actuators, and Programmable Logic Controllers (PLC). The *execution level* controls the PLCs and converts the sensor outputs into structured data to be sent to the *logic level*; it also receives and releases the motors execution commands. The *logic level* is related to the functions of monitoring and supervising the process and the MES services such as the execution of production orders. The fourth level can be composed by several tools such as simulation-optimization or modules dedicated to production management (e.g. ERP). The communication among the different levels is done by using Internet of Things standards, thus guaranteeing the connection between the physical and digital instances. Hence, the proposed laboratory is effectively a CPPS on a smaller scale.

4.1. Physical System

The physical components include both structural pieces such as beams, shafts, conveyor belts, and actuators, sensors and PLCs. The assembled models can be used to replicate the behavior of a real production line by moving parts such as spheres or discs along a proper route and reflect operation times by letting parts wait in a station for an appropriate time span. Exploiting physical system models guarantees several advantages: (1) *high flexibility*, since it is possible to build several kinds of production systems. (2) *facilitated development*, because it is relatively easy to develop a model and the whole assembly process does not require any particular tool. Indeed,

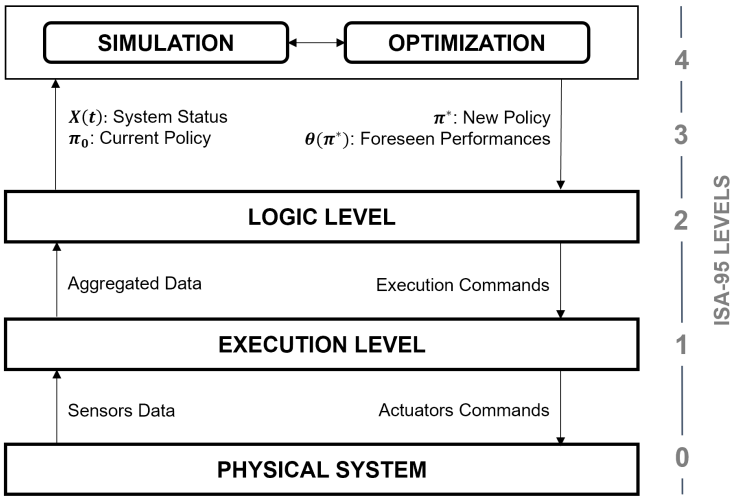


Figure 3: The developed architecture with reference to the ISA95 levels.

small models can be built in a few hours by a single person while the most complex ones can generally be completed within days. (3) *easy management*: it is undoubtedly easier to manage a small scale model with respect to a real production system. The models are usually built in a modular fashion in order to be easily disassembled, transported, and re-assembled. (4) *limited capital expenditure and re-usability*: the cost of the models is orders of magnitude lower than the investments required in a real system. Further, almost 100% of components can be reused after a project completion. Despite the aforementioned advantages, most applications of lab-scale models in industrial engineering focus on educational purposes [47, 48]. The physical models proposed in this work have been built with LEGO MINDSTORMS. Next, the common components of the proposed lab-scale manufacturing system models are presented:

- **Parts** are modeled by wooden discs ($\varnothing 25mm$) marked with a color plate. The colors are used to represent different part types and are recognized by the sensors along the system for assigning the right setup and processing times.
- **Conveyors** are controlled by dedicated electrical motors and compose the transportation system that moves the parts in the system. Each conveyor can be set to run at a specific speed which can be changed at runtime.
- **Buffers** are represented by the conveyors which bring parts from a station to another. It is possible to define a specific buffer size through the position of the downstream sensor of each station (sensor 3 in Figure 4a), while the maximum buffer size is superiorly limited by the length of the conveyor. Special types of buffers can be modeled as well. For instance, Figure 7 shows a model with a three-slide buffer system, in which each part type is stored in a dedicated slide.
- **Stations** are represented by dedicated areas which hold parts for an amount of time that mimics the setup and processing operations on the parts as well as production disruptions such as failures. A station can be in either one among three states: (1) working, (2) idle, and (3) blocked. Figure 4a shows an example of a station built with LEGO. A station is

composed by an EV3 brick, three EV3 optical sensors, a part-entrance system and a motor. The part-entrance system is in front of each station. A beam is driven by Motor 1 and blocks the parts in front of the station to avoid the entrance of more than one part at a time. Figure 4b summarizes the workflow of the station model. Sensor 1 lies over the part-entrance system to recognize if a part is waiting to be worked. When the station is idle and a part is available, the part-entrance system pushes the part inside the station. Motor 2 drives the part inside the station. Sensor 2 is placed in the middle of the station structure to check if a pallet has entered the machine and to distinguish the product type. As soon as the part has entered, Motor 2 is stopped and the station is set to working state. Sensor 3 is installed over the conveyor on the downstream conveyor and determines if the downstream buffer is full. When the operation is done, if there is enough space on the downstream conveyor, Motor 2 downloads the part and the station is set to idle state. On the other hand, the station is set to blocked state while the downstream buffer is full. The station model exploits three optical sensors to control the part flows.

- **Programmable Logic Controllers.** Each station is controlled by an EV3 device. In this work, EV3DEV OS has been used [49]. This open-source operating system is based on Debian Linux and allows the execution of *python* scripts[1] for controlling the sensors and motors through dedicated libraries. Each EV3 is assigned an IP address in a local network and can communicate with a centralized controller. The execution level software is explained in section 4.2.

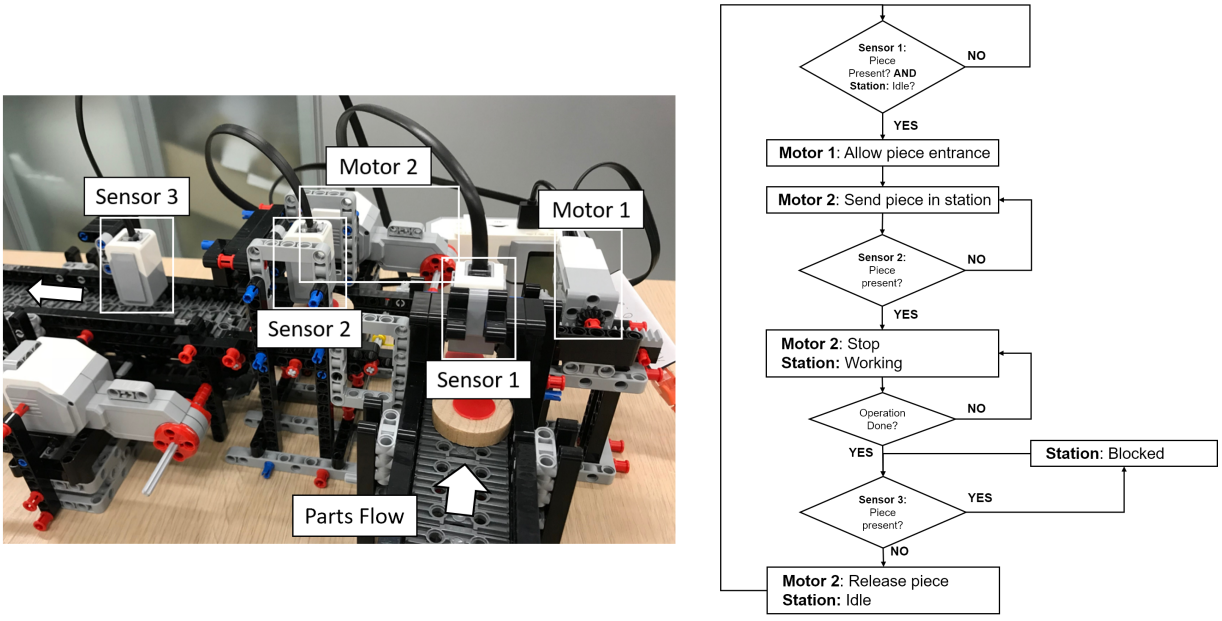
Further details on LEGO-based production system models can be found in related works [50].

4.2. Execution Level

This level represents the software running on the PLCs that controls the physical devices. In this work, the execution level consists in a script which runs on each EV3 brick. The execution level is responsible for (1) releasing start/stop commands to the motors whenever required and (2) acquiring and sharing the sensor outputs. The motors activation is triggered by specific messages communicating the desired actions. The sensor outputs can be conveyed in either two modes: (1) *on-demand*, namely required by a higher hierarchical level or (2) *on-change*, hence triggered by specific events. The code of the execution level is object-oriented. Specifically, three classes correspond to the three main physical devices in the system: the EV3s, the motors, and the sensors. Each class contains an attribute which is a list of all the relative instantiated objects. The classes that have been developed (Figure 5a) are the following:

- The **Ev3** class represents the logic controller. All the motors and sensors executed by the controller are listed in the *peripherals_list* and are contextually instantiated at startup.
- The **Motor** class has the attributes *name* and *ev3motor*. *ev3motor* is an object available in the EV3DEV library that allows for interfacing with the motors through *python* commands.
- The **Sensor** class has the attributes *name*, *ev3sensor* and *color_seen*. *ev3sensor* is an object available in the EV3DEV library that allows controlling the EV3 sensors. *color_seen* is an attribute that indicates the last color identified by the sensor.

