Politecnico di Torino

Master of Science in Engineering and Management (Class LM-31)
Department of Management and Production Engineering

Degree Thesis

# Application of Digital Twin and simulation for dynamic task allocation

Supervisor:
Professor Giulia Bruno

Candidate:
Elena Bianco Prevot
288870

A.y. 2021/2022
December 2022

## Contents

## Abstract

The topic covered in this paper explains a practical application of the project created in the Mind4Lab laboratory of the Politecnico di Torino, with the aim of implementing a Digital Twin, an extremely promising technology in the manufacturing and other fields, research on which is still ongoing.

The assumption that was made at the beginning of the project was that, since the DT is able to replicate the behavior of a system in real time, then by exploiting this data it would be possible to work in a dynamic environment and making decisions consequently to the changes detected in the physical system.

The use of simulation models as a decision-making tool at a company level is a great strategy to solve industrial problems, especially in the manufacturing field, which is why it was chosen as the environment to which to apply the case study. The reasons for this are related to the wide variety of problems that can arise within a production line, to name a few it is possible to find machinery maintenance, staff absences, and unforeseen changes in scheduling. The decision of the specific field to be analyzed was made by looking for the one that would allow the case study to be developed as broadly and comprehensively as possible. The staff absence scenario was ruled out regardless, as a virtual model of human operators would have been extremely error-prone due firstly to the variability of system behavior, which is impossible to predict, and secondly to the practical and technical difficulty of collecting data from such resources. The two candidates from which the decision was made were therefore the topic of predictive maintenance and the topic of dynamic scheduling. Both gave the possibility to monitor a physical production system through sensors, and to collect and analyze data in order to evaluate performance; the reason why the topic chosen in the end was the dynamic scheduling is related to the fact that it also gives the possibility to see how machinery interacts with each other when a variation in production compromises the planned scheduling, thus adding a focus on the connection within the physical system, and not just the connections between physical and virtual and vice versa.

Dynamic scheduling is an evolution of the *Predictive scheduling*. The latter is a strategy widely used for the preparation of a good-quality optimized baseline schedule, which is done in advance taking into account many possible scenarios, and easy to maintain. Although it turns out to be a good technique, it is important to remind that in the manufacturing context, the integrity of a schedule is very easy to be disrupted. The effect of previously introduced events may in some cases lead to the need to completely reschedule the project; Dynamic scheduling is useful in that it allows a strategy on how to first create the initial baseline, but more importantly, in times of need, a strategy on how to respond to events in real time.

The hypothesis to be proved is that, following a generic production line failure, the final results in a dynamic environment should perform better than the theoretical results calculated a priori. The entire work will therefore be carried out with the aim of collecting data in favor of this thesis, and consequently demonstrating that the application of a digital twin is able to bring benefits in terms of performance.

The work is divided into three main parts: the first is intended to introduce all the basic concepts needed to understand the context, the basics of the industry, and the platforms used for implementation; the second is related to the detailed presentation of the case study, with the explanation of the machinery used, the connection methods, and all the programming and development phases that were addressed; the third is related to the previous one, as it includes the phases of data collection, analysis, and comparison of results.

# 1. Introduction to the Digital Twin technology

"A digital twin is a virtual representation of an entity such as an asset, person, organization, or process. Its elements include model, data, unique one-to-one association and monitorability.[1]"

Digital twin is indeed used as an industry 4.0's core technology to the realization of Cyber-Physical Systems (CPSs), capable of bringing many different benefits, depending on the type of application. According to some research, the digital twin was born with two broad purposes: interrogative features and predictive features. The first one refers to the ability of the system to analyze the current and past state of the system, while the latter refers to the ability to predict future states.

In this chapter some topics will be mainly introduced: the history of digital twin technology, the definition of its characteristics, the research progress, and in conclusion the potential implementations of the product.

## 1.1 History and generation of Digital Twin technology

The first digital twin was conceived in the 1960s by NASA as a response to the Apollo 13 explosion. The technology was useful in subsequent Apollo missions, allowing the real-time status of the vehicle to be reflected on a 'twin' left on Earth. Thanks to this type of connection, it was not only possible to observe the actions of the spacecraft, but also to predict future states as optimally as possible, and thus be able to advise the astronaut on the most appropriate maneuvers, based on flexible evaluations that could be modified in real time and no longer programmed in advance.

Later, in 2003, discussions began about the 'mirrored spaced model', a technology based on the same principles as the one produced by NASA, the only difference being that in

[1] Gartner, 21 Lessons From Successful Digital Twin Implementations for Manufacturing, 21 December 2021.

this case the device that collects and analyses data from a physical model, instead of being physical itself, is digital. One of the first to introduce a proper definition of this emerging technology, later to be known as the digital twin, was Professor Grieves, who called it “a digital copy of one or a set of specific devices that can abstractly represent a real device and can be used as a basis for testing under real or simulated conditions”.

In the first period after its presentation, the model was not very successful, especially its possible application to the field of manufacturing. The reasons for this lack of hype lay in the fact that in those years, the means by which information related to the production process could be obtained were extremely limited and difficult to adapt to a digital platform such as the digital twin: much of the information, in fact, was contained in manuals and paper documents. The other reason is instead related to the purely technological aspect, as at the time it was extremely difficult to create an algorithm capable of processing a large amount of data in real time, and consequently the connection between the two spaces was difficult to implement.

The first conceptual model will arrive in 2011, again proposed by Grieves together with John Vickers, and will remain in use for years to come. The system is depicted in image 1-1, in which its main components can be seen: the real space, the virtual space, and the interface zone between the two, characterized by a flow of data from the physical space to the virtual space, and a flow of information in the opposite direction, from the virtual to the physical space. The model is a useful representation of how to make the two spaces interact, and how the virtual model can be used to integrate and modify the real model.

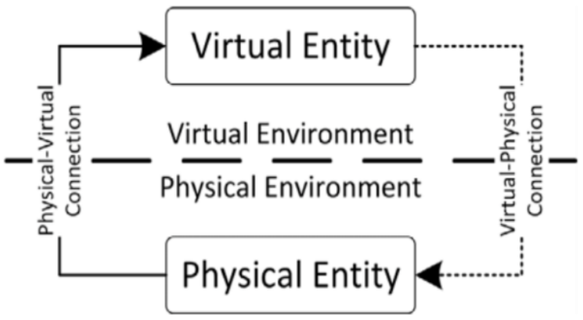


*Figure 1-1 Conceptual model of a digital twin*

From 2011 onwards, therefore, opportunities to develop a digital twin began to materialize. The first was produced by the US Air Force Research Laboratory, with the aim of creating an aircraft twin to evaluate and predict the actions of the physical copy during its whole lifecycle; in particular, its main purpose was to collect data about the performance of the aircraft, analyze them, and simulate the optimal scenario in which the task associated to that specific product could be performed. This data was not the only type of information collected: the Airframe Digital Twin was also associated with all the mechanical characteristics of the object, so that a complete view of the manufacturing information was also available, so that the entire maintenance aspect could be managed.

In 2013, Industry 4.0 and Cyber-Physical Systems, one of its core technologies, were discussed for the first time in Germany.

Industry 4.0, also known as Smart Manufacturing, "is based on the intensified application of Information and Communication Technologies in the industrial environment", with the final aim of achieving industrial automation and introducing some new production technologies to improve working conditions, create new business models, increase plant productivity, and improve product quality, during the whole product lifecycle.

As introduced above, one of the core technologies of Industry 4.0 is the Cyber-Physical System (CPS), which is part of the four categories of core technologies defined by Knudsen, Kaivo-oja, and Lauraeus. The other three technologies are the Internet of Things (IoT), the Cloud Computing, and the Industrial Integration category, with the well-known SOA and BPM. The focus will be on the CPS category, as it is the one including the Digital Twin technology. CPS, in fact, concerns the integration of different kinds of systems; the main purpose of which is to control and manage a physical process, through the exchange of feedback and information for adjustments to be made in real time. The introduction of the Digital Twin concept was therefore crucial to concretize and implement the functionality of CPS.

From 2016 onwards, the real hype moment for the Digital Twin begins. Many companies start to recognize its potential for the application in various sectors; within a year, in fact, DT is cited among the ten most strategic and promising technologies.

This technology has developed rapidly in recent years, both in theory and in practical application. The reasons for this booming are linked to a technological development that has laid a more solid foundation for the DTs implementation, such as the rapid growth of new ICTs, which are more and more adaptable on a large scale and on big data, and the development of intelligent algorithms, related to machine learning and deep learning.

## 1.2 Characteristics and definition of Digital Twin

As introduced in the previous section, the digital twin is not to be interpreted simply as a digital prototype, as it does not only represent the geometry, functions, and performance of its physical object; for this reason, defining it as such would be very reductive. This section will therefore introduce all the main features of the technology, so that to fully understand the potential of this technology.

### 1.2.1 Digital Twin related concepts

When it comes to the digital twin it is common to risk creating misunderstandings. The technology under consideration has some common related terms which, however, represent different concepts. The difference between Digital Model, Digital Shadow, and Digital Twin will then be explained here[2].

With the term Digital Model is intended a digital creation which is not related to an existing object. A typical example could be the blueprint of buildings or roads that are just planned to be built, but that do not already exist. The typical characteristics of this

[2] Robert Woitsch, Anna Sumereder, Damiano Falcioni – Model based data integration along the product & service lifecycle supported by digital twinning, April 2022

concept are: 1. The absence of data exchange between the physical and virtual environment; 2. The absence of impact on the digital model as a result of changes in the physical model, once built.

The second related term is Digital Shadow; in this case the physical object already exists, and the shadow represents it. The main difference is that the data flow is unidirectional because the physical object sends data to the virtual one, affecting its changes, but the digital object cannot send data or information to impose o suggest a change to the physical sphere.

Considering the characteristics that emerged for the two previous technologies, it is therefore clear that the digital twin represents the most complete model, in which data travels in both directions. This means that each component is fully integrated, and so that each change in one environment affects the other. Below is an image that encapsulates all the main concepts of the three models presented in this section.

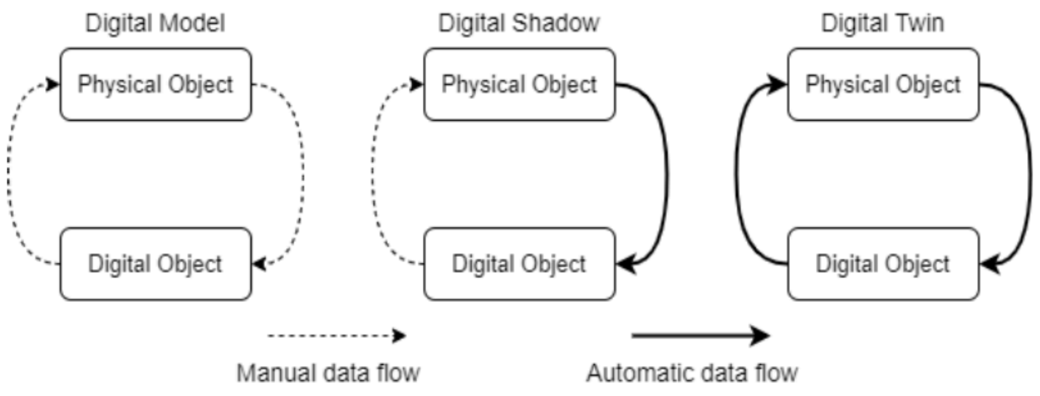


*Figure 1-2 Digital Model, Digital Shadow, and Digital Twin*

### 1.2.2 Main characteristics of the technology

An analysis of the literature reveals several main characteristics of the Digital Twin, some of which have been defined since the emergence of the first models, while others have been introduced over time in accordance with developments in research.

From an academic article[3] it emerges that the main characteristics of the technology under consideration are 12: 1. Physical Entity; 2. Virtual Entity; 3. Physical Environment; 4. Virtual Environment; 5. Fidelity; 6. State; 7. Parameters; 8. Physical-to-Virtual Connection; 9. Virtual-to-Physical connection; 10. Twinning and Twinning Rate; 11. Physical Process; 12. Virtual Process. The same article subsequently defines seven further categories, which are related to system characteristics that have not yet been fully developed, but on which the current research is being based. These are: 13. Perceived Benefits; 14. Digital Twin across the Product Life-Cycle; 15. Use-Cases; 16. Technical Implementation; 17. Different levels of fidelity; 18. Data Ownership; 19. Integration between Virtual Entities.

The first four characteristics are related to the concepts expressed above, namely the difference between physical and virtual entities; a further distinction is done between entity and environment, in order to better differentiate which is the representation of a single object, and which instead represents the whole system operation. Two other important characteristics which have already been introduced are the Physical-to-Virtual and Virtual-to-Physical connections, which permit to implement the bidirectional flow of data.  To conclude the part related to the two different environments, it is also important to differentiate the Physical Process from the Virtual Process, the integration of which forms the basis of learning factories. Here, the digital copy is no longer only linked to the object and the environment in which it is located, but to the entire set of phenomena and actions performed, linked by a nexus, that constitute a process.

The remaining characteristics of a DT system are mainly related to its performance. Fidelity is a concept that has to be estimated, the value of which represents how frequently the model is able to adapt and update itself according to the input it receives. In order to have a good level of fidelity within the system, it is therefore important to focus on the signal transmission optimization, because it is precisely from the fact that

[3] Jones, Snider, Nassehi, Yon, Hicks – Characterising the digital twin: A systemcatic literature review. CIRP Journal of Manufacturing Science and Technology, 2020

the distribution of the time function is discrete that the greatest problems with the accuracy of the system arise. Some of the typical topics that make problems arise, related to discrete signal transmission, are the necessity of time stamps related to the events, the clock synchronization across and within platforms, and a careful analysis and selection of events that need to be processed, and those that are negligible.

The State is a characteristic useful to measure the current condition of both physical and virtual entities, and of all the environment parameters. These Parameters refer to the types of data and information that flows between the two environments. Some examples are: 1. Form, which refers to the geometric structure of the physical entity; 2. Location, related to the geographic position; 3. Processes, which are all the activities in which the entity is engaged; 4. Time, that includes both the duration of a specific action and the time at which an action takes place.

One of the last characteristics to be analyzed is the Twinning, which represents the synchronization between the two different entities. This process consists into measuring the state of the physical and virtual entities to verify that they are equal in each parameter considered in the system. The Twinning process is influenced by every change that occurs in the two entities, which are kept under control and communicated through the two-way connections, Virtual-to-Physical and Physical-to-Virtual. The condition in which both states coincide is defined 'twinned'. The Twinning Rate is then the frequency related to the twinning process, and theoretically it should be in real-time, in order to permit the two entities to work simultaneously and together, obtaining an instant response to changes.

An explanation of some of the seven additional points will also follow, in order to better understand the requirements of Digital Twin across its entire lifecycle, and to focus on existing and under-development methodologies that can be applied to improve the technology.

A particularly important concept is that of Perceived Benefits because it defines all the positive consequences, both economically and organizationally, associated with the application of a DT to a system. These leads to a reduction of costs, risks, reconfiguration time, but also to increasing efficiency, reliability, and optimal manufacturing management and maintenance decision making.