[1]The choice of *python* as programming language is not restrictive and the proposed architecture can be extended to other languages.



(a) (b)

Figure 4: Example of station model: (a) physical model components, (b) logical workflow.

4.3. Logic Level

This level manages production rules, handles constraints and determines the characteristics of the manufacturing system (e.g., station workflow as in Figure 4b). At startup, the logic level sends a configuration message to the EV3s (*ev3_config*) containing a description of the physical system logical layout (i.e. the *peripherals_list*). Further, the logic level contains the messages definition for communicating with the execution level and for exporting significant data towards other management services (e.g., time-series database with the sensor data). In this work, the logic level is a *python* script running on a central controller (e.g., an industrial PC). The code is object-oriented and consists in the following classes (Figure 5b). Each class contains an attribute which is a list of all the relative instantiated objects.

- The **Motor** class has two attributes: *name* is the motor identifier and *ev3* is the name of the EV3 device that controls the motor. Motor instances also possess methods corresponding to the executable actions. Whenever one of these methods is called, a corresponding message requesting the motor activation is published on the network to be processed by the execution level (section 4.5).

- The **Sensor** class contains the attributes *name* and *output*, where the latter is a dictionary with the indication of the sensor name and information read by the sensor. Moreover, the Sensor class possesses the *read* method that can be used for reading sensor output (*on-demand*).

- The **ColorSensor** class is a subclass of the Sensor class, with the addition of an instance attribute *ev3* representing the name of the EV3 that is connected to the sensor.

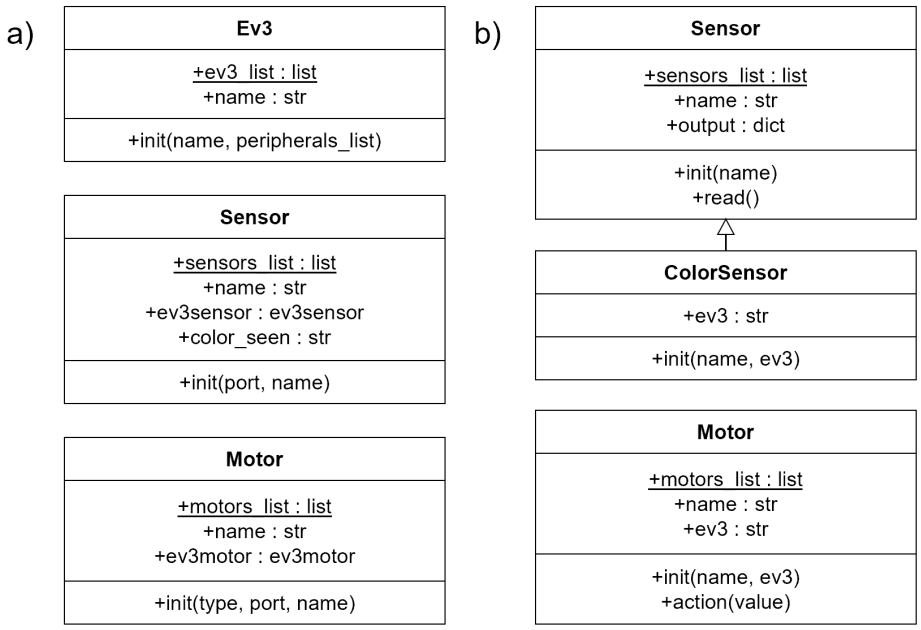


Figure 5: The developed classes for (a) the execution level and (b) the logic level.

4.4. Fourth Level

The fourth level of the proposed cyber-physical architecture includes software components exploiting the data from the system to perform several high-level operations. For instance, simulation can be exploited for building digital twins of the physical models. The developed architecture allows to communicate the data measured on the shop-floor, hence it is able to infer the system status and use it as initial condition for simulation models. It is thus possible to simulate different production and management policies in order to determine which one is optimal.

In this work, a digital model of the manufacturing system has been built in Simulink Simevents. The synchronization of the base simulation model is done through *csv* files that contain all the information regarding the production plan and the system status. Specifically, three files describe the nominal processing times, the setup times, and the initial production schedule, respectively. Further, the current system status is represented by two files containing: (1) the job currently under process by each machine and the remaining time until each machine is expected to be in idle state, and (2) the effective production schedule to be followed, which is read each time a machine is idle. The files are shared between the logic level and the simulation model. Hence, the system status can be updated continuously during production.

It is worth to notice that other software components can be added at this level. For instance, data flows management tools are intermediary services to transfer data between two utilities. Databases allow the storage, manipulation and query of the acquired data for obtaining useful information: for example, the time series of a machine state may be used to derive both availability and reliability indicators. Dashboards are applications for the real-time visualization of both raw data and custom indicators and indexes. Cloud computing components enable the interface with software tools such as ERP.

Message Topic	Source	Destination	Purpose
data	Logic Level	Fourth Level	Extract specific data
ev3_config	Logic Level	Execution Level	EV3s configuration
logic_config	Execution Level	Logic Level	EV3s configuration
sensor/request	Logic Level	Execution Level	Sensor output (*on-demand*)
sensor/on_demand	Execution Level	Logic Level	Sensor output (*on-demand*)
sensor/on_change	Execution Level	Logic level	Sensor output (*on-change*)
motor/action/action_name	Logic Level	Execution Level	Activate motors
stop	Any	Execution/Logic Level	Stop software execution

Table 2: Summary of the messages exchanged across different levels of the developed architecture.

4.5. Communication Protocol

The communication between the software levels is possible thanks to an IoT infrastructure based on the Message Queue Telemetry Transfer (MQTT) protocol. This allows the PLCs to send and receive messages to any kind of IoT-compatible device connected to the network. Hence, it is possible to share and store data from the real system and the architecture levels exploiting the message-based communication protocol. Table 2 summarizes the messages exchanged. The messages are written in the JavaScript Object Notation (JSON) format. Following we list two significant examples. Messages from the *sensor/request* topic are sent from the logic level to the execution level and they contain the request to read a specific sensor output. In this case, the message contains the name of the sensor and the EV3 device which is controlling it. Messages of the topic *motor/action* are sent from the logic level to the execution level and contain the actions that should be executed by the motors. In the developed physical models the possible actions are the following: start moving at a certain speed, run for a certain amount of time at a certain speed, turn the axis to a specific angle value, run back-and-forth of a specific angle value, stop. Notice that other actions can be designed accordingly to specific system requirements. The message content is a JSON object containing the motor name, the name of the EV3 that controls it and a value that describes how to execute the prescribed action.

5. Assessment of the Technology Readiness Level

Let us consider the case in which a new RTS-based technology has to be tested. It is reasonable to reflect on the advantages of exploiting a lab-scale environment such as the one described in section 4.1. An established way to assess the maturity of a technology is the Technology Readiness Level (TRL). The TRL is an nine-level indicator originally developed by NASA [51] which has been used extensively in the last 40 years to assess the maturity of a technology in the aerospace sector [52]. Nowadays, the TRL is also used to assign grants and evaluate research proposals such as European Horizon 2020 [53]. It is assumed that a TRL level cannot be reached before the previous levels are obtained. The first three levels refer to the observation of basic principles, the formulation of technology concepts and proof-of-concepts. TRL 4 is achieved if the proposed technology is validated in a laboratory. In such a setting, the operating environment is not realistic. For instance, consider load cells tested with lower weights or an actuator used to provide smaller displacements than in the intended use cases. Since the proposed lab-scale models are compatible with the ISA95 industrial standard and can incorporate IoT components, we infer that the proposed environment allows specific component types to advance their own TRL to a level higher than 4.