Another important topic to highlight is the Integration between virtual entities. Digital Twin, in fact, can also consists into multiple virtual environments, each one with its own specific use-case. An example to clarify the concept may be represented by the need to deliver a product by a specific deadline, taking into account the possible failure, which must be predicted: each of these tasks can be managed by different, but integrated, digital twins. The basic operation of this concept is to take the output of a virtual entity and use it as a trigger for a further entity, which will analyze it and exploit it to complete its task. It is important to take into account the fact that as the number of digital twins increases, also the complexity of the integration process significantly increases.

## 1.3 Analysis of the research progress

The digital twin in recent years has experienced a great increase in interest from both the academic and research worlds, as well as from industry due to all its possible applications. After introducing the DT and defining its main characteristics, it is also important to assess the current state of research and try to predict the future directions it may take. This paragraph will be indispensable in order to gain clarity about the current state of development of the technology, so as to be able to proceed to analyze, in the next step, some of its most probable applications in the field of interest considered in this work.

### 1.3.1 Gartner’s Hype Cycle

The hype cycle is a useful tool for the graphical representation of the maturity and adoption of a specific technology, as it also indirectly provides information related to future exploiting opportunities.

Gartner Hype Cycle[4] in figure 1-3 shows the point of evolution of many different technologies related to the manufacturing operations, by locating them in specific points of the graph. The Hype Cycle represented above is composed by five phases:

1. Innovation Trigger: A potential technological breakthrough triggers research. At this stage, there are often no usable products, and commercial feasibility is unproven.
2. Peak of Inflated Expectations: initial enthusiasm produces a series of successes, clearly accompanied by multiple failures. Some companies act, continuing with research and development, many others do not and exit the market.
3. Through of Disillusionment: it is known as the phase that creates or destroys the product. Here, experiments and implementations have a high probability of failure. Investment only continues if the surviving suppliers improve their products to the satisfaction of early adopters; if this happens, the technology is very likely to proceed to the plateau stage.
4. Slope of Enlightenment: at this stage, not only the possibility of using the technology materializes, but also do different cases in which it can be applied. Technology suppliers present new-generation products.
5. Plateau of Productivity: mainstream adoption is beginning. The broad applicability and market relevance of the technology is clearly bearing fruit, expanding the earnings of the manufacturing companies as well.

[4] Gartner, Hype Cycle for Manufacturing Operations Strategy, 2022

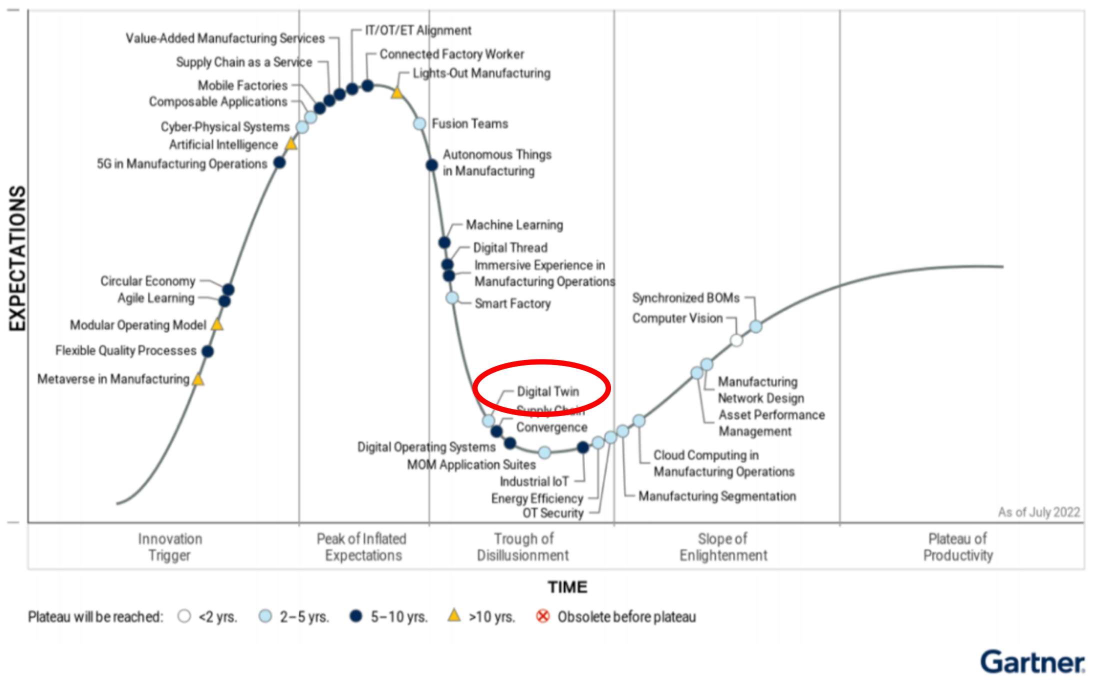


*Figure 1-3 Hype Cycle for manufacturing operations strategy, 2022*

In this case, DT technology is in the third phase, the Through of Disillusionment. The graph in image 1-3 provides further information in addition to the positioning of the technology: it also defines a time range within which the plateau is estimated to be reached. The digital twin, unlike other technologies in the same phase, has an estimated entry time of 2 to 5 years, making it a particularly promising technology.

Gartner's Hype Cycle analysis defined also the market penetration of the technology, which resulted between 1% and 5% of the target audience, and the maturity of the technology, which resulted being 'adolescent'.

### 1.3.2 Drivers of the technology development

Considering the potential of technology, companies are accelerating the adoption of the digital twin as a means of supporting so many of their internal activities, even if nowadays it is still not possible to define a clear and distinct set of drivers for the development of this technology. However, there are some areas where the use of the digital twin is even more pronounced than in others; among these, it is important to mention: 1. Asset-intensive industries (i.e., oil and gas companies, manufacturing, and automotive), in which the technology is indispensable to improve business operations;

2. Military field; 3. Leading-edge enterprises use digital twin to model their IT system or supply chain processes, with the final aim of optimizing cost management; 4. Enterprises that manage huge amount of data use DT technology to create models able to extract the meaningful ones. These environments, although different from each other both in terms of the nature of the business and the scope of application of the digital twin, form a good base of structural factors, useful for generating knowledge and continuing the product innovation process.

### 1.3.3 Potential obstacles in technology development

As mentioned above in the general introduction of the technology, DT has certain fundamental requirements for its implementation, which, if they were to be lacking within specific sectors, could jeopardize its development. The main obstacles that can be noticed are related to many large enterprises' lacks in technological or managerial field. More specifically, companies often do not have enough business objectives, structures, teams or processes to start developing such technology, or in other situations they may not have enough finance and technology. In fact, the implementation of a DT models has a large amount of indirect costs, e.g., related to the preparation of the teams, that must be able to create and maintain a possible portfolio of corporate digital twins, and that must be able to manage the synchronization between the various models. A further example of costs that companies have to bear are the costs of adapting the DT to the corporate system, or the creation from scratch of a technological structure that can enable its implementation. Not many companies have the budget to bear all these costs.

## 1.4 Potential implementations

The purpose of this section is to explain in more detail some of the possible practical applications of the digital twin; to do so, four main fields have been selected in which the technology shows being most promising. The four fields, with their related relevant characteristics, are listed below:

1. Supply Chain: warehouse design optimization, creation of logistic networks.
2. Retail: modeling and simulation of customer's behaviors
3. Healthcare: improvement of operational efficiency of healthcare operations, improvement of personalized care.
4. Manufacturing: performance improvement, predictive maintenance, tasks flexible scheduling.

### 1.4.1 Supply Chain

The functioning of the digital twin, when applied to a supply chain, is always based on the same concept discussed of creating a simulation of the environment, that can predict problems in advance and provide various possible solutions. The use of DT in this field provides some major benefits, including increased productivity, optimized transport, better management of emergency situations, and a much more detailed view of the system.

One of the major form of weaknesses related to the productivity of a supply chain is represented by the inventory management strategy, because it involves changing the supply chain itself, which often means having to deal with unexpected consequences that could not have been foreseen by a simple analysis made beforehand. Another different application concerns the optimization of transportations: through the use of the virtual model, companies have the opportunity to better execute their deliveries, having immediate information about possible slowdowns and blockages and, as a consequence, having the opportunity to suggest the better alternative strategy among all the scenarios evaluated in real-time.

### 1.4.2 Retail

The retail sector is included in the list published by Forbes Technology Council in 2020 that selects 16 most promising application fields for DT able to modify the consumer market. In this area, it is therefore important to focus on transforming the customer experience, while also integrating analysis and forecasting of buyer behavior. In the case of the customer experience, it is important to consider the self-checkout model as example, which, when integrated with digital twin, allows for real-time action that avoids queues at the checkout, improving queue management, and facilitating transactions by working on the speed of automated checkouts.

The application of this technology can, in addition, have direct and indirect impacts on the product itself, which in its digital version can be monitored, modified, and updated, thus allowing changes to be made in parallel on the physical product, after careful evaluation of possible optimizations.

A final benefit is related to the industry's financial analyses and projections; technology provides companies with the ability to refine and optimize profit forecasts, to adjust and adapt prices, and to customize discount and promotion policies for customers. Here again, the main problem is the same as highlighted above: all these implementations, however, presuppose the existence of a suitable support architecture.

### 1.4.3 Healthcare

The healthcare sector is beginning to adopt digital twins firstly to improve the performance of the healthcare organization, but also to enhance the personalized medicine. Today's DTs in the medical field are an important innovation; they are able to create models based on information from wearable devices and shared patient records, to generate connected networks involving patients, doctors and healthcare organizations, as well as drug and device manufacturers.

The term Personalized Medicine refers to the implementation of medical treatments that are customized to the individual's genetics, anatomy, behavior, and other factors. The most important example is given by virtual organs, which make it possible to understand the progression of a disease over time, in order to assess the response to drugs, treatments, or surgeries. To summarize, the underlying concept is to create a patient-specific virtual replica of the organ, with an anatomical analysis performed by artificial intelligence, in order to improve and predict the study and treatment of diseases.

Further promising applications in this area include scheduling of surgical operations, full-body scanning, and optimization of drug dosing.

### 1.4.4 Manufacturing

The application in the field of manufacturing is certainly one of the most promising; the industry, in fact, is characterized by the use of high-cost equipment, which consequently produces a large amount of data. This creates the perfect environment for the implementation of the digital twin. Some of the main applications in manufacturing are product development, performance improvement, predictive maintenance, and tasks dynamic scheduling. The first two applications are particularly related. In fact, the digital twin can be used to monitor and analyze products during their entire life cycle, thus helping experts to decide whether or not to produce a product by understanding its feasibility. At a later stage, on the other hand, it is useful for analyzing and monitoring final products to see which are faulty or underperforming.

A further very useful application, capable of bringing both economic and operational benefits, is predictive maintenance. Manufacturers use the digital twin to predict machine downtime, so that companies can manage maintenance activities without wasting time and incurring additional costs. This also improves the overall efficiency of the machines, as technicians intervene before a breakdown occurs. The problem that arises is an economic one as the use of digital twins for predictive maintenance activities is not scalable. It is in fact a machine-specific virtual replica and requires costly investments on the part of the company related to data science to build and maintain the twins.

The last application is that of dynamic scheduling, or flexible scheduling. This method is a crucial point in manufacturing systems, as these are always characterized by uncertainty of various kinds, such as job insertion, machine breakdown, workers absence and so forth. The integration of a DT can help monitoring and detecting disturbance in the system in an immediate way, due to the constant comparison between virtual and physical model.

# 2. Introduction to the platforms utilized to build the system: FlexSim and Node-Red

In this chapter, the two platforms used for the implementation of the digital twin will be introduced, with a special focus on their main functions, in order to understand as comprehensively as possible how their use was crucial.

The first to be introduced is FlexSim, an object-oriented software whose purpose is to develop, model, simulate and monitor dynamic flow process systems. This platform was used for the graphic visualization of the system, based on the information received from the physical objects. With regard to the connections between physical and virtual entities, and vice versa, Node-Red was used as an intermediate platform, capable of receiving and collecting data from the physical system and sending it to the virtual one for graphic representation. The phase of analyzing the information, followed by its processing and decision-making to suggest corrections or changes to the physical system.

## 2.1 FlexSim: simulation environment

Flexsim is a software that provides a complete suite of development tools, to create and compile simulations of real systems. Animations allow for optimal visualization of the system, and the platform supports tree view, 2D, 3D, and virtual reality.

The problems that usually arise within a system can all be traced back to two main sources: uncertainty and complexity. As the number of components in a system increases, it will become increasingly complex to understand and describe the relationships between the components. Similarly, when uncertainty is introduced into a system consisting of several components, most analytical or predictive methods used in prior fail. This happens because the uncertainty can be of various kinds, e.g., machinery breaking down, variable times, batch sizes, or absence of workers. FlexSim is therefore an excellent tool for gaining knowledge about a complex system that is subject to uncertainty. It is indeed very important to remember that only when a system is well analyzed and the relationships between its components are understood, then it can be improved.

The core functionality of FlexSim is a simulation application compiler that allows users to develop simulations with the help of graphical user interfaces, object libraries and menu structures for a variety of applications in the marketplace.

A further level belonging to the simulation environment is the Flexsim Developer, which is used to develop simulation applications. This developer contains tools and interfaces that, as shown in Figure 2-1, provide standard objects required for simulations, such as sources, queues, robots, processors, etc., which can be used directly, or modified by adding user-defined designs and information to match the desired model. The action of modifying these features is permitted by the Developer, which in itself already provides a number of standard alternatives; in addition, the user, using the Visual C++ compiler, can go on to make further modifications as required.

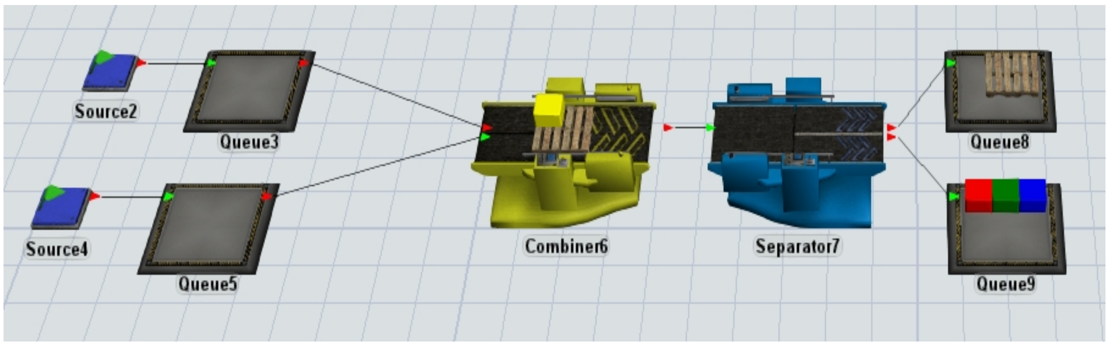


*Figure 2- 1 Example of simulation objects provided by the developer*

The last level of use is the Simulation Application, whose function is to construct a discrete-event simulation using the tools listed before from Flexsim. This simulation application permits many different types of simulations, such as the FlexSim GP (general purpose simulations), the FlexSim Port and SANS (for Shares Access Network Storage systems). Each application can be designed and modified, which means that each model can be faster, more efficient, and more effective, on the basis of the needs of the creator.