In order to clarify our assumption, we have exploited a questionnaire developed by the US Air Force Research Laboratory (AFRL) [54]. The questionnaire consists in 274 questions. If a newly-developed technology can satisfy all the questions related to a TRL, it can be considered at that readiness level. Table 3 summarizes the questions that we believe could be answered positively by using the lab-scale models. Specifically, we selected three types of components that could be evaluated exploiting the proposed testing environment: (1) a software component, (2) a hardware device (e.g., gateway, sensor, PLC), and (3) a method (e.g., a scheduling algorithm). Depending on the component type, we have identified the achievable TRL level. As a result, the lab-scale setting can grant each tested component to advance on its own TRL. It is worth to notice that this analysis cannot assess the TRL of the entire manufacturing system.

5.1. Software Component

If the software component is in the loop with the lab-scale model, the laboratory can be considered a relevant environment. Indeed, if the actuators and sensors can provide the same functionality of the real system (e.g., changing routes, stopping part flows) it is possible to reproduce material flows representative of a real system behavior. If the steady state performance of the model is comparable with a real system, it is possible to design tests to address specific factory requirements such as quality control frequency or production rate. Further, the hardware processors for the lab-scale environment can be the same one as the real environment (e.g., gateways and PLCs), with no specific limitations concerning the integration among software tool components. The interfaces can be described with reference to real components. For example, the cyber-physical architecture can incorporate the same data formats required by a specific PLC model. Moreover, the whole architecture can be verified through the established communication channels, for instance by testing the conformance to priority rules among different hierarchical levels.

5.2. Hardware Component

In case of a hardware component such as a PLC or gateway, if the test scope is production planning and control and the implemented functionalities of the lab-scale model correspond to large scale systems (e.g., flow control) the laboratory setting can be considered a field environment equivalent. Also, anomalous conditions can be designed properly in the lab-scale model to be representative of a realistic situation. For instance, an anomalous flow in a real plant could be simply recorded and replicated in the lab-scale model. The hardware can be tested through the connection between the lab-scale architecture levels and the field devices (e.g., serial ports). Additionally, interfaces can be tested on the lab scale models the same way as in a realistic environment (e.g., through dashboard visualizations and database connections).

5.3. Method

In case the intended technology is a method for production planning and control (e.g. scheduling algorithm), if the stream of parts replicated in the model is realistic (e.g., steady state performances are comparable) we may consider the lab-scale environment as representative of a field environment. Indeed, the algorithm uses data coming from the field sources as inputs and outputs, regardless of the production system mechanics. Hence, if the installed devices are representative of the production system of interest, logical functions can be tested on the lab-scale models with the same expected outputs. For instance, a scheduling algorithm can be tested against realistic disruptions replicated in the physical model.

Method	TRL	Question
Software	5	System software architecture established
		External interfaces described as to source, format, structure, content, and method of support
		Interfaces between components/subsystems are realistic (Breadboard with realistic interfaces)
		High fidelity lab integration of system completed, ready for test in simulated environments
		Some special purpose components combined with available laboratory components
		Laboratory environment modified to approximate operational environment
		Individual functions tested to verify that they work
		Individual modules and functions tested for bugs
		Integration of modules/functions demonstrated in a laboratory environment
		Algorithms run on processor with characteristics representative of target environment
	6	Factory acceptance testing of laboratory system in laboratory setting
		Representative model/prototype tested in high-fidelity lab/simulated operational environment
		Realistic environment outside the lab, but not the eventual operating environment
		Prototype implementation includes functionality to handle large scale realistic problems
		Algorithms partially integrated with existing hardware / software systems
		Individual modules tested to verify that the module components (functions) work together
		Components are functionally compatible with operational system
		Representative software system or prototype demonstrated in a laboratory environment
		Laboratory system is high-fidelity functional prototype of operational system
		Integration demonstrations have been completed
		Production demonstrations are complete
Hardware	7	Materials and manufacturing process and procedures initially demonstrated
		Each system/software interface tested individually under stressed and anomalous conditions
		Algorithms run on processor(s) in operating environment
		Most functionality available for demonstration in simulated operational environment
		Operational/flight testing of laboratory system in representational environment
		Fully integrated prototype demonstrated in actual or simulated operational environment
		System prototype successfully tested in a field environment.
	8	Components are form, fit, and function compatible with operational system
		Form, fit, and function demonstrated in eventual platform/weapon system
		All functionality demonstrated in simulated operational environment
		System qualified through test and evaluation on actual platform (DT&E completed)

Table 3: AFRL questions that can be satisfied by the proposed lab-scale models.

6. Case Study: Online Re-scheduling

This section presents the case study designed to test a real-time production planning method exploiting the proposed lab-scale physical models. The case study refers to a Flexible Manufacturing System (FMS) with parallel machines producing different product types. The system has been chosen with the intent to address a significant level of complexity while maintaining a size that facilitates the understanding of the obtained results.

6.1. Manufacturing System

Let us refer to an FMS with parallel machines $m \in \mathbb{M}$. The system has to produce a set of jobs $k \in \mathbb{K}$ belonging to part types $j \in \mathbb{J}$ within an expected time horizon T. Let us accept the short notation $j(k)$ to indicate the part type to which the k-th job belongs. At any time $t \in [0, T]$, the production schedule $\sigma_m(t) = \{j_1(k), \ldots, j_{N_m}(k)\}$ is defined as the sequence of N_m jobs to be produced on the m-th machine from time t until completion. For instance, $\sigma_1(0) = \{1, 1, 2\}$ means that at time $t = 0$ on machine $m = 1$ three jobs are scheduled: the first two jobs belong to part type 1 and are followed by a job of part type 2. The machines are unreliable and can be subject to failures at any time. Let us define three random variables: P_{jm} is the time to process a part of type j on machine m, S_{ijm} is the setup time to switch from producing a job of part type i to a job of type j on machine m, and F_{jm} is the downtime that may occur during the processing of part type j on machine m. Hence, let $\tilde{P}_{jk}$, $\tilde{S}_{ijm}$, $\tilde{F}_{jm}$ indicate the effective processing time, setup time, and downtime, respectively. The makespan is the time in which all the products in the production schedule have been completed. It can be written as follows:

$$C_{max} = \max_{m} \left\{ \sum_{k \in \sigma_m} \left(\tilde{P}_{j(k),m} + \tilde{S}_{j(k)-1,j(k),m} + \tilde{F}_{j(k)m} \right) \right\}. \tag{2}$$

The production schedule is determined exploiting the *predictable schedule* concept proposed by Arnaout [55], which is based on the idea that a robust schedule should contain adequate safety times to account for the expected disruptive events along the production. The safety time is proportional to the expected failure rate of the machines and the production activities duration. For example, if a machine is expected to spend a tenth of the available time in downtime, a predictable schedule would include one unit of safety time every ten time units of scheduled production activity. Let us call $\hat{P}_{jm}$ the expected time to process jobs of part type j on machine m, and $\hat{S}_{ijm}$ is the expected setup time to switch from producing jobs of type i to type j on machine m. The safety time to be accounted for each k-th job scheduled on machine m can be estimated with equation (3).

$$ST_{km} = R_m \delta_m (\hat{P}_{j(k)m} + \hat{S}_{j(k)-1,j(k),m}) \left(1 - \frac{PO_{km}}{N_m}\right) \quad \forall\, k \in \sigma_m, m \in \mathbb{M} \tag{3}$$

where R_m is the Mean Time To Repair (MTTR) on machine m, δ_m is the estimated number of breakdowns on machine m per unit time, PO_{km} is the k-th job position in the schedule of the m-th machine, and N_m is the total number of jobs that are scheduled on the m-th machine. Figure 6 shows an example of schedule including the variables exploited in equation (3). Once an initial schedule is set, the system starts to produce pieces. If no failures occur before a scheduled safety time, the latter is removed from the schedule and the next programmed job is anticipated. At any moment t, each machine m has to produce the remaining fraction of its schedule, $\sigma_m(t)$. The Remaining Safety Time (RST) of a machine m at time t is defined as $RST_m(t) = \sum_{k \in \sigma_m(t)} ST_{km}$.