## 2.2 FlexSim: model development

The process of model development is composed by five basic steps: 1. Layout development; 2. Object connection; 3. Addition of details; 4. Run of the model; 5. View of output.

### 2.2.1 Layout development

The layout is structured by selecting objects from the library, which is shown in Figure 2-2, and dragging them into the model creation space, which is a virtual 3D spatial environment. For each object, it is possible to position it where optimal by changing its x, y and z spatial coordinates manually. After being placed in the environment, when the layout is complete, what will be displayed on the screen will be an image similar to the one shown in figure 2-1, in which the objects represent a real system, and are connected to each other.

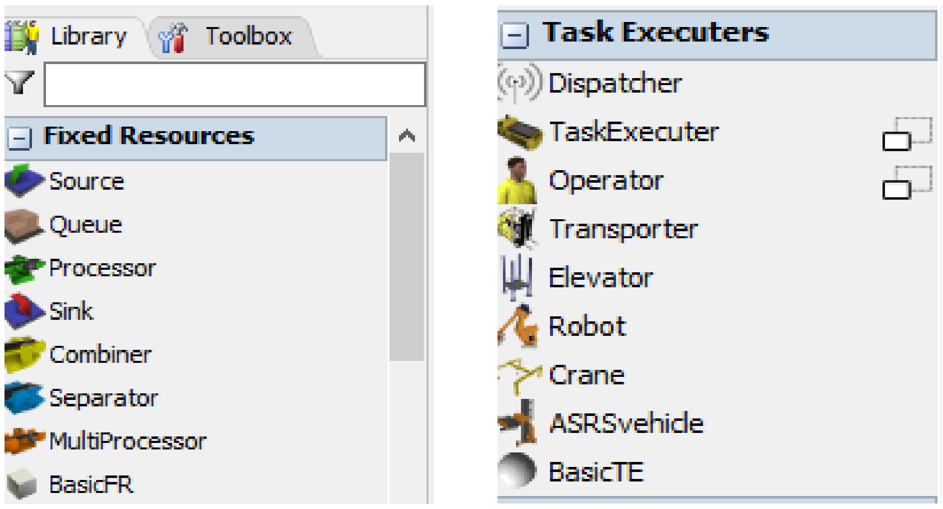


*Figure 2- 2 FlexSim library tool*

As can be seen in the previous image, there are different types of resources in the library. Actually, it contains many different categories, but these two, Fixed Resources and Task Executers, are the main ones. A fixed resource, which represents sources, queues, processors, sinks, combiners, etc., is an object that remains stationary in the model. On the contrary, the Task Executers are resources that are not stationary: they may travel, execute load and unload tasks of flow items, or act like a shared resource among fixed objects.

### 2.2.2 Object connection

FlexSim provides two different types of connection, one for the objects (A-Connects), and another for the center ports (S-Connects). Each single object, instead, is equipped with three ports: input, output and center port.

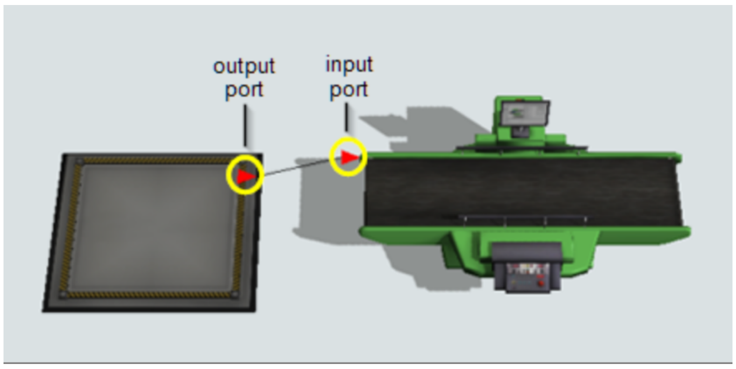


*Figure 2- 3 A-Connects*

Input and output ports are represented with small red triangles, and the direction of the connection is illustrated by the orientation of the triangle, which points in toward the next object for input, and away from the object for output.

The S-Connects, instead, is usually used to connect task-executers to fixed resources, while in other cases it can also connect two resources that need to communicate. The main actions enabled by center ports are:

- Flow items transport: if a task executer is connected by center port between two fixed resources, it can be used as a transporter.
- Setting up: the model may include setup for its resources, and so there could be the need to have a task executer connected through the center port during setup times.
- General reference: as said before, fixed resources may simply have the need to communicate or reference each other.

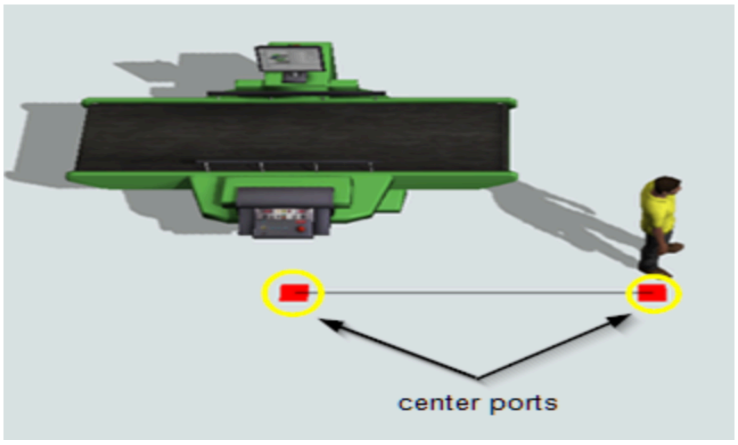
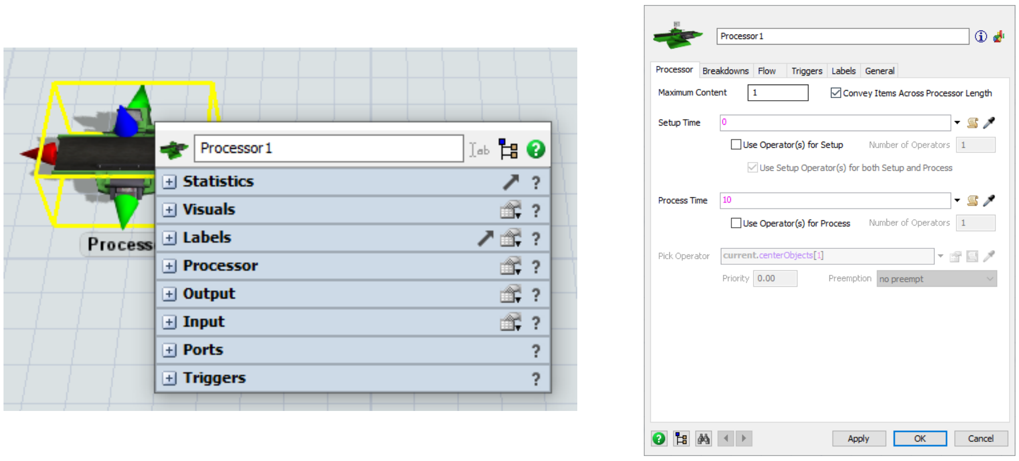


*Figure 2- 4 S-Connects between center ports*

### 2.2.3 Addition of details

Once the layout has been completed and the links created, the platform will allow the user to add logic and other types of data to the objects. The process consists of selecting the layout window, and then editing or entering information such as cycle times, capacity, routing logic, downtime, and statistics. Also in this case, the information inserted can be chosen among the standard ones provided by FlexSim, or can be added by the user through the use of flexscript or C++.



Processor1

Maximum Content 1 ☑ Convey Items Across Processor Length

Setup Time 0

☐ Use Operator(s) for Setup Number of Operators 1

☑ Use Setup Operator(s) for both Setup and Process

Process Time 10

☐ Use Operator(s) for Process Number of Operators 1

Pick Operator current.centerObjects[1]

Priority 0.00 Preemption no preempt

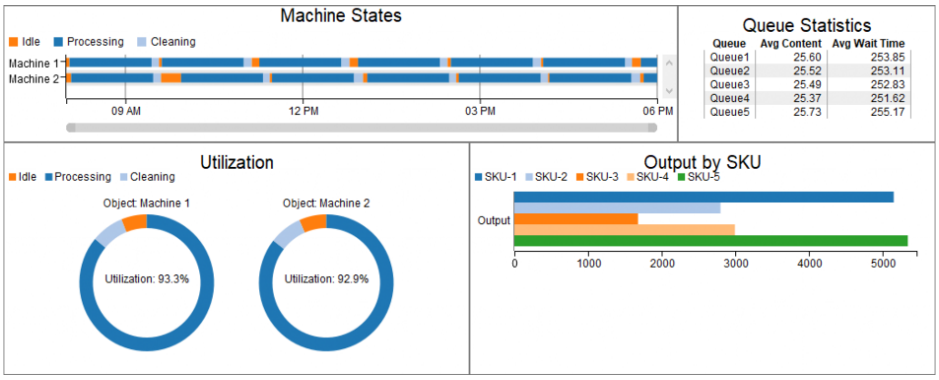
*Figure 2- 5 Panel that permits to modify data*

### 2.2.4 Run the model simulation

As soon as the simulation is activated, the model execution begins, and various conditional scenarios are evaluated. FlexSim collects and returns the data generated by each execution, which can comprise a single scenario, or multiple scenarios. The FlexSim simulation has multiple functions: users can define the conditions, variables, and constraints to be tested, as well as decide the number of times each condition is to be executed and the duration of each execution. At the end of the simulation, performance measures will then be provided, which will provide. a starting point on which to evaluate scenarios and how to optimize them.

### 2.2.5 View of output

Simulation results tend to be displayed in two ways; in the first case, each simulation can be viewed dynamically in 2D, 3D and VR animation as the model runs, while in the second case, reports are printed on a dashboard, showing the parameters chosen by the user. FlexSim's animation provides the ability to view multiple windows of the same model, to move within them in order to zoom in and out of the model, and to view it from the most appropriate angle. All these view manipulations can be performed without affecting the execution. As for the dashboard visualization, the results of each model run are based on predefined parameters, user-defined parameters, predefined graphs, and user-defined graphs, as can be seen in the following illustration, Figure 2-6.



Queue Statistics

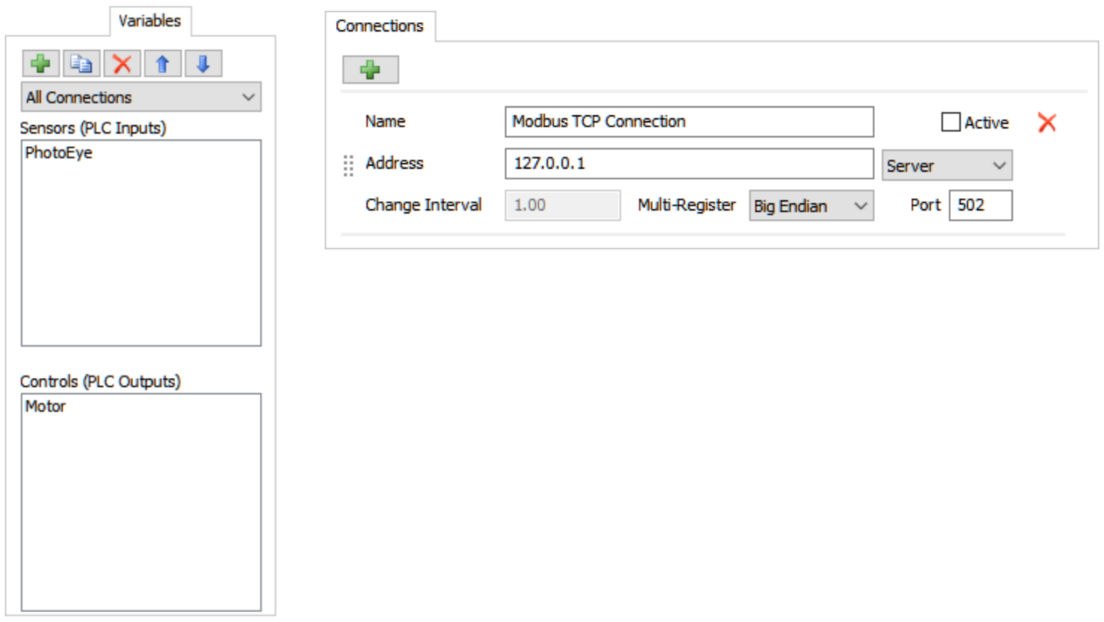
| Queue | Avg Content | Avg Wait Time |
|---|---|---|
| Queue1 | 25.60 | 253.85 |
| Queue2 | 25.52 | 253.11 |
| Queue3 | 25.49 | 252.83 |
| Queue4 | 25.37 | 251.62 |
| Queue5 | 25.73 | 255.17 |

*Figure 2- 6 Example of dashboard in FlexSim*

## 2.3 FlexSim: emulation tool

Emulation refers to the ability to simulate a Programmable Logic Controller (PLC). The emulation tool in this context is in fact used whenever the system needs to use a PLC, so that it can be developed and tested. A PLC, as mentioned before Programmable Logic Controller, is a computer used to interact with machinery in production systems. Specifically, these computers possess a logic, which acts as the brain of a production system, managing all inputs, outputs, internal communications, and the behavior of the system as a whole.

In this context, it is therefore necessary to create a connection between the emulation tool in FlexSim and an external server, or another external PLC. The first of the emulator's two main characteristics, shown in Figure 2-7, is therefore represented by the *connection*; the details about how it happens are very technical, so they will only be introduced in this section to provide the basic concepts but will then be expanded upon later, in the part where the application in the case study will be explained. The second technical characteristic of the emulation tool is called *variables*, and it refers to any input and output that can be received by or sent to the PLC. The two variables recognized by the platform are: sensors, for the inputs, and controls, for the outputs.



All Connections
Sensors (PLC Inputs): PhotoEye
Controls (PLC Outputs): Motor

Name: Modbus TCP Connection ☐ Active
Address: 127.0.0.1 Server
Change Interval: 1.00 Multi-Register: Big Endian Port: 502

*Figure 2- 7 Two FlexSim windows that show respectively variables and connections*

As said above, controls are the PLC's outputs, which means that they receive an information from the server and tell the system what to do to respond to that specific signal. Typical examples of controls are: start and stop to the conveyors or to the processors, positioning in a specific place, issue a warning, etc.

Sensors, instead, are inputs to the PLC, that contain information about the physical environment considered. Usually, sensors are connected to position detectors, photo eyes, and any other sensor able to monitor the system. The basic logic related to the interactions between these entities is based on the PLC, which receives inputs from the system, processes them and communicates with the system, via the outputs, to advise what action to take on the basis of the previously evaluated data.

## 2.4 Node-red: functionalities of the platform

The digital visualization of the real model on FlexSim presented a challenge, as not all data are digitally available, and therefore need to be processed or derived in different ways. Thus, the simple transfer of data from the physical to the digital system required the use of an intermediate platform, Node-Red.