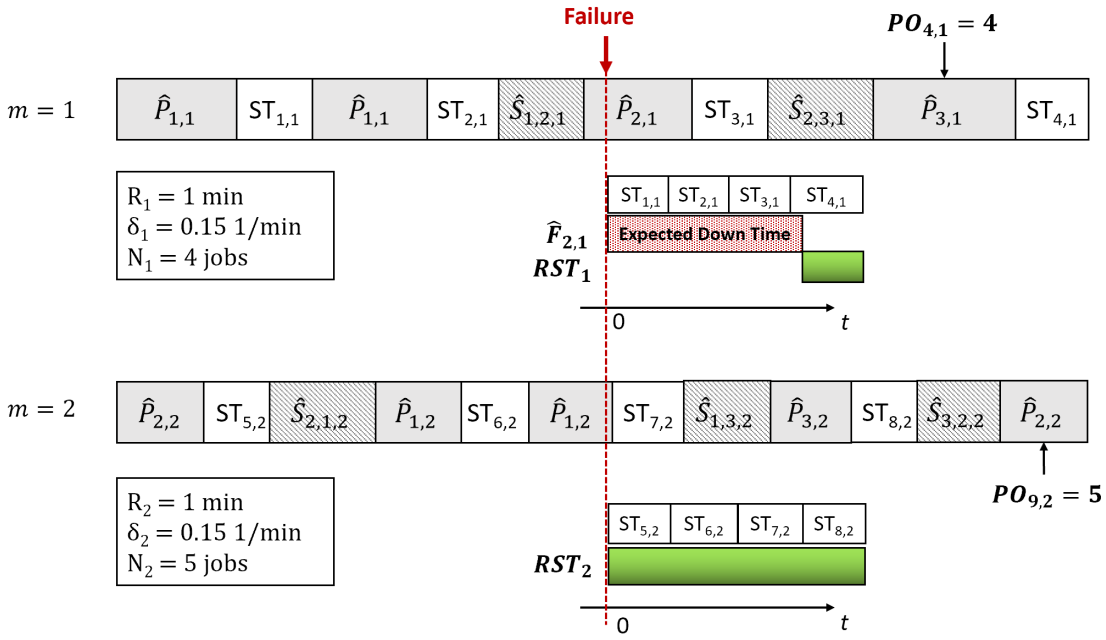


Figure 6: Predictable schedule example on two machines with three part types.

Whenever an unexpected event occurs on a machine, the corresponding RST is diminished by the expected failure time, $\hat{F}_{jm}$.

6.2. Rescheduling Problem

Consider the situation in which a failure occurs while a machine is working a part. Typically, the incomplete job is simply rescheduled as the next one to be produced on the failed machine as soon as it returns available. Alternatively, it is possible to trigger an RTS procedure (section 3) in order to evaluate online if a better solution can be found. Let us introduce two policies based on the application of either one of the following reaction rules.

- **Base Policy π_0: Right Shift Repair (RSR)**. The job is rescheduled on the same machine, right after the failure has been resolved. The production schedules on the other machines do not undergo any modification.
- **Alternative Policy π_1: Modified Fit Job Repair (MFJR)**. The Fit Job Repair (FJR) rule has been proposed by Arnaout [55] and prescribes that a job which is unfinished due to a failure at time t_e has to be assigned to the machine with the highest RST. We introduce the Modified FJT (MFJT) which establishes that on the machine chosen by the FJT rule, the job is to be rescheduled in a position that also minimizes the expected setup time.

For example, consider the case in which machine $m = 1$ fails at $t_e = 100s$ and a job of type $j = 2$ has to be rescheduled (Figure 6). According to RSR rule, the job will be rescheduled on the same machine just after the end of the downtime, whilst the MFJR rule would reschedule it on $m = 2$, because it is the one with the highest RST. Further, since the schedule on the second machine contains jobs of type $j = 2$, the rescheduled job will be programmed after any job so to guarantee no additional setup time.

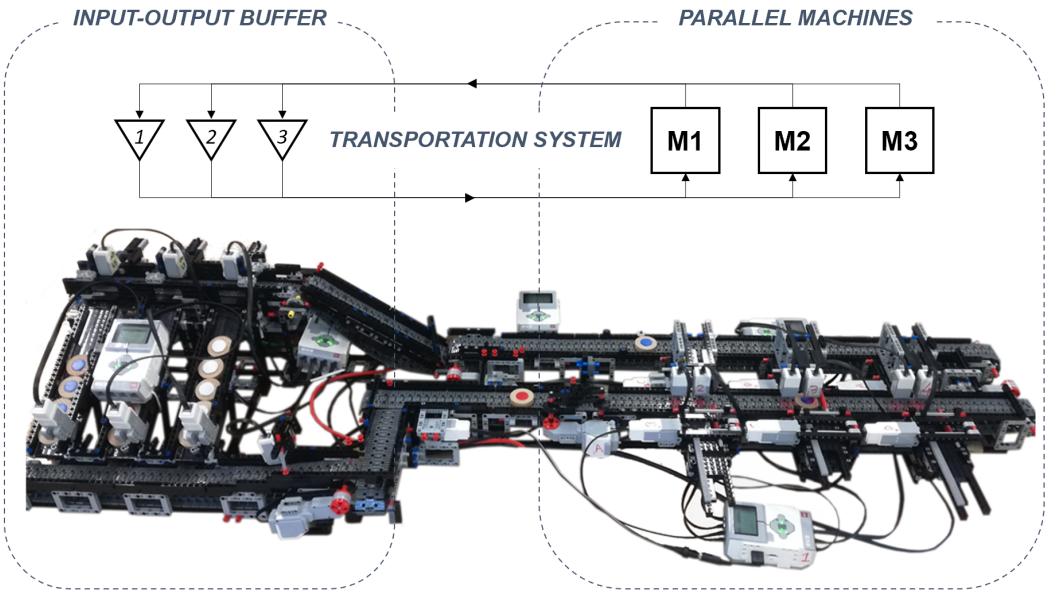


Figure 7: FMS model built with LEGO MINDSTORMS.

6.3. Lab-scale Model

In this case study, we refer to the production planning on an FMS composed by three non-identical parallel machines ($|\mathbb{M}| = 3$). 21 jobs of three part types have to be produced ($|\mathbb{J}| = 3$). The part type is modeled by three different wooden discs colors: blue, red, and white. Each part type can be worked by any of the three machines. The FMS physical model has been built with LEGO MINDSTORM components. Figure 7 shows the developed model together with the material flow. The input-output buffer can host 21 parts, 7 per type. In this buffer, parts are hosted in three dedicated sliders and can be released into the system in any sequence.

The initial schedule is shown in Table 4. At startup, the initial position of all the discs is in the input-output buffer. The buffer releases the discs into the system according to the production schedule. A job is released as soon as one of the machines is idle. If all the machines are busy, the buffer does not release jobs. The processing and setup times follow uniform distributions as indicated in Table 5, and are assigned each time a part enters a machine. Namely, each machine holds a disc for a duration equal to the corresponding sampled production time, and – similarly – it remains idle during setup times. At the end of the assigned processing time, each machine releases the disc on the downstream conveyor and returns to the idle state. In the event of failures, the downtime is modeled as a processing time. Specifically, the involved disc remains in the machine for a duration equal to the downtime before it is sent back to the input-output buffer. The corresponding job is considered as unfinished. The choice of uniform distributions is in accordance with Arnaout [56], who reminds that high variances assure disadvantageous conditions for testing scheduling algorithms.

6.4. Experimental Setting

In order to prove the effectiveness of online rescheduling, we have performed experiments based on the application of the RTS procedure presented in section 3. We have exploited a discrete event simulation model built in Simulink Simevents and synchronized with the system as *base model*.

Machine: m	**Initial Schedule: $\sigma_m(0)$**						
1	1	1	2	2	2	3	3
2	3	3	3	1	1	2	2
3	2	2	3	3	1	1	1

Table 4: Parameters of the FMS model – Initial schedule.

		Processing time [s]		
Part Type j:		$P_{j,1}$	$P_{j,2}$	$P_{j,3}$
1		UNIF(10,12)	UNIF(12,18)	UNIF(5,9)
2		UNIF(12,14)	UNIF(14,20)	UNIF(6,10)
3		UNIF(14,16)	UNIF(16,22)	UNIF(7,11)

		Setup time: S_{ij1} [s]		
	To j:	1	2	3
	1	0	UNIF(28,32)	UNIF(24,32)
From i:	2	UNIF(24,40)	0	UNIF(20,28)
	3	UNIF(28,36)	UNIF(30,32)	0

		Setup time: S_{ij2} [s]		
	To j:	1	2	3
	1	0	UNIF(24,40)	UNIF(20,32)
From i:	2	UNIF(30,40)	0	UNIF(20,32)
	3	UNIF(30,40)	UNIF(24,40)	0

		Setup time: S_{ij3} [s]		
	To j:	1	2	3
	1	0	UNIF(30,40)	UNIF(26,32)
From i:	2	UNIF(36,44)	0	UNIF(28,32)
	3	UNIF(40,48)	UNIF(32,40)	0

Table 5: Parameters of the FMS model – Processing and Setup times.

The physical system is set to produce according to the initial production schedule. If the schedule is modified during production, the system will produce part types accordingly until the production plan is completed. We have designed two specific cases in which a rescheduling activity is needed during production.