Node-RED is a programming tool conceived with the idea of managing the IoT world through data flows. Its basis programming language is JavaScript. The main feature of this platform is to create flows that permit a communication between hardware devices and online services. In these flows, packets of data travel via some protocols, and pass between nodes performing actions, calculations, or analyses.

*Figure 2- 8 On the left the creation of a flow with four nodes; on the right the programming environment available for each node*

As already mentioned, the operation of Node-Red is based on its nodes, also known as 'black-boxes', each of which has its own specific purpose; it receives data, processes it as ordered in the programming environment, and finally passes it on. This functionality is the main reason why it is used as an intermediate step in the communication between the physical model and FlexSim, in order to pass already processed compatible data to the final application.

Also, in Node-Red there is the possibility to visualize the collected data, through the use of a dashboard. The latter can be created on the basis of the user's needs, by placing specific nodes in the vicinity of the data of interest, in order to collect them as output and plot them on tables, graphs and other types of interfaces for their graphic visualization.

## 2.5 Node-Red: representation of a model through flows

With Node-Red, the programming is done through the concatenation of objects, assigning each of them a different task in order to have a complete description of the assumed, and to be assumed, behavior of the model. Nodes, in order to communicate with each other, use a pre-established data packet, known as a message, the content of which may vary according to requirements. Each message exchanged has two characteristics: the *topic*, and the *payload*. The topic identifies the scope, is set using a string and may be associated with any value, while the payload represents the value of the information being transmitted.

### 2.5.1 Flows

The *flow*, defined by a name and a specific description, is the main way to organize the model logic and represents a set of organized and connected nodes. Each flow has two main characteristics: Properties and Environment Variables. The first one is used to simply define the flow with name and description, while the second one collects all the properties that are valid as environment variables within the flow.

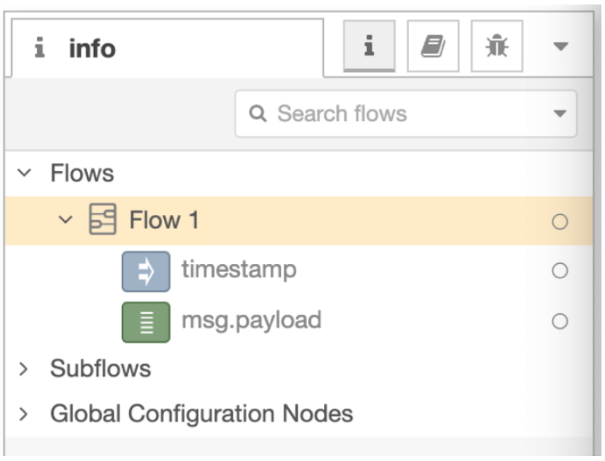


*Figure 2- 9 Information box showing a flow with its nodes*

### 2.5.2 Nodes

Nodes are the objects whose interaction constitutes a flow. Node-Red provides a wide set of predefined nodes that can be adapted and used in many different flows, or the user may import them from the library or from the clipboard.

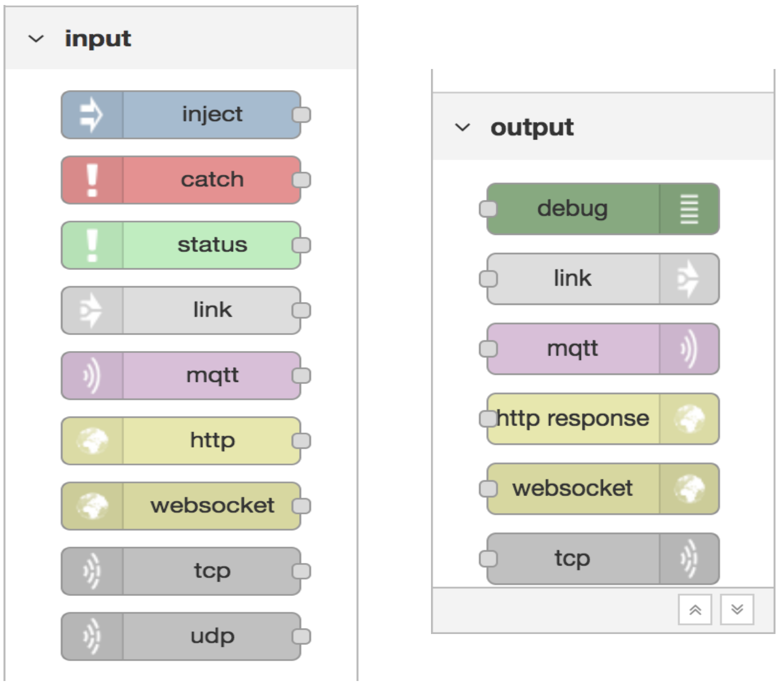


*Figure 2- 10 Palette containing some input and output nodes*

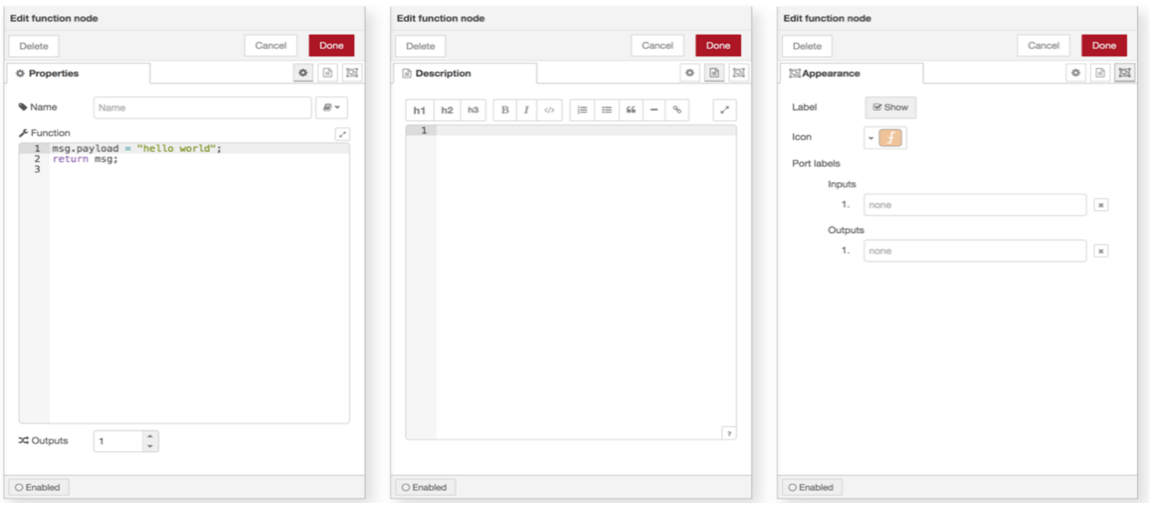
Once the appropriate node has been selected from those available, its configuration can be edited. The edit contains three tabs, that are shown in Figure 2-11: 1. Properties; 2. Description; 3. Appearance.

The properties tab permits to edit the node's properties in a JavaScript programming environment. It could be important to set:

- Value, which is the type related to the property, and it is mandatory to define.
- Optional Boolean that activates whether the property is required, and that is set to null if the property is invalid.
- Optional function that can be used to validate the value of the property.
- Optional Node's Type, if the property is a pointer to a configuration node that needs a type.

The description is a per-node documentation, which is shown in the information bar when the node is selected.

The appearance is instead the option to customize the node, in fact, the tab provides many options, such as to select whether the node's label is shown, to change the icon, to provide custom port labels, etc.



```
msg.payload = "hello world";
return msg;
```

Name: Name
Outputs: 1
Inputs 1. none
Outputs 1. none

Figure 2- 11 Node edit: properties, description, and appearance tab

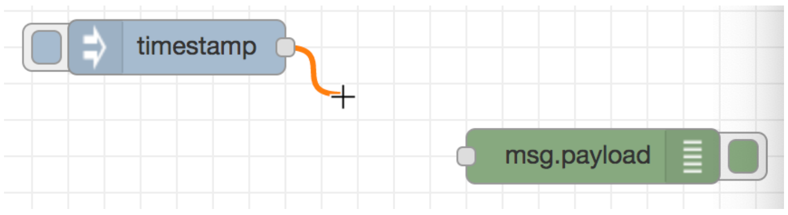


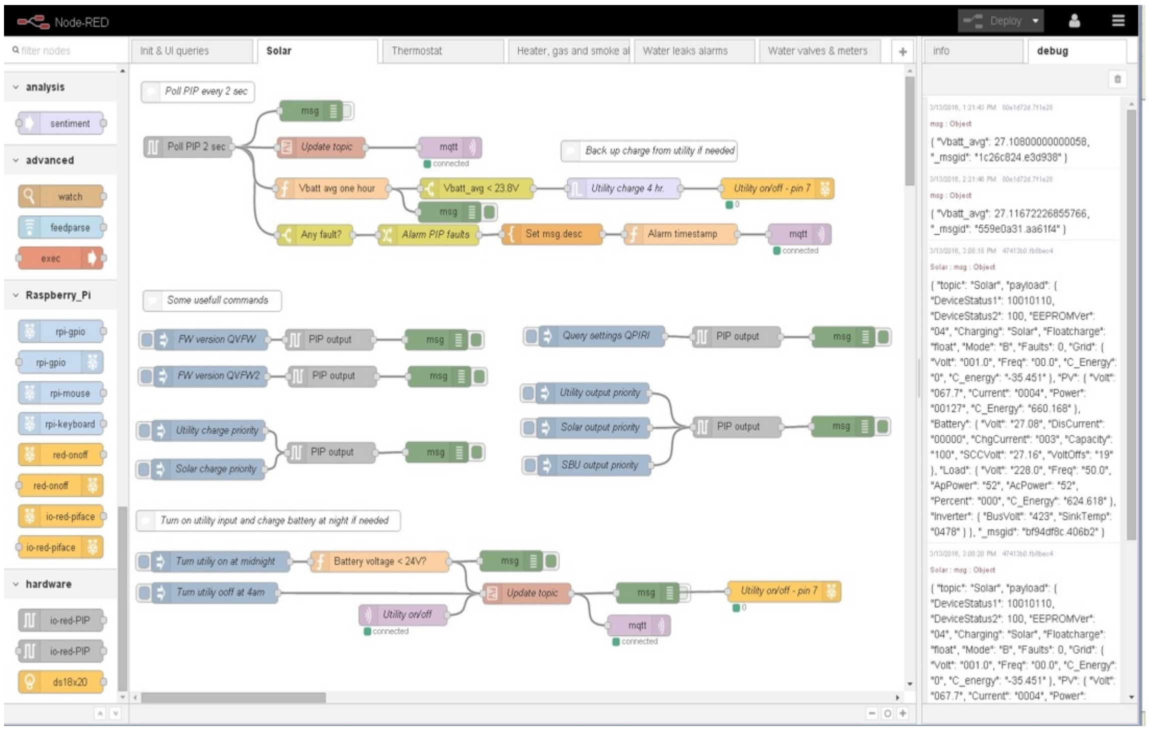
Figure 2- 12 Wiring nodes

Nodes are connected to each other through wires, that are created by simply dragging the link from the output port of a node to the input port of the following, as can be seen in Figure 2-12. Other actions that can be done on wires are splitting and moving; if a node with both input and output ports is placed on a wire, it can be inserted in the flow simply by releasing it on that point on the wire. To move a wire from a node, it would be sufficient to take it from the extremity that is to be moved and move it to a new port.

In conclusion, it is also important to remember that nodes can be switched on and off as needed, using a simple command to stop the flow of messages from that node forward in the flow.

An important tool is the one that allows you to manage the palette of nodes, managing them or installing new ones. The palette manager has two tabs: one lists the nodes currently installed in the runtime, and the other permits to see a list of nodes that can be installed.

The following image, Figure 2-13, shows a representation of what a complete Node-Red flow with nodes, connections, and implemented logic looks like.



```
3/13/2016, 1:21:40 PM  80e1d72d.7f1e28
msg : Object
{ "Vbatt_avg": 27.1080000000058,
"_msgid": "1c26c824.e3d938" }
3/13/2016, 2:21:46 PM  80e1d72d.7f1e28
msg : Object
{ "Vbatt_avg": 27.11672226855766,
"_msgid": "559e0a31.aa61f4" }
3/13/2016, 3:08:18 PM  4741360.fb8bec4
Solar : msg : Object
{ "topic": "Solar", "payload": {
"DeviceStatus1": 10010110,
"DeviceStatus2": 100, "EEPROMVer":
"04", "Charging": "Solar", "Floatcharge":
"float", "Mode": "B", "Faults": 0, "Grid": {
"Volt": "001.0", "Freq": "00.0", "C_Energy":
"0", "C_energy": "-35.451" }, "PV": { "Volt":
"067.7", "Current": "0004", "Power":
"00127", "C_Energy": "660.168" },
"Battery": { "Volt": "27.08", "DisCurrent":
"00000", "ChgCurrent": "003", "Capacity":
"100", "SCCVolt": "27.16", "VoltOffs": "19"
}, "Load": { "Volt": "228.0", "Freq": "50.0",
"ApPower": "52", "AcPower": "52",
"Percent": "000", "C_Energy": "624.618" },
"Inverter": { "BusVolt": "423", "SinkTemp":
"0478" } }, "_msgid": "bf94df8c.406b2" }
3/13/2016, 3:08:20 PM  4741360.fb8bec4
Solar : msg : Object
{ "topic": "Solar", "payload": {
"DeviceStatus1": 10010110,
"DeviceStatus2": 100, "EEPROMVer":
"04", "Charging": "Solar", "Floatcharge":
"float", "Mode": "B", "Faults": 0, "Grid": {
"Volt": "001.0", "Freq": "00.0", "C_Energy":
"0", "C_energy": "-35.451" }, "PV": { "Volt":
"067.7", "Current": "0004", "Power":
```

*Figure 2- 11 Example of a complete flow in Node-Red*

# 3. Case study developed in the laboratory

This chapter will discuss the case study which has been chosen to finalize this work, and to illustrate all the processes that will lead to prove (or disprove) the assumption made at the very beginning, i.e. that the implementation of a digital twin to support flexible scheduling, leads to cost and performance advantages in a production line.

The situation that will be proposed in the case study is as follows: two collaborative robotic arms perform the same task, which consists of taking a unit to be processed from the starting point, and releasing it at the finishing point. The batch size is ten units and therefore, in the better scenario, each robot will process five units. In the next paragraph, it will be seen how, in the programming phase, loops of five repetitions were generated, so that if the best-case scenario occurred, each of the two arms would perform its own sequence of tasks, without having to make any changes to the system.