- **Case A: failure at deterministic time.** In this case, machine $m = 1$ fails one time at $t_e = 100s$ for a downtime of $60\,s$. This single scenario is replicated ten times, i.e. the lab-scale physical system is deployed in 10 independent experiments.
- **Case B: failure at stochastic time.** In this setting, three different failure scenarios are considered, one for each machine. In each scenario, one failure happens at a time t_e, which is sampled from an exponential distribution with mean $360s$. The failure duration is $70\,s$. Each scenario is replicated three times and the replications are independent.

Rescheduling is triggered by machine failures. The scheduler is a software at the fourth level of the architecture (section 4.4) and consists of a MATLAB script that controls the Real-time Simulation procedure and communicates the updated schedule to the logic level. Namely, when a failure occurs on a machine, the MATLAB script initializes the *base* simulation model to the system status at the failure moment t_e. Then, two *variants* of this model are used to simulate the production following the schedules generated by both the reaction rules RSR and MFJR (section 6.2). For each rule in each scenario, the simulations are replicated three times in order to account for the noise of machine behavior. If either one of the rules has obtained significantly better results than the other in terms of obtained makespan, the corresponding schedule is executed in the system. The implementation is done by modifying the production schedule files. Let us define $C_{max}^{(S)}(\pi_i)$ the makespan obtained in a variant simulation model of the system by applying the production policy π_i. Hence, the optimal reaction rule which will be applied in the real system satisfies the following:

$$\pi^* = \arg \min_{\pi_i \in \{RSR, MFJR\}} \{C_{max}^{(S)}(\pi_i)\}. \tag{4}$$

The effective makespan measured on the lab-scale physical system is $C_{max}^{(L)}(\pi^*)$. Hence, the duration of each run is the effective makespan. All the experiments have been done using a laptop with a 1.60GHz CPU and 8.00GB memory. Section 7 presents the numerical results.

7. Numerical Results

The proposed lab-scale model and the related architecture have been used to assess the advantage of the rescheduling approach described in section 6.2 in the two cases listed in section 6.4. In order to assess the successful implementation of the rescheduling optimal policy, a full factorial design has been performed to compare the obtained results in terms of makespan C_{max} with the following factors:

- *Rescheduling* is a two-level factor that describes the following conditions: (1) Rescheduling OFF: rescheduling is not allowed and if a failure occurs the base policy is always applied (i.e. RSR). Hence, the makespan in this case is always $C_{max}^{(L)}(RSR)$. (2) Rescheduling ON: both the RSR and the MFJR rules can be applied. In accordance with the RTS procedure,

			Simulation (S)					Lab-scale model (L)	
Resch.	**Case**	**Scenario**	$\bar{C}^{(S)}_{max}$(RSR)	95% C.I.	$\bar{C}^{(S)}_{max}$(MFJR)	95% C.I.	π^*	$\bar{C}^{(S)}_{max}(\pi^*)$	95% C.I.
ON	A	-	404.9	(403.7, 406.2)	**363.2**	(362.3, 364.2)	MFJR	364.4	(362.5, 366.3)
	B	1	404.8	(400.8, 408.7)	**379.3**	(376.7, 381.9)	MFJR	373.0	(367.0, 379.1)
		2	**370.3**	(368.8, 371.7)	375.7	(373.2, 378.3)	RSR	368.6	(365.0, 372.2)
		3	449.2	(447.2, 451.1)	**407.3**	(404.9, 409.7)	MFJR	411.3	(398.0, 424.5)
OFF	A	-	404.9	(403.7, 406.1)	363.2	(362.3, 364.2)	RSR	407.3	(404.9, 409.7)
	B	1	403.5	(399.2, 407.9)	372.4	(368.1, 376.7)	RSR	407.1	(403.5, 410.8)
		2	370.3	(368.8, 371.7)	375.7	(373.2, 378.3)	RSR	368.6	(365.0, 372.2)
		3	449.5	(446.7, 452.3)	406.7	(404.4, 409.1)	RSR	454.8	(446.1, 463.6)

Table 6: Cases A and B – Comparison between the makespan foreseen by the simulation model (S) and the one measured on the lab-scale model (L).

Factor	**DF**	**H-Value**	**P-Value**
Evaluation	1	0.9	0.344
Rescheduling	1	29.27	0.000

Table 7: Case A – Kruscall-Wallis test results on the two factors.

the reaction rule obtaining the lowest makespan in the digital model is applied online in the physical system.

- *Evaluation tool* is a factor that indicates if the makespan has been obtained by the simulation model (i.e. the average value of three online simulation runs) or the physical system. Hence, the factor has two levels: (S) Simulation model, (L) Lab-scale physical model.
- *Scenario* is a factor only used for case B. It has three levels that correspond to the three respective failure scenarios: 1, 2, and 3 (section 6.4).

Table 6 summarizes the results that have been obtained in each experimental condition in terms of average makespan $\bar{C}_{max}$. The next two sections comment on the numerical results.

7.1. Case A: Deterministic Failure

In this case, the production is affected by a failure at $t_e = 100s$. Ten replications are done for each experimental condition. Since in each replication three simulation runs are performed, this experiment counts 60 data points for simulation and 20 for the physical model. Figure 8 shows two of the schedules generated by the two reaction rules. The MFJR rule always resulted more advantageous than RSR and has been applied online. The values of makespan measured on the physical system and foreseen by the simulation model can be found in Table 8.

Figures 9a and 9b show the main effects plot and the interaction plot of the results obtained in this case, respectively. Due to non normality of ANOVA residuals, we performed a non parametric test (i.e. Kruscall-Wallis) on the two factors separately. Table 7 shows the obtained results. The *Evaluation tool* factor is not significant and it demonstrates the alignment between the physical system and its digital counterpart. Hence, we may infer that the optimal policy found on the simulation model is also optimal on the physical system. This is confirmed by the numerical results of this case because in all the experiments the MFJR rule outperformed RSR both in the digital and physical models. The *Rescheduling* factor is significantly influencing the response.

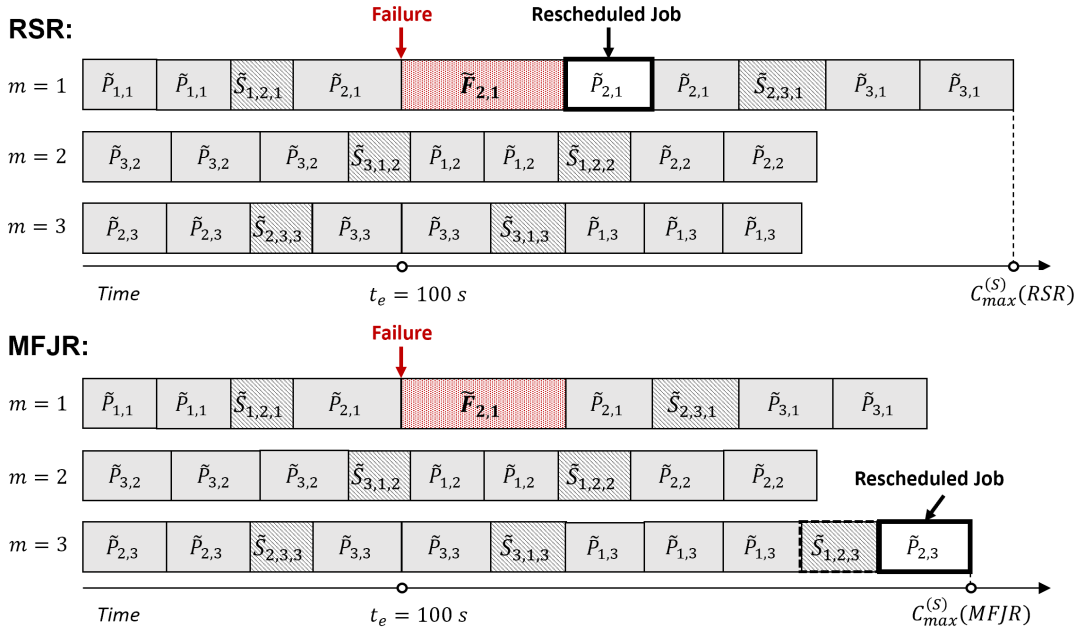
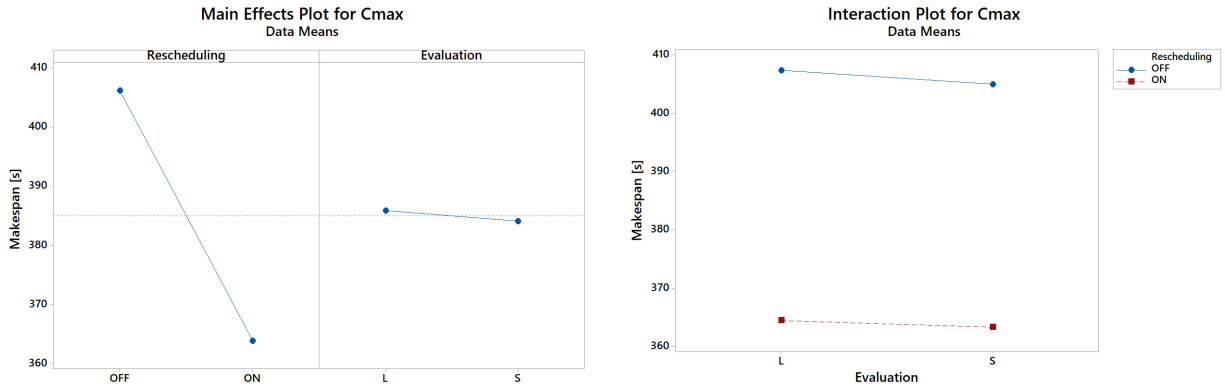


Figure 8: Case A – The schedules generated by RSR and MFJR rules in response to the failure on $m = 1$.