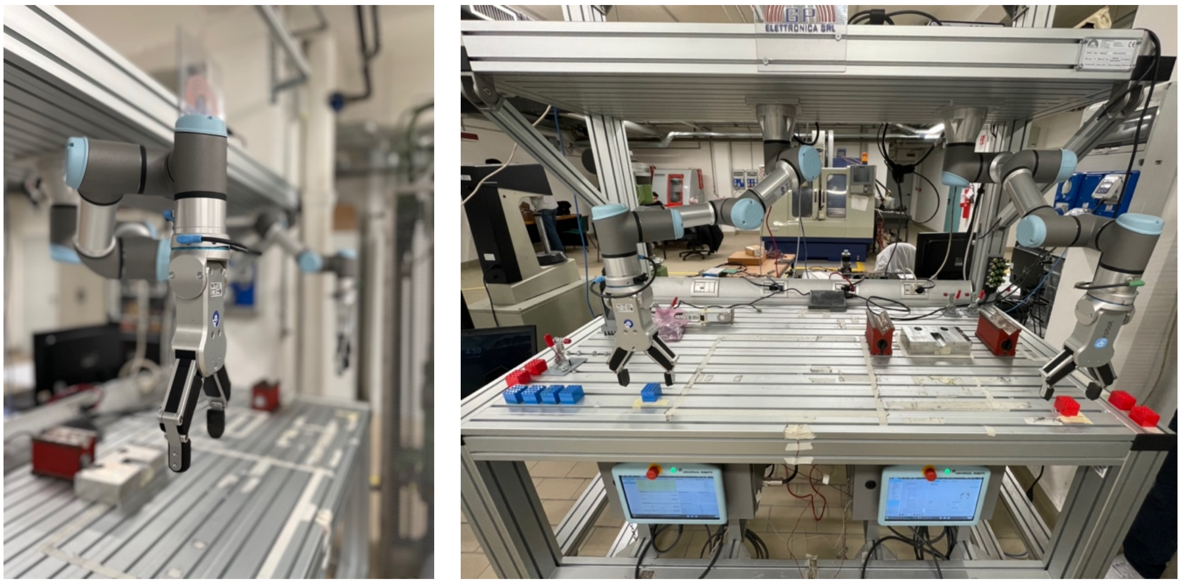


*Figure 3-1 Physical system used to test the Practical Worst-Case*

The alternative situation to be evaluated will instead aim to propose a worst-case, a *time failure* scenario, in which one of the two robots will process the units more slowly. At this point, there are two possible outcomes:

2. Do-nothing scenario, in which the unaffected robot will process its five units as planned, and the other will cause a delay in production.
3. Mitigation scenario, which is the focus of this work, and involves using the digital twin to recognize the robot that is causing delay, and to then send a signal to the working robot. The latter, once it has processed its five units, will again enter the production phase, processing the units for the slow robot as well. By doing so, there should be instant scheduling changes.

The final aim of this whole experiment involves a phase of collecting the results and comparing the various scenarios. Obviously, despite the application of the digital twin, it will be impossible to mitigate and obtain performance values on a par with the optimal scenario; however, with this method of solving a time failure, the final results should lie in an intermediate scenario between the best and the worst, in which no action is taken at all.

## 3.1 Digital Twin applied to a production line to achieve dynamic scheduling

The case study developed in the laboratory involves the creation of three different entities: the physical space, the virtual space, and the connections between the two, in both directions. The physical space is represented by two collaborative UR3e robots whose associated task is to perform the pick and place action on a batch of units to be processed. The two robots therefore represent two machines of a very simple production line, which is consequently represented in the virtual model on FlexSim. The virtual model is organized in such a way that the two objects shadowing the physical machineries not only receive the data, but are also particularly realistic representations of them. For the graphic visualization part FlexSim was used, as it provides an optimal environment in which to create a faithful model. For data processing, Node-Red software was used to read the data from the robots, pass it to FlexSim, and to process the information collected and send commands to the robots to optimize, when possible, the process.

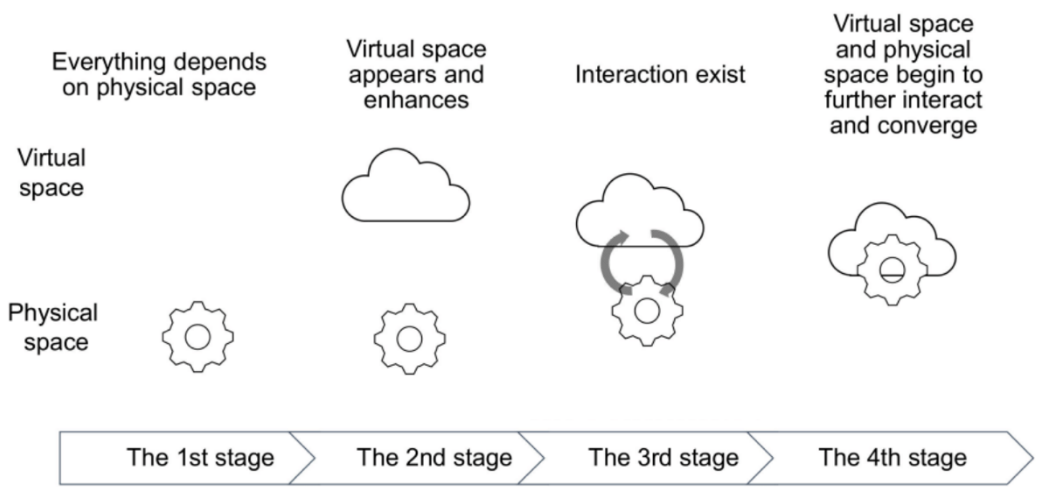

Everything depends on physical space
Virtual space appears and enhances
Interaction exist
Virtual space and physical space begin to further interact and converge
Virtual space
Physical space
The 1st stage
The 2nd stage
The 3rd stage
The 4th stage

*Figure 3-2 Steps related to the creation of a Digital Twin model*

Three scheduling models will be proposed to analyze this production line:

- non-dynamic scheduling: calculated a priori, representing the optimal production case, i.e. the scenario in which no unexpected events of any kind occur.
- non-dynamic scheduling, also calculated a priori, which shows what would happen to the system if one of the two machines slowed down but there was no sensor to detect it, and thus no corrective action (do-nothing scenario).
- dynamic scheduling, achieved by evaluating the data passing through the digital twin's system in real time, and thus able to take corrective action (mitigation).

The following sections will introduce in detail the creation of the various entities, explaining all the procedures carried out in the laboratory, and then it will be explained in detail how the various scheduling models were designed.

## 3.2 Physical system

The physical system considered in this case study consists of two machines of a production line. The two machines chosen are two collaborative robotic arms, capable of performing the pick and place action. The machines were programmed to work in parallel, equally sharing the production of a batch of ten units.

### 3.2.1 UR3e collaborative robot arm

UR3e is a compact collaborative robot, and the smaller of the e-series produced by Universal Robots. These robots are made entirely of plastic and aluminum, with the addition of some steel components, resulting in an extremely low weight. Other typical features are a maximum load of 3 kg and a reach of 500 mm, making them particularly suitable for assembly tasks. Thanks to their compactness, they can be easily positioned in any orientation, and the assembly procedure is also particularly simple due to their total weight of around 11 kg.

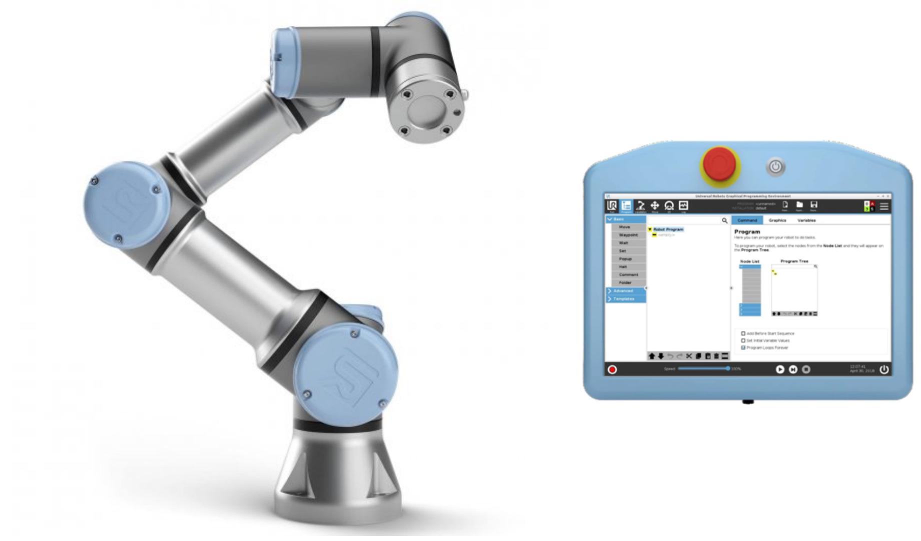
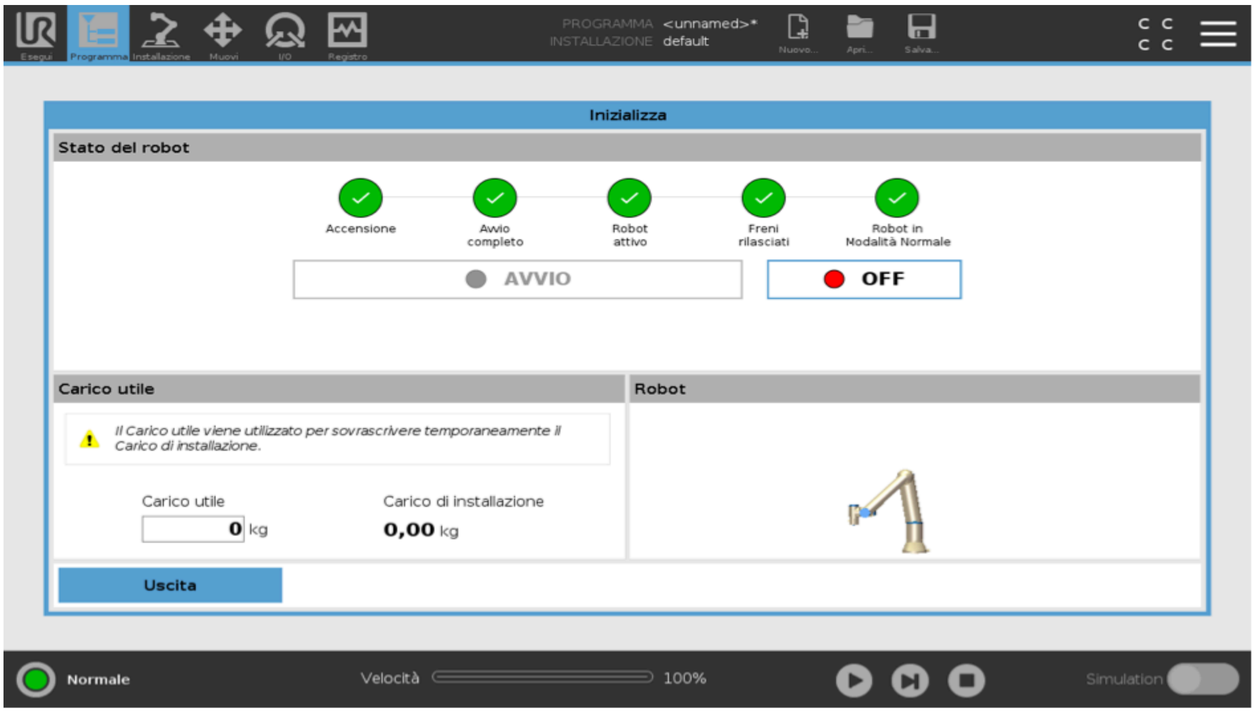


*Figure 3-3 UR3e collaborative robot and its Teach Pendant*

The arm of the Universal Robot consists of tubes and joints, the movement of which can be easily coordinated by locating the tool in the desired position, except for the part bordering the base. There are 6 joints, and these are the ones that ensure the movement of the robot; you have the base, which is the element on which the robot is mounted, followed by the shoulder and then the elbow, which are the ones that perform the movements with greater amplitude, while at the end of the robot there are then the wrists, 1, 2, and 3. Wrists 1 and 2 take care of the finer movements, while wrist 3 is the one to which the tool is attached.

Each robot has its own control unit, which includes all electrical inputs and outputs connecting the robot arm, the Teach Pendant (on the right in figure 3-3) and any other peripherals.

For switching the robot on, off, and programming tasks, the Teach Pendant is used. The robot switches on from the main on/off button but remains in a state of waiting for the



Il Carico utile viene utilizzato per sovrascrivere temporaneamente il Carico di installazione.

Carico utile 0 kg

Carico di installazione 0,00 kg

*Figure 3-4 Start-up screen on the Teach Pendant*

start signal, to prepare it for normal operation. Only after the start signal has been given via handheld, then the robot is actually ready. Figure 3-4 shows the screen to start it up for operation, once it is switched on.

### 3.2.2 Robot programming phase

In the previous section, the main characteristics and some basics concerning the use of robots were introduced. In this section, however, all the operations performed on the physical system will be explained in order to implement the production line by programming the tasks to be performed by the robots.

The production line to be created consists, as explained above, of two machines that process a batch of ten units in parallel, splitting it in half. Theoretically, to complete their respective tasks, the two robots would then have to pick up a unit at a time at the initial position and transport it to the final position, working at the same speed.

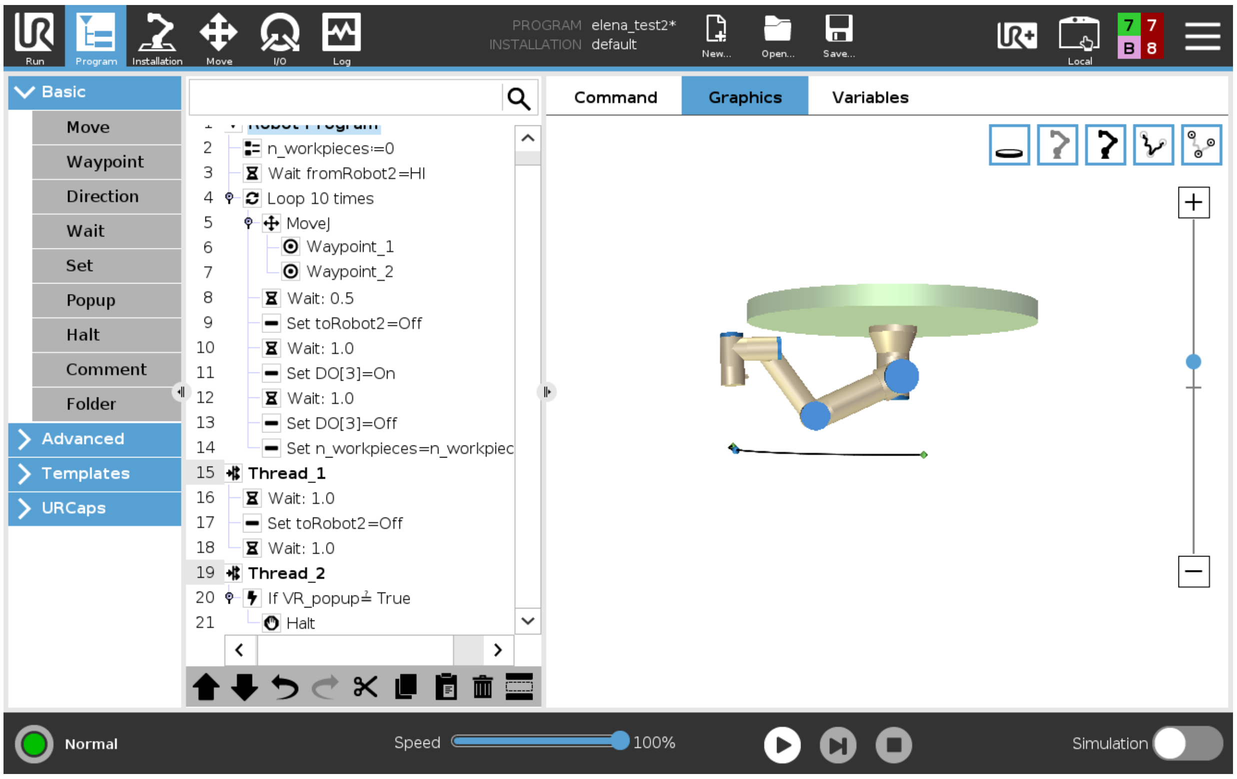
In order to achieve this, it is therefore necessary to create a program using the Teach Pendant; this consists of serially inserting commands from the list on the left in Figure 3-5.