(a) (b)

Figure 9: Case A – Main effects plot (a) and interaction plot (b) for C_{max} depending on the two factors: rescheduling condition, evaluation tool.

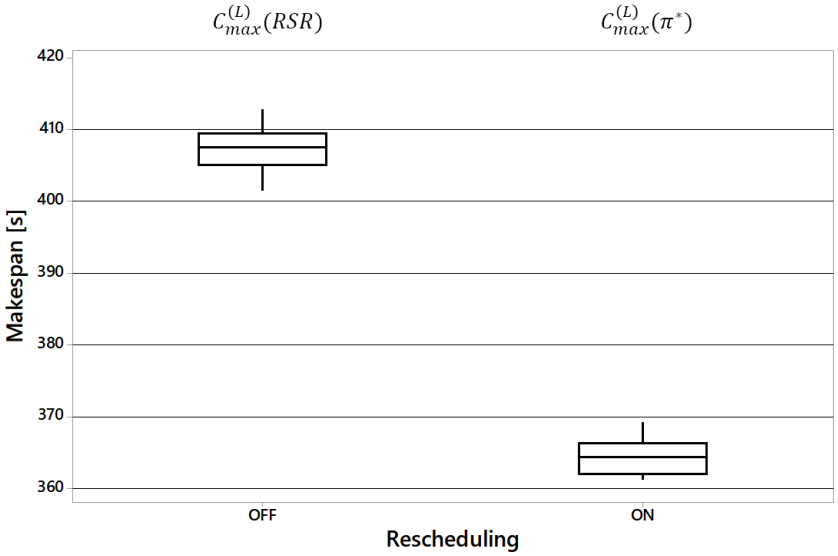


Figure 10: Case A – Box plots comparing the actual makespan obtained in the conditions (1) Rescheduling ON and (2) Rescheduling OFF (10 data samples).

Figure 10 shows the box plots of the makespan obtained in the two rescheduling conditions for Case A. The average difference is $42.91s$ and it is contained in the 95% confidence interval $[40.05; 45.77]$. Finally, from the numerical results we can conclude that the online RTS-based rescheduling led to a significant improvement of the system performance.

7.2. Case B: Stochastic Failure

In this case, three replications are made in each experimental condition. For each replicate, three simulations for both the rules are performed. Hence, this experiment counts 18 data points for the physical system and 54 for the digital model. The values of C_{max} measured on the physical system and foreseen by the simulation model are collected in Table 10.

Figures 11a and 11b show the main effects plot and the interaction plot of the makespan values depending on the experimental factors, respectively. Due to missed normality of ANOVA residuals, a non-parametric test (i.e. Kruscall-Wallis) has been performed to test the influence of the three factors. Table 9 shows the results. The factors *Rescheduling* and *Scenario* are significantly influencing the makespan results, while the *Evaluation tool* factor is not significant. Also in this case, the alignment between the digital and physical model is demonstrated by the fact that the *Evaluation tool* factor is not significantly influencing the makespan results. Figure 11b shows that there is an interaction between the factors *Rescheduling* and *Scenario*. Indeed, in this case, MFJR rule has not always been applied online. Specifically, in the second scenario, a failure occurs on $m = 3$ at $t_e = 46s$ on a job of part type $j = 2$. Given the system state, the most performing decision is to reschedule the failed job according to the RSR rule. Hence – as expected – in the second scenario there is no significant difference between enabling rescheduling or not. This demonstrates that even if a specific reaction rule could prove to be better on average, it may still perform worse in certain settings. Real-time Simulation is able to identify these cases and it allows for a prompt evaluation of the best decision to take so that the performance of a system in a particular status can be maximized.

Rescheduling	**Replication**	$C_{max}^{(S)}$**(RSR)**	$C_{max}^{(S)}$**(MFJR)**	$C_{max}^{(L)}$**(RSR)**	$C_{max}^{(L)}$**(MFJR)**	$t'_e - t_e$
ON	1	403.4 402.2 409.3	360.8 362.7 366.3	369.2		57.9
	2	402.7 401.5 408.6	360.1 362.0 365.6	366.0		54.5
	3	404.1 402.8 409.9	361.5 363.3 366.9	362.8		51.2
	4	403.8 402.5 409.6	361.2 363.0 366.7	362.1		58.4
	5	400.5 399.3 406.3	357.9 359.8 363.4	366.2		53.7
	6	404.3 403.0 410.1	361.7 363.5 367.2	361.7		52.7
	7	404.1 402.8 409.9	361.5 363.3 366.9	366.0		53.7
	8	404.2 402.9 410.0	361.6 363.4 367.0	361.2		51.2
	9	403.8 402.6 409.6	361.2 363.1 366.7	366.6		53.8
	10	403.6 402.4 409.4	361.0 362.9 366.5	362.7		58.3
OFF	1	403.4 402.2 409.2	360.8 362.7 366.3		408.4	57.8
	2	402.7 401.4 408.5	360.1 361.9 365.5		412.9	53.2
	3	404.0 402.7 409.8	361.4 363.2 366.8		406.4	52.9
	4	403.8 402.6 409.6	361.2 363.1 366.7		401.4	57.4
	5	400.5 399.2 406.3	357.9 359.7 363.4		408.7	53.6
	6	404.3 403.0 410.1	361.7 363.5 367.1		405.6	52.0
	7	404.1 402.8 409.9	361.5 363.3 366.9		406.5	53.1
	8	404.2 402.9 410.0	361.6 363.4 367.0		408.9	52.1
	9	403.8 402.5 409.6	361.2 363.0 366.7		403.9	54.7
	10	403.6 402.4 409.5	361.0 362.9 366.5		411.0	58.1

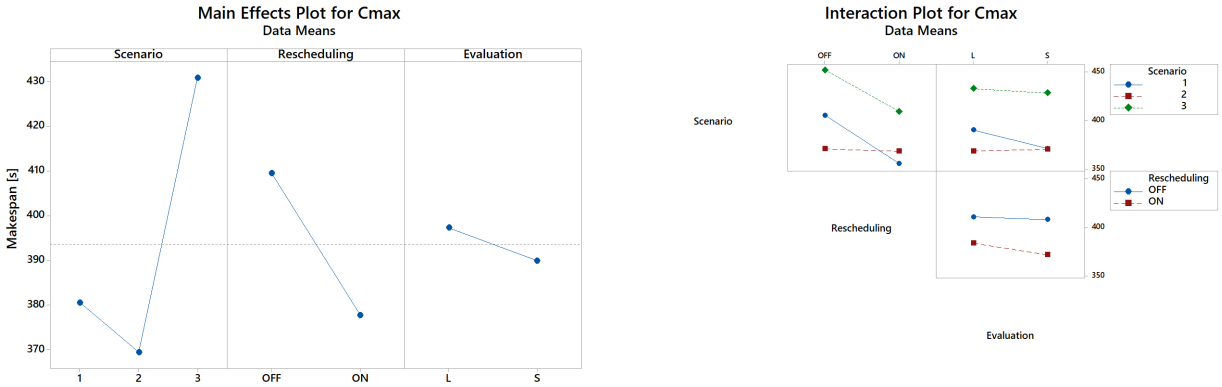
Table 8: Case A – Experimental results.