The main steps to focus on when creating the robot program are the following:

- Set the initial and final positions of the robot. There are two methods to do that, the first one, the *freedrive* function, consists in choosing the position by moving the robot to the point of preference: the spatial co-ordinates where it is located will be read automatically and will be recorded as a *waypoint*. The second method permits to choose the x, y and z coordinates in millimeters and in radians, relative to the tool position, and then choose the angle in degrees for each joint. Figure 3-6 shows the panels used to position the robot: waypoints 1 and 2 were set manually so that the first corresponded to the robot's start position, i.e., the position in which it performs the gripping action on the units in the batch, while the second represents the point in which the robot releases the unit at the end of the task.
- Define the main characteristics of the task: in this case, having to process five units each in an optimal situation, a loop was defined for both robots consisting of five repetitions, which can be restarted at any time if necessary. The loop is illustrated in Figure 3-7.
- Adding all the extra features that permits to optimize the process. In this case, it was necessary to add waiting times for digital input in order to regulate the execution of the program. All the waiting times that can be read in Figure 3-5

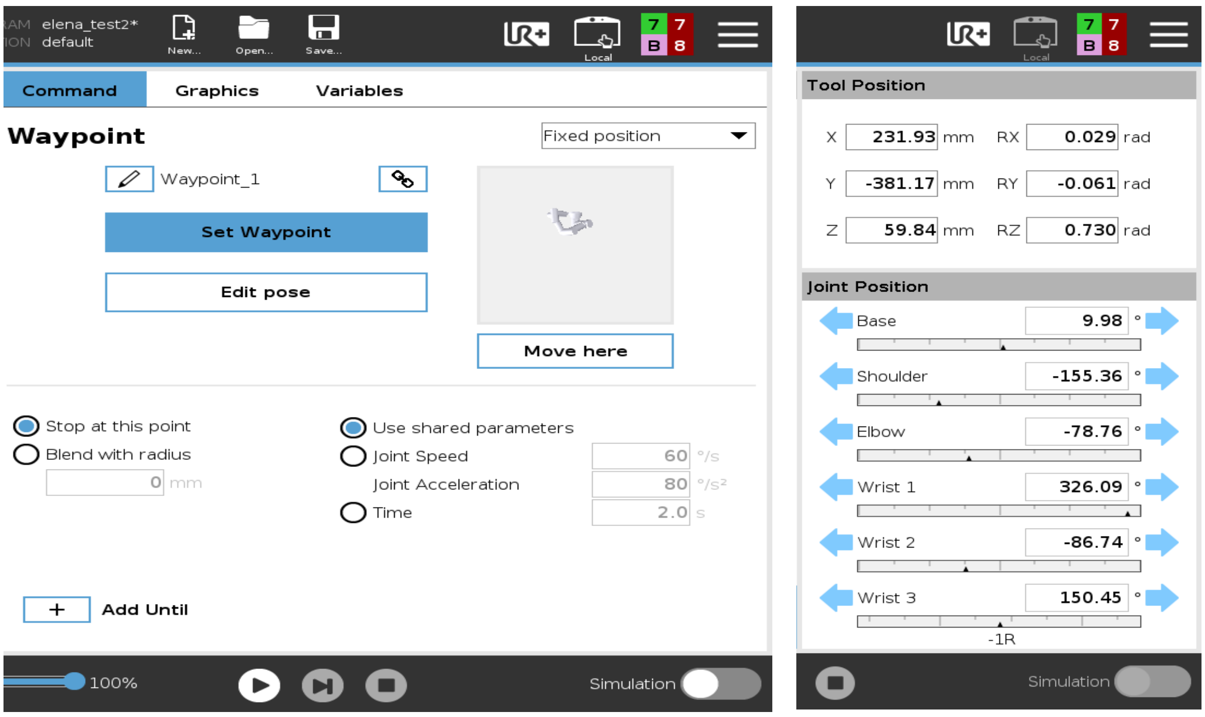
relating to the program represent waiting times and the exchange of external or internal signals, the so-called digital inputs and digital outputs, which allow more control over the execution of the loop.

- Adding threads. A thread is a part of a program outside the main thread, but which is executed concurrently. The usefulness of setting up threads containing start and stop signals in this case is that the robot is always listening for these signals. Threads 1 and 2 are two subprograms which perform a start and a reset action on command; these two actions will be explained in more detail in the section on connections, as they are handled by the internal connections between robots, in the case of thread 1, and between robot and node-red, in the case of thread 2.



```
▾ Robot Program
    n_workpieces≔0
    Wait fromRobot2=HI
    Loop 10 times
      MoveJ
        Waypoint_1
        Waypoint_2
      Wait: 0.5
      Set toRobot2=Off
      Wait: 1.0
      Set DO[3]=On
      Wait: 1.0
      Set DO[3]=Off
      Set n_workpieces=n_workpiec
Thread_1
    Wait: 1.0
    Set toRobot2=Off
    Wait: 1.0
Thread_2
    If VR_popup≟ True
      Halt
```

*Figure 3-5 Robot's main program on the left, and graphical representation on the right*
*Screenshot taken from Robot 1*



[✓ Stop at this point]
Blend with radius
0 mm

[✓ Use shared parameters]
Joint Speed 60 °/s
Joint Acceleration 80 °/s²
Time 2.0 s

**Tool Position**

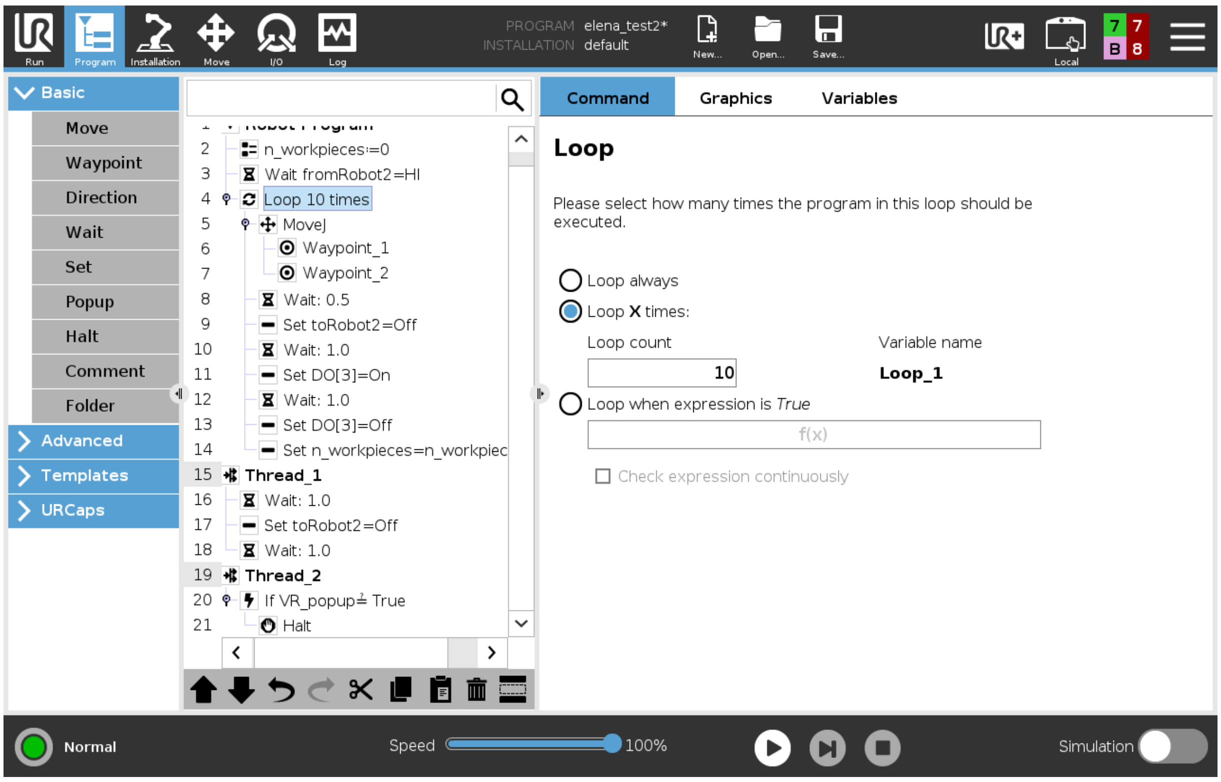
| | | | | | |
|---|---|---|---|---|---|
| X | 231.93 mm | RX | 0.029 rad | | |
| Y | -381.17 mm | RY | -0.061 rad | | |
| Z | 59.84 mm | RZ | 0.730 rad | | |

**Joint Position**

| | |
|---|---|
| Base | 9.98 ° |
| Shoulder | -155.36 ° |
| Elbow | -78.76 ° |
| Wrist 1 | 326.09 ° |
| Wrist 2 | -86.74 ° |
| Wrist 3 | 150.45 ° |

-1R

*Figure 3-6 Position settings*
*Screenshot taken from Robot 1*



PROGRAM elena_test2*
INSTALLATION default

```
Robot Program
  n_workpieces:=0
  Wait fromRobot2=HI
  Loop 10 times
    MoveJ
      Waypoint_1
      Waypoint_2
    Wait: 0.5
    Set toRobot2=Off
    Wait: 1.0
    Set DO[3]=On
    Wait: 1.0
    Set DO[3]=Off
    Set n_workpieces=n_workpiec
Thread_1
  Wait: 1.0
  Set toRobot2=Off
  Wait: 1.0
Thread_2
  If VR_popup≟ True
    Halt
```

**Loop**

Please select how many times the program in this loop should be executed.

Loop always [✓ Loop **X** times:]

Loop count: 10 — Variable name: **Loop_1**

Loop when expression is *True*: f(x)

Check expression continuously

Speed 100%

Figure 3-7 Focus on definition of the loop
Screenshot taken from Robot 1

## 3.3 Virtual system

The creation of the virtual system is the second fundamental step in the creation of a digital twin. As already explained, the environment used for the graphic visualization of the model is FlexSim.

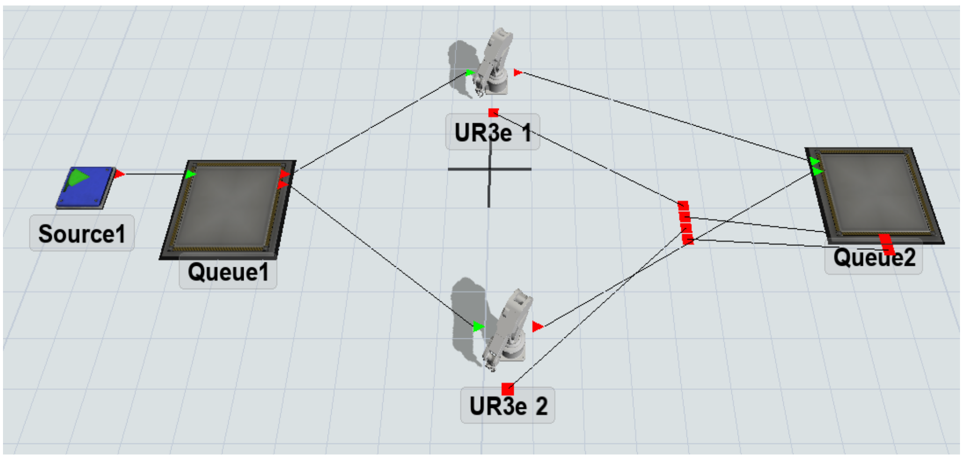


*Figure 3-8 Graphical representation of the model in which the two robots in parallel are digital shadows of the real UR3e in the laboratory*

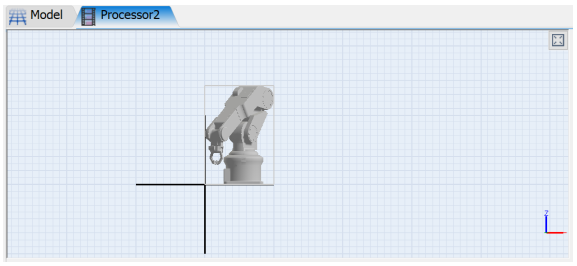
Next, the entire digital model building phase will be explained, including any problems encountered and solutions.

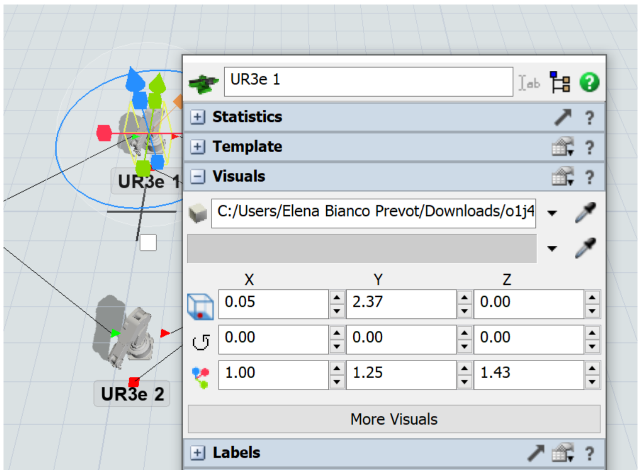
The virtual system was created in such a way as to respect the organization of the physical one. Therefore, two processors were placed in parallel, so that they could perform their tasks in the same way as the UR3e in the laboratory. The remaining system components, on the other hand, are unrelated to the real model, but were indispensable to ensure the most realistic process possible on FlexSim. These other components will be explained below.

- The *source* was used to generate the batch of ten units, thanks to its function, named arrival schedule, which allows the user to select the size and arrival time of the batch.
- The queues represent the initial and final coordinates in which the physical system’s robots do their pick and place actions*. Queue* 1 automatically receives all the units to be processed in the instant they are generated, and places them in a specific space, ready to be collected by the robots. The system's default settings require that the queue itself sends the units to the next machines, thus using a push strategy; these settings have consequently been modified to follow a pull strategy as in the physical system, according to which it is the robot that fetches the unit to be processed at the moment it is required.

As mentioned above, physical robots are represented on FlexSim via processors, although by default these have a completely different geometric shape. The reason why they were chosen as objects for representation anyway are twofold: the processors have a large number of functionalities, especially when connected to the emulator tool, while the robot, despite existing as a representable object, is regarded by the platform as a task executor, and consequently would not have provided useful data if connected to the emulator.

The following images in Figure 3-9 will show how it is possible to change the appearance of objects in the system to make them as realistic as possible.





UR3e 1

C:/Users/Elena Bianco Prevot/Downloads/o1j4

| | X | Y | Z |
|---|---|---|---|
| | 0.05 | 2.37 | 0.00 |
| | 0.00 | 0.00 | 0.00 |
| | 1.00 | 1.25 | 1.43 |

*Figure 3-9 On the left the graphical representation of the robots; on the right the panel that permits to modify the geometry and import the robot shape*

## 3.4 Connections

Connections are the fundamental part of a digital twin, as without them it would only be a digital shadow of a physical system. The presence of connections in fact not only allows data to be passed to the digital system, which will then be able to replicate the form and behavior of the physical one, but more importantly the connections allow the flow of information in the opposite direction, which is important for suggesting changes to the physical system. It is precisely this real-time, bidirectional flow that allows the optimization of the system, being able to recognize changes and propose solutions while the process is in progress.

In this section, all types of connections implemented within the system will be introduced: the connection from physical to virtual, i.e. between robot and FlexSim, via Node-red, and the connection from virtual to physical, which mainly takes place from Node-Red to the robots, as it is in this platform that the data analysis and program execution takes place. However, albeit slightly less optimal, a reverse connection starting from FlexSim was implemented in order to demonstrate that the system is able to work in its entirety.

Finally, a third type of connection will be introduced, which is not characteristic of a digital twin system, but which in this case was nonetheless fundamental to the development stages: the connection between the two robots in parallel. It is important because the two machines are able to send signals via digital input and digital output to each other, which are easier to manage at the program level in the Teach Pendant, and highly functional.

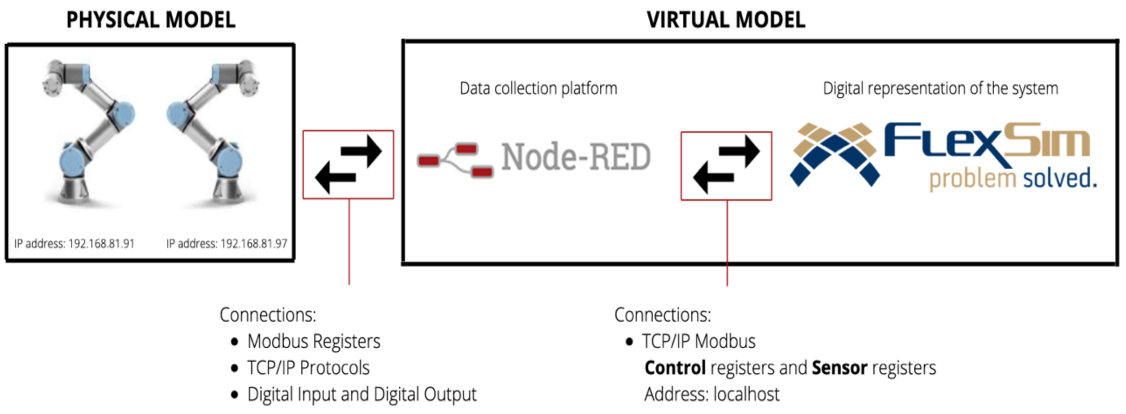


*Figure 3-10 Representation of bi-directional interactions between systems and platforms*

### 3.4.1 Physical-to-Virtual data flow

The connection that takes place from the robots to the digital model is called Physical-to-Virtual data flow, as a large amount of data is passed from the former system to the latter.

With the connection implemented between the physical model and the virtual model, it was possible to collect the following data:

- Status of the process
- Robot process time, which is calculated as the time between when the robot is in its initial position, and when it returns to the same, after unloading the unit in its final position
- The number of units processed by each robot

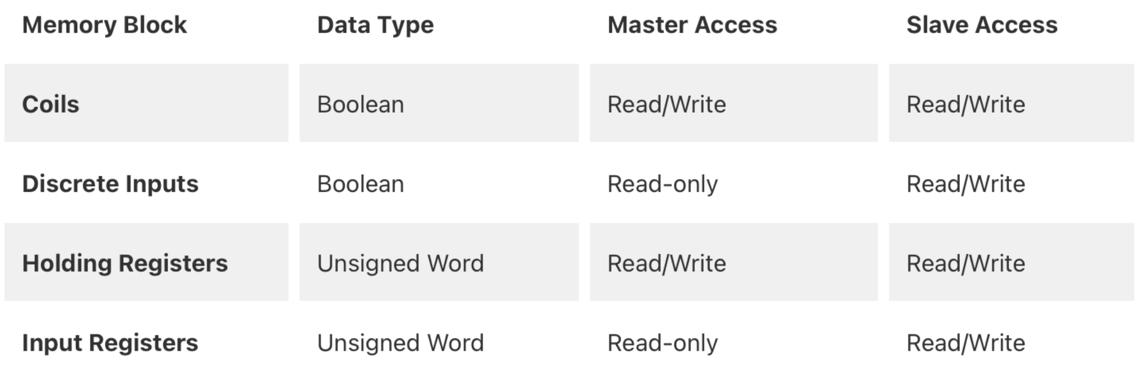
All this data is calculated based on the data received from the TCP-in node, which is then processed to obtain different information to be passed to FlexSim.

To obtain or send data and information packets, most modern systems use open-source industrial connection protocols, whose main characteristic is the ease of implementation, thanks in part to the large number of connected software that has emerged since talk of Industry 4.0 and the Internet of Things began. Next, it will be seen how these methods are applied to achieve the purpose of manipulating data from the various models.

### 3.4.1.1 Modbus Protocol introduction

The protocol used in this case study is called Modbus. Modbus is an industrial protocol, developed in 1979, that represents the main means of communication between automation devices. More specifically, Modbus is a request-response protocol based on the connection between a master and a slave, that in this specific case is the Programmable Logic Controller (PLC) of the robot.

Modbus is characterized by four data banks, whose main purpose is to define the type and the access rights of the contained data. It is important to remind that these four blocks are purely conceptual, in fact, they can both exist as separate memory schemes or be overlapped. These banks are resumed in the following table.



| Memory Block | Data Type | Master Access | Slave Access |
|---|---|---|---|
| **Coils** | Boolean | Read/Write | Read/Write |
| **Discrete Inputs** | Boolean | Read-only | Read/Write |
| **Holding Registers** | Unsigned Word | Read/Write | Read/Write |
| **Input Registers** | Unsigned Word | Read-only | Read/Write |

*Table 3-1 List of the different banks with their main characteristics and data access permitted*

Modbus provides a specific Register Indexing Scheme: it defined a specific range, from 0 to $2^{16}$, in which each number can address a specific data, and it also associated a prefix from 0 to 4 to specify which of the four data blocks the user wants to exploit, in the order in which they are listed in Table 3-1.

A problem that may arise concerns a mismatch between the functions allowed by the master, and what the slave can handle. To solve this, the Modbus TCP specification has listed three classes called *conformance classes*: class 0, class 1, and class 2, each of them with some related codes supported both by master and slave.

- Class 0: these codes are considered the basic ones of a Modbus device, and are Read Multiple Registers and Write Multiple Registers, with their associated number. Their unique mission is to permit the master to write and read data.
- Class 1: this class provides 7 codes, for all the four memory blocks listed before, to both read and write data.
- Class 2: it includes more specialized functions, but it is very difficult to implement.

#### 3.4.1.2 Data reading from robots: TCP-in node

Following the brief theoretical introduction on Modbus protocols, it is now possible to see how they are implemented in the real case in the laboratory. This section will specifically introduce the connection between the robots and the Node-Red platform, which is done via TCP/IP protocols. The Transmission Control Protocol / Internet Protocol is a set of network standards that encompasses the rules of communication on the Internet, using an IP address for sending data packages.

In Figure 3-10 it is possible to notice that Node-Red uses two connections of the type introduced above, which are circled in red: the grey nodes on the left are TCP-in node, that use an IP address and a specific port (the number on the right after the colons) to get the data. The two IP addresses are related to the two robots: the first one, 192.168.81.91, is associated to Robot 1, while the second one, 192.168.81.97, is associated to Robot 2.

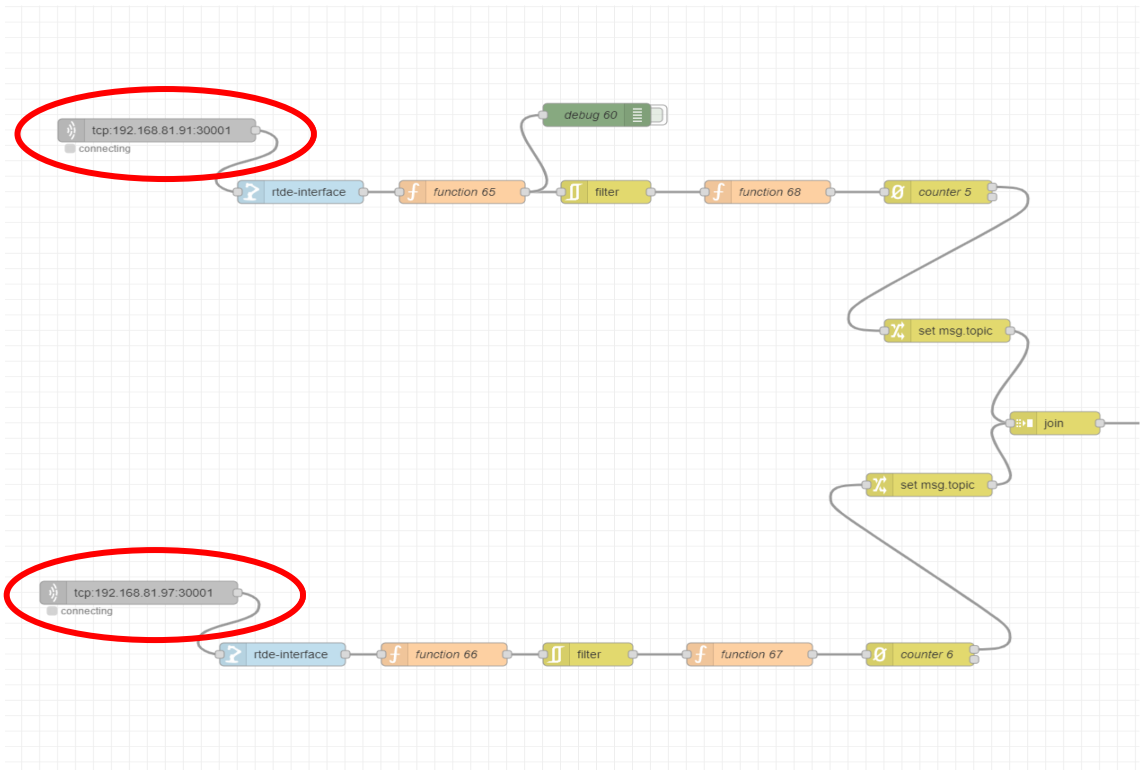
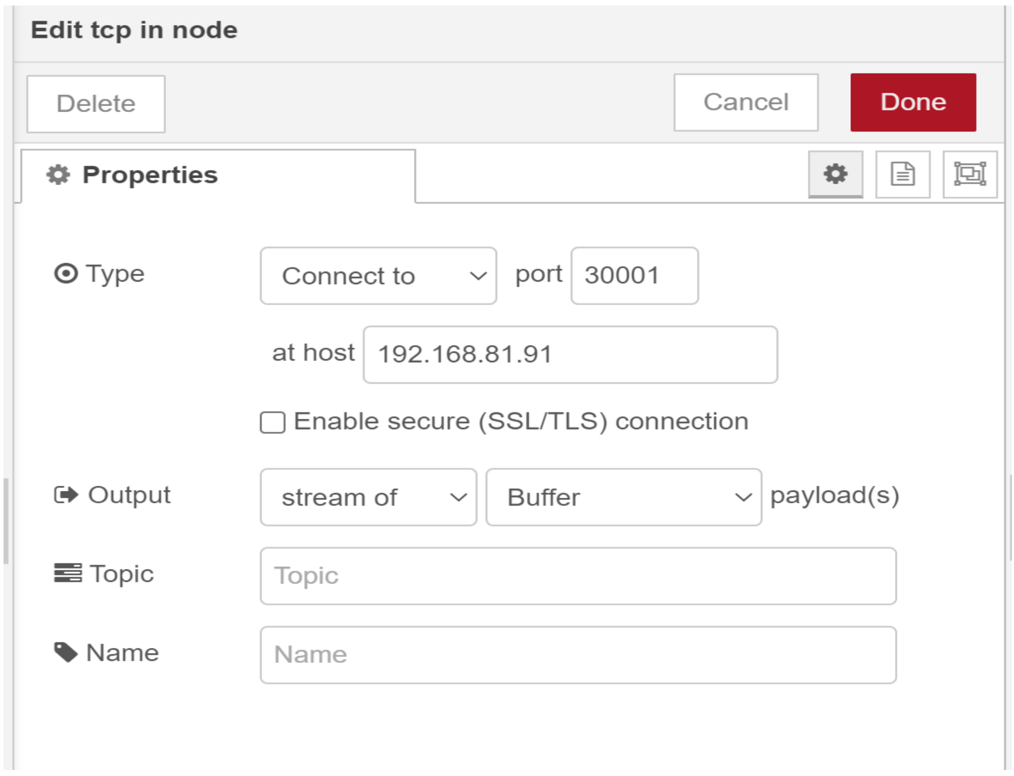


*Figure 3-11 Part of the flow that reads data from robots' PLCs*



Edit tcp in node

Delete | Cancel | Done

Properties

Type: Connect to — port 30001

at host 192.168.81.91

☐ Enable secure (SSL/TLS) connection

Output: stream of — Buffer — payload(s)

Topic: Topic

Name: Name

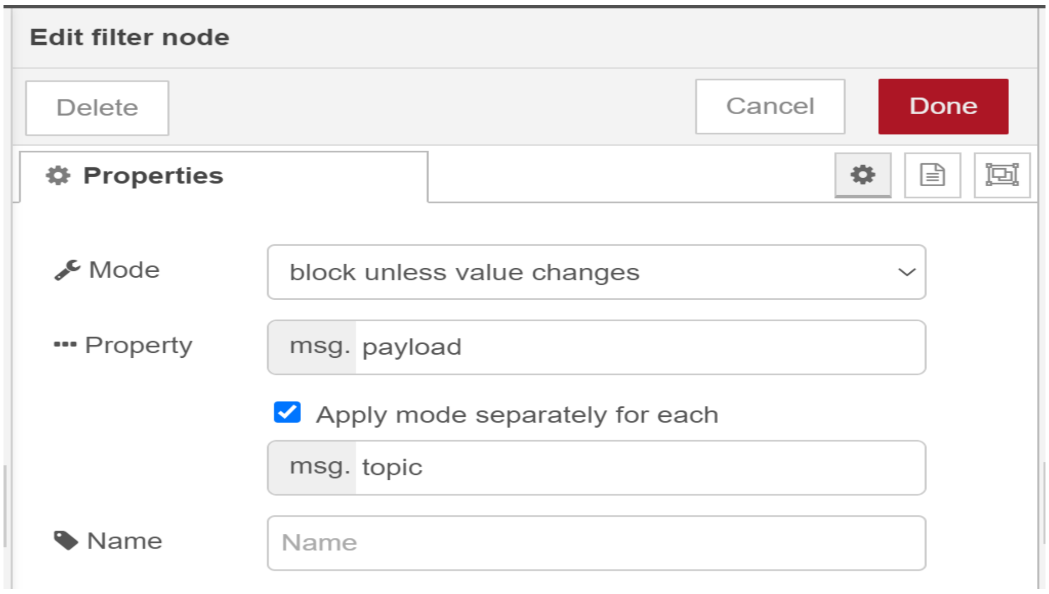
*Figure 3-12 Panel for setting up the TCP-in node that reads data from robots PLC*

In Figure 3-12, instead, it can be seen how the TCP-in node is created. The panel permits to choose the IP address and the port, which are critical to implement the connection, but it also gives the possibility to choose the type of output. In this case a buffer type was chosen in order to obtain a set of data, and then the ones of interest have been selected and collected via the following nodes and functions.