Factor	DF	H-Value	P-Value
Scenario	2	26.06	0.000
Rescheduling	1	3.85	0.050
Evaluation	1	0.03	0.874

Table 9: Case B – Kruscall-Wallis test results on the three factors.

Rescheduling	**Scenario**	**Replication**	$C_{max}^{(S)}$**(RSR)**	$C_{max}^{(S)}$**(MFJR)**	$\tilde{F}$	$C_{max}^{(L)}$**(RSR)**	$C_{max}^{(L)}$**(MFJR)**
ON	1	1	398.2 410.0 402.9	379.0 381.7 379.8	169.0		374.9
		2	288.7 276.6 280.7	262.2 256.1 257.9	168.9		374.1
		3	403.0 412.1 402.3	375.0 379.9 373.7	168.8		370.3
	2	1	368.8 365.2 368.5	374.7 376.4 373.6	46.0	366.4	
		2	373.5 368.8 368.3	373.5 384.1 388.3	46.0	365.3	
		3	372.7 369.2 372.7	371.3 373.3 379.4	46.0	368.6	
	3	1	447.6 449.5 453.1	405.2 402.4 412.0	178.7		413.3
		2	445.3 449.3 447.4	407.1 412.1 408.5	178.8		415.4
		3	450.2 452.9 447.8	406.8 406.8 405.4	178.7		405.3
OFF	1	1	401.4 399.1 415.9	374.9 366.6 376.9	169.0	408.5	
		2	398.6 404.8 406.1	371.3 380.1 377.3	169.0	405.6	
		3	397.3 403.1 405.8	372.6 362.6 369.4	169.0	407.5	
	2	1	370.8 368.5 374.3	369.9 370.8 372.9	45.9	374.9	
		2	377.0 369.9 367.0	374.3 375.7 378.1	46.0	367.3	
		3	367.3 371.2 372.0	381.7 377.8 368.1	45.9	369.5	
	3	1	449.1 443.5 451.8	408.2 402.2 410.6	178.8	454.1	
		2	448.9 451.3 447.1	404.3 407.2 411.7	178.8	458.7	
		3	456.3 451.2 447.0	406.4 405.8 404.5	178.7	451.8	

Table 10: Case B – Experimental results.



(a) (b)

Figure 11: Case B – Main effects plot (a) and Interaction Plot (b) for C_{max} depending on the three factors: *Scenario*, *Rescheduling condition*, *Evaluation tool*.

8. Conclusions

In this work, we have proposed a lab-scale environment that exploits physical models of manufacturing systems to test production planning and control approaches based on Real-time Simulation. By exploiting easy-to-build components such as LEGO, several types of manufacturing systems can be translated into lab-scale models in a very short time. The advantage of exploiting such a laboratory has been evaluated in terms of the reachable Technology Readiness Level and by using the models for testing an online rescheduling problem. The case study proved that RTS-based production planning and control approaches can be assessed on lab-scale models of common manufacturing systems. Differently from tests performed on real factories, the proposed lab-scale models allow for a much more versatile and cost-effective setting, while maintaining the information loop through industrial components. Hence, we believe the proposed laboratory will be beneficial in several industrial applications. For instance, the design of new production control algorithms could include a testing phase exploiting the physical models. Similarly, new devices such as PLCs could benefit from trials on lab-scale models. Several issues still need to be solved. The synchronization between digital and physical models theoretically allows for the online identification of the optimal production policy. However, the computation time represents a major obstacle. Indeed, although in this work the online comparison has involved two alternative policies, the time span $t'_e - t_e$ has lied between 51 and 59 seconds, which is very close to the downtime duration. In general, simulation time is non negligible and it represents one of the most important challenges of RTS: the number of replications are superiorly limited by the necessity of a timely application of the prescribed actions on the system. On the other hand, reducing the simulation effort may weaken the confidence in the simulation results. Hence, more work is needed for testing production planning problems requiring higher computation effort. In the future, we aim at providing more case studies based on different types of manufacturing systems and introducing more production policies alternatives to explore the applicability boundaries. Another interesting development of this work is the study and formalization of the component types which can benefit from an increased TRL. The TRL that can be obtained by an integrated system can also be assessed, by taking into account the interactions and compatibility between all connected components and digital models.

Acknowledgements

The construction of the physical model and the related software architecture has been partially funded by the *Sme.UP Group* (`www.smeup.com`).

References

[1] Fei Tao, Qinglin Qi, Ang Liu, and Andrew Kusiak. Data-driven smart manufacturing. *Journal of Manufacturing Systems*, 48:157 – 169, 2018. Special Issue on Smart Manufacturing.

[2] Elisa Negri, Stefano Berardi, Luca Fumagalli, and Marco Macchi. Mes-integrated digital twin frameworks. *Journal of Manufacturing Systems*, 56:58 – 71, 2020.

[3] L. Monostori, B. Kádár, T. Bauernhansl, S. Kondoh, S. Kumara, G. Reinhart, O. Sauer, G. Schuh, W. Sihn, and K. Ueda. Cyber-physical systems in manufacturing. *CIRP Annals*, 65(2):621–641, 2016.

[4] Mohsen Moghaddam, Marissa N. Cadavid, C. Robert Kenley, and Abhijit V. Deshmukh. Reference architectures for smart manufacturing: A critical review. *Journal of Manufacturing Systems*, 49:215 – 225, 2018.

[5] Rafal Cupek, Adam Ziebinski, Lukasz Huczala, and Huseyin Erdogan. Agent-based manufacturing execution systems for short-series production scheduling. *Computers in Industry*, 82:245–258, 2016.

[6] Daniel Rossit and Fernando Tohmé. Scheduling research contributions to Smart manufacturing. *Manufacturing Letters*, 15:111–114, 2018.

[7] Mengnan Liu, Shuiliang Fang, Huiyue Dong, and Cunzhi Xu. Review of digital twin about concepts, technologies, and industrial applications. *Journal of Manufacturing Systems*, 2020.

[8] Giovanni Lugaresi and Andrea Matta. Real-time simulation in manufacturing systems: Challenges and research directions. In *Proceedings of the 2018 Winter Simulation Conference*, pages 3319–3330. IEEE, 2018.

[9] Magnus Akerman. *Implementing Shop Floor IT for Industry 4.0*. PhD thesis, 2018.

[10] Qinglin Qi, Fei Tao, Tianliang Hu, Nabil Anwer, Ang Liu, Yongli Wei, Lihui Wang, and AYC Nee. Enabling technologies and tools for digital twin. *Journal of Manufacturing Systems*, 2019.

[11] S. Manivannan and J. Banks. Design of a knowledge-based on-line simulation system to control a manufacturing shop floor. *IIE Transactions*, 24(3):72–83, 1992.

[12] A Mousavi and HRA Siervo. Automatic translation of plant data into management performance metrics: a case for real-time and predictive production control. *International Journal of Production Research*, 55(17):4862–4877, 2017.

[13] N Robertson and T Perera. Automated data collection for simulation? *Simulation Practice and Theory*, 9(6-8):349–364, 2002.

[14] André Hanisch, Juri Tolujew, and Thomas Schulze. Initialization of online simulation models. In *Proceedings of the 2005 Winter Simulation Conference*, pages 1795–1803. IEEE, 2005.

[15] Matthias Blum and Guenther Schuh. Towards a Data-oriented Optimization of Manufacturing Processes. In *Proceedings of the 19th International Conference on Enterprise Information Systems, Porto, Portugal*, pages 26–29, 2017.

[16] Masaki Kitazawa, Satoshi Takahashi, Toru B Takahashi, Atsushi Yoshikawa, and Takao Terano. Real time workers' behavior analyzing system for productivity measurement using wearable sensor. *SICE Journal of Control, Measurement, and System Integration*, 10(6):536–543, 2017.

[17] Mohammed Sadiq Altaf, Hexu Liu, Mohamed Al-Hussein, and Haitao Yu. Online simulation modeling of pre-fabricated wall panel production using RFID system. In *Proceedings of the 2015 Winter Simulation Conference*, pages 3379–3390. IEEE, 2015.

[18] Hao Luo, Ji Fang, and George Q Huang. Real-time scheduling for hybrid flowshop in ubiquitous manufacturing environment. *Computers & Industrial Engineering*, 84:12–23, 2015.

[19] Olivier Cardin and Pierre Castagna. Proactive production activity control by online simulation. *International Journal of Simulation and Process Modelling*, 6(3):177–186, 2011.

[20] Florian Biesinger, Davis Meike, Benedikt Kraß, and Michael Weyrich. A digital twin for production planning based on cyber-physical systems: A Case Study for a Cyber-Physical System-Based Creation of a Digital Twin. *Procedia CIRP*, 79:355–360, 2019.

[21] Young Jun Son and Richard A Wysk. Automatic simulation model generation for simulation-based, real-time shop floor control. *Computers in Industry*, 45(3):291–308, 2001.

[22] Gerardo Santillán Martínez, Seppo Sierla, Tommi Karhela, and Valeriy Vyatkin. Automatic Generation of a Simulation-based Digital Twin of an Industrial Process Plant. In *Proceedings of the 44th Annual Conference*

of the IEEE Industrial Electronics Society, IECON, pages 3084–3089. Institute of Electrical and Electronics Engineers, 2018.

[23] Mutsumi Fujihara and Kiyoshi Yoneda. Simulation through explicit state description and its application to semiconductor fab operation. In *Proceedings of the 1992 Winter Simulation Conference*, pages 899–907. ACM, 1992.

[24] Wayne J Davis. On-line simulation: Need and evolving research requirements. In J. Banks, editor, *Handbook of Simulation*, pages 465–516. New York: John Wiley & Sons, 1998.

[25] Adnan Khan, Martin Dahl, Petter Falkman, and Martin Fabian. Digital Twin for Legacy Systems: Simulation Model Testing and Validation. In *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, pages 421–426. IEEE, 2018.

[26] Giovanni Lugaresi, Gianluca Aglio, Federico Folgheraiter, and Andrea Matta. Real-time validation of digital models for manufacturing systems: a novel signal-processing-based approach. In *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*, pages 450–455. IEEE, 2019.

[27] D. Katz and S. Manivannan. Exception management on a shop floor using online simulation. In *Proceedings of the 1993 Winter Simulation Conference*, pages 888–896. IEEE, 1993.

[28] Gerardo Santillán Martínez, Tommi Karhela, Reino Ruusu, Tuomas Lackman, and Valeriy Vyatkin. Towards a systematic path for dynamic simulation to plant operation: Opc ua-enabled model adaptation method for tracking simulation. In *IECON 2017-43rd Annual Conference of the IEEE Industrial Electronics Society*, pages 5503–5508. IEEE, 2017.

[29] Jerry Banks, John S. Carson, Barry L. Nelson, and David M. Nicol. *Discrete-Event System Simulation.* Upper Saddle River, New Jersey: Prentice Hall, 3 edition, 2000.

[30] Botond Kádár, A Lengyel, László Monostori, Y Suginishi, András Pfeiffer, and Y Nonaka. Enhanced control of complex production structures by tight coupling of the digital and the physical worlds. *CIRP Annals*, 59(1):437–440, 2010.

[31] Behrang Ashtari Talkhestani, Nasser Jazdi, Wolfgang Schloegl, and Michael Weyrich. Consistency check to synchronize the Digital Twin of manufacturing automation based on anchor points. *Proc. CIRP*, 72:159–164, 2018.

[32] L Damiani, M Demartini, P Giribone, M Maggiani, R Revetria, and F Tonelli. Simulation and digital twin based design of a production line: A case study. In *Lecture Notes in Engineering and Computer Science*, volume 2, 2018.

[33] Mohammad Mahdi Nasiri, Reza Yazdanparast, and Fariborz Jolai. A simulation optimisation approach for real-time scheduling in an open shop environment using a composite dispatching rule. *International Journal of Computer Integrated Manufacturing*, 30(12):1239–1252, 2017.

[34] András Pfeiffer, B Kádár, L Monostori, and Zoltán Vén. Situation detection in production control by applying on-line simulation. In *5th International Conference on Digital Enterprise Technology, DET 2008*, pages 225–241. Publibook, 2008.

[35] Pradeep Suresh, John M Wassick, and Jeff Ferrio. Real time performance measurement for batch chemical plants. In *Proceedings of the 2011 Winter Simulation Conference*, pages 2330–2340. IEEE, 2011.

[36] Iracyanne Retto Uhlmann and Enzo Morosini Frazzon. Production rescheduling review: Opportunities for industrial integration and practical applications. *Journal of Manufacturing Systems*, 49:186–193, 2018.

[37] Guilherme E. Vieira, Jeffrey W. Herrmann, and Edward Lin. Rescheduling manufacturing systems: A framework of strategies, policies, and methods. *Journal of Scheduling*, 6(1):39–62, 2003.

[38] S Bohlmann, M Becker, S Balci, H Szczerbicka, and E Hund. Online simulation based decision support system for resource failure management in multi-site production environments. In *2013 IEEE 18th Conference on Emerging Technologies Factory Automation (ETFA)*, pages 1–4, 2013.

[39] Olivier Cardin and Pierre Castagna. Myopia of service oriented manufacturing systems: benefits of data centralization with a discrete-event observer. In *Service Orientation in Holonic and Multi-Agent Manufacturing Control*, pages 197–210. Springer, 2012.

[40] Catherine M Harmonosky, Robert H Farr, and Ming-Chuan Ni. Selective rerouting using simulated steady state system data. In S. Andradottir et al., editor, *Proceedings of the 1997 Winter Simulation Conference*, pages 1293–1298, Piscataway, New Jersey, 1997. IEEE.

[41] Jose M Framinan, Paz Perez-Gonzalez, and Victor Fernandez-Viagas Escudero. The value of real-time data in stochastic flowshop scheduling: A simulation study for makespan. In *Proceedings of the 2017 Winter Simulation Conference*, pages 3299–3310. IEEE, 2017.

[42] Samieh Mirdamadi, Franck Fontanili, and Lionel Dupont. Discrete event simulation-based real-time shop floor control. In *Proceedings of the 2007 European Conference on Modelling and Simulation, June 4^{th}-6^{th}, Prague,*

Czech Republic, pages 235–240, 2007.

[43] Yunqing Rao, Fei He, Xinyu Shao, and Chaoyong Zhang. On-line simulation for shop floor control in manufacturing execution system. In C. Xiong et al., editor, *Intelligent Robotics and Applications. Lecture Notes in Computer Science*, volume 5315, pages 141–150. Berlin, Heidelberg: Springer, 2008.

[44] Heiko Aydt, Wentong Cai, and Stephen John Turner. Dynamic specialization for symbiotic simulation-based operational decision support using the evolutionary computing modelling language (ECML). *Journal of Simulation*, 8(2):105–114, 2014.

[45] Sören Bergmann, Sören Stelzer, and Steffen Straßburger. Initialization of simulation models using CMSD. In *Proceedings of the 2011 Winter Simulation Conference (WSC)*, pages 2223–2234. IEEE, 2011.

[46] Malcolm Yoke Hean Low, Kong Wei Lye, Peter Lendermann, Stephen John Turner, Reman Tat Wee Chim, and Surya Hadisaputra Leo. An Agent-based Approach for Managing Symbiotic Simulation of Semiconductor Assembly and Test Operation. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '05, pages 85–92, New York, NY, USA, 2005. ACM.

[47] Young Jae Jang and Vina Sari Yosephine. Lego robotics based project for industrial engineering education. *International Journal of Engineering Education*, 32(3):1268–1278, 2016.

[48] Giovanni Lugaresi, Nicla Frigerio, Mengyi Zhang, Ziwei Lin, and Andrea Matta. Active learning experience in simulation class using a lego®-based manufacturing system. In *Proceedings of the 2019 Winter Simulation Conference*, pages 3307–3318. IEEE.

[49] *ev3dev*, accessed July 1, 2020. `http://www.ev3dev.org`.

[50] Giovanni Lugaresi, Davide Travaglini, and Andrea Matta. A Lego Manufacturing System As Demonstrator for a Real-Time Simulation Proof of Concept. In *Proceedings of the 2019 Winter Simulation Conference*. IEEE, 2019.

[51] Stanley R Sadin, Frederick P Povinelli, and Robert Rosen. The nasa technology push towards future space mission systems. In *Space and Humanity*, pages 73–77. Elsevier, 1989.

[52] John C Mankins. Technology readiness levels. *White Paper, April*, 6:1995, 1995.

[53] Mihály Héder. From nasa to eu: The evolution of the trl scale in public sector innovation. *The Innovation Journal*, 22(2):1–23, 2017.

[54] William L Nolte. Trl calculator. In *AFRL Assessing Technology Readiness Development Seminar*, 2005.

[55] Jean Paul Arnaout. Rescheduling of parallel machines with stochastic processing and setup times. *Journal of Manufacturing Systems*, 33(3):376–384, 2014.

[56] Jean Paul Arnaout. Heuristics for the maximization of operating rooms utilization using simulation. *Simulation*, 86(8-9):573–583, 2010.