These two TCP-in nodes are the ones that bring Node-Red a set of data from robot 1 and robot 2 respectively. This data set consists of the digital outputs, each representing a specific piece of information. In this case, the digital output selected is DO3, containing values associated with a complete movement performed by the robot. The digital output sent by the robot contains two different values, one, which usually is a number different than 40, during the entire movement phase, and one, equal to 40, at the instant it is in the starting position ready to execute the task all over again. At this point is has been essential the use of a *filter* node (Figure 3-13), which passes a value only when it detects the change from 40 to another number, and vice versa. In this way, the flow receives a single value that warns that the robot has finished its task. In order to simplify the writing of the code, especially in view of the subsequent steps, the filter node that detects a change sends the value 1 to the following functions, which will use it to calculate some of the process variables listed above.

- Process time is calculated through the use of an *Interval length* node, which exploits the DO3 values. Each time it receives the value 1 (which, as said before, is associated to the robot ready in the start position), it starts a timer, that stops at the following value 1. The value returned by this node will then be the time elapsed from the moment the robot starts the task to the moment it finishes it, the process time.
- The number of units processed by each robot is instead calculated through the use of a *counter* node. The counter node has the function of keeping numerical track of each time a user-specified event occurs. In this case, the node, starting from 0, counts each time the value 1 is received from the filter node. The number of times a task is repeated also represents, as a direct consequence, the number

of units processed by each robot, as each movement corresponds to a piece transported.


Edit filter node

Mode: block unless value changes

Property: msg. payload

☑ Apply mode separately for each

msg. topic

Name: Name

Figure 3-13 Use of the filter node in 'block values as long as they remain equal' mode

### 3.4.1.3 Data writing to FlexSim: Modbus Write

Up to now, only the connection part between the physical system and the data collection platform has been introduced. This section will instead show how the data is actually passed to the virtual system, and then displayed graphically by FlexSim. The data transmission method is the same as before, the only difference being that instead of using the IP addresses of the robots, the connection is via the localhost, as both platforms are on the same device.

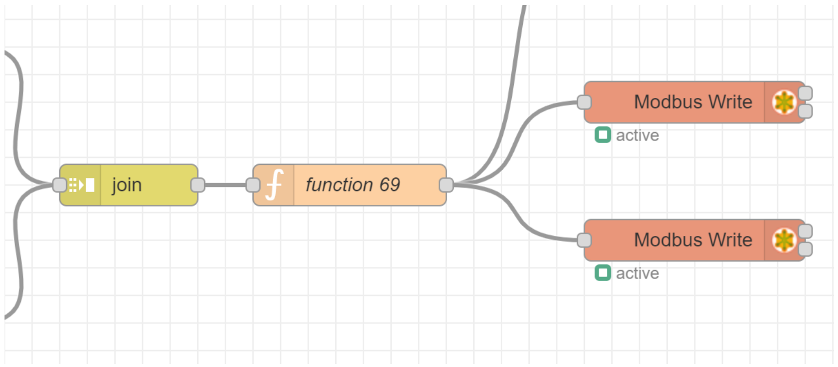
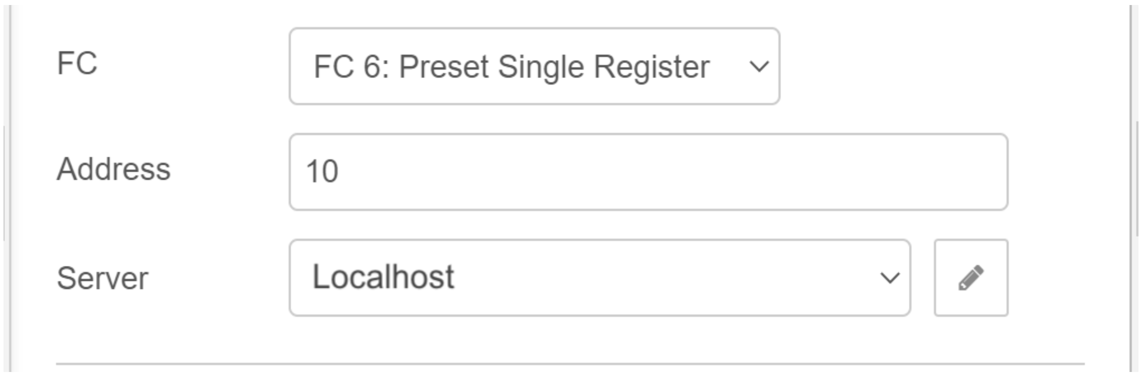


Figure 3-14 Modbus Write nodes that send data to FlexSim


FC: FC 6: Preset Single Register

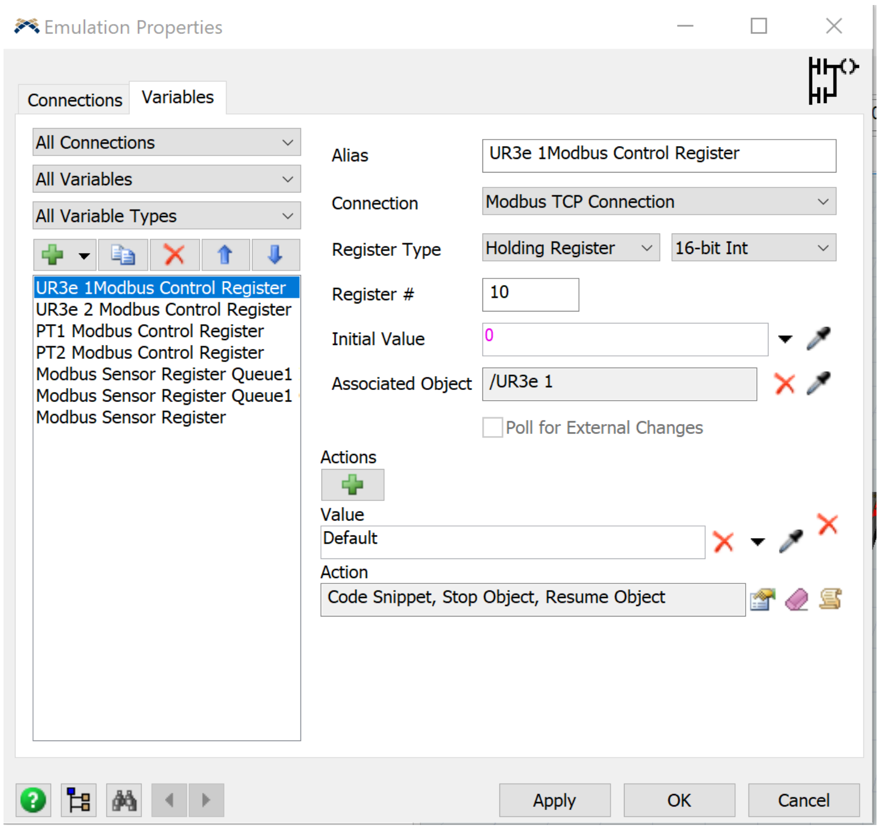
Address: 10

Server: Localhost

*Figure 3-15 Properties panel of the Modbus Write node*

The fundamental node to be used is the one shown in Figure 3-14 and is called the Modbus Write node, while Figure 3-15 shows the main properties that can be set for this kind of connection, such as the *FC*, which represents the class 1 codes discussed in the Modbus introduction section, the *address* which represents the register used to pass a specific data recognized both from master and slave, and finally the *server*, which in this case is the localhost.

For this type of connection, however, it is also necessary to perform certain actions on FlexSim, as the emulator tool must be able to interpret the registers with their data, and thus be able to associate them with a specific object and action. For example, registers 30 and 40 pass the process times of robot 1 and robot 2 respectively to FlexSim. On FlexSim, in the appropriate panel, it will therefore be essential to associate these registers firstly with the correct object, and then also with the specific action with which they are associated. In this case, a short piece of code has therefore been written in this environment as well, so as to associate the value received as input, which on FlexSim is recognized with the variable named 'newValue', and thus make the model understand that this time will be associated with the process time of the machines, a variable that appears in the statistics of each object. Another practical example related, instead of process time, to system status is shown in Figure 3-16. In particular, at the top it can be seen how the connection was created: the type is specified, Modbus TCP Connection, the type of register with its number, and finally the associated object, which in this case is robot 1. At the bottom there is a brief focus on a piece of code which, based on the value received from Node-Red (1 or 2), blocks or starts the model robot.



Emulation Properties

Connections | Variables

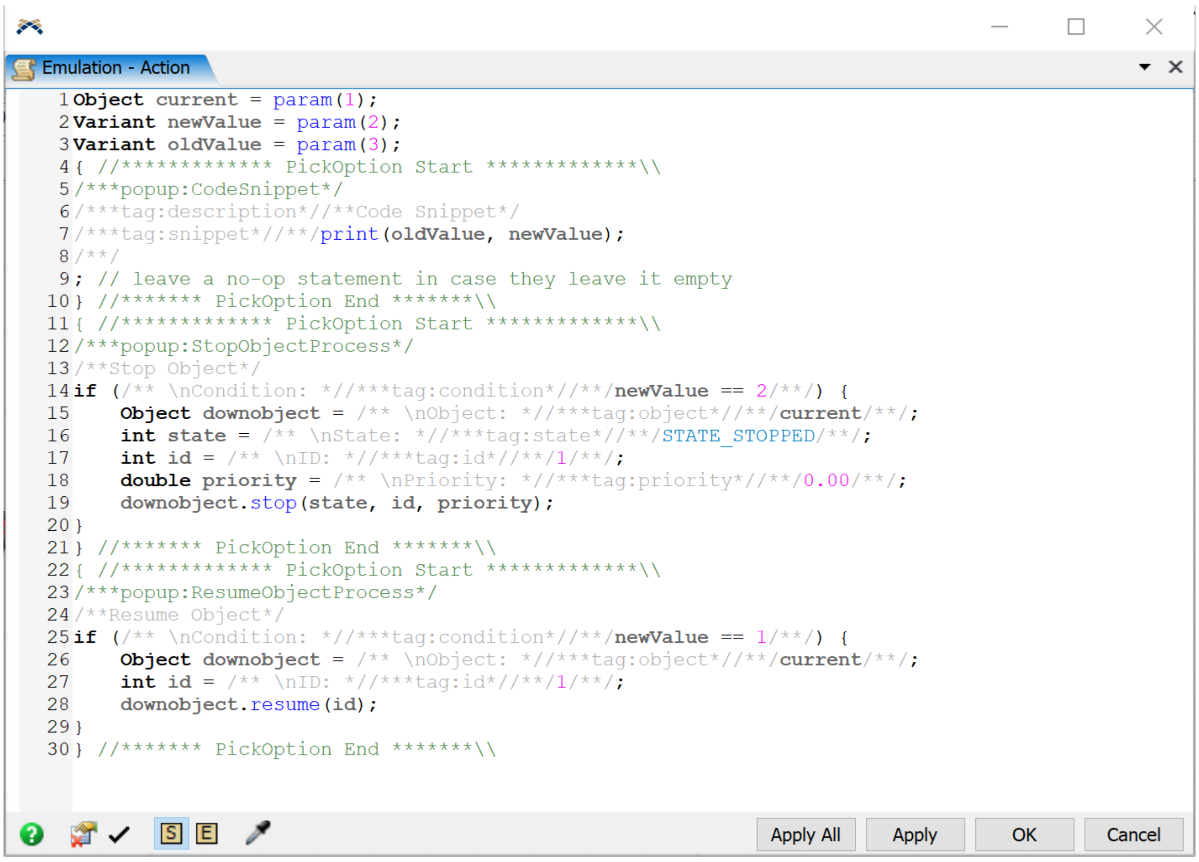
All Connections
All Variables
All Variable Types

UR3e 1Modbus Control Register
UR3e 2 Modbus Control Register
PT1 Modbus Control Register
PT2 Modbus Control Register
Modbus Sensor Register Queue1
Modbus Sensor Register Queue1
Modbus Sensor Register

| | |
|---|---|
| Alias | UR3e 1Modbus Control Register |
| Connection | Modbus TCP Connection |
| Register Type | Holding Register, 16-bit Int |
| Register # | 10 |
| Initial Value | 0 |
| Associated Object | /UR3e 1 |

Poll for External Changes

Actions
Value: Default
Action: Code Snippet, Stop Object, Resume Object



Emulation - Action

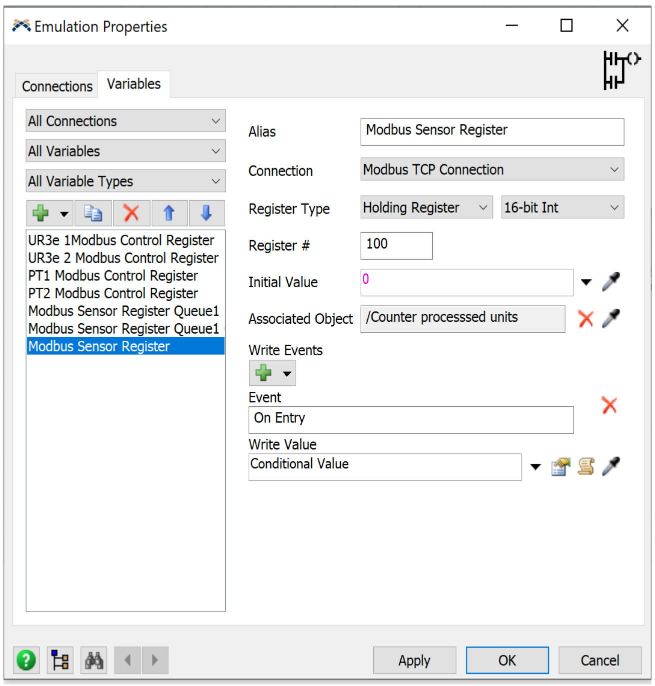
```
Object current = param(1);
Variant newValue = param(2);
Variant oldValue = param(3);
{ //************* PickOption Start *************\\
/***popup:CodeSnippet*/
/***tag:description*//**Code Snippet*/
/***tag:snippet*//**/print(oldValue, newValue);
/**/
; // leave a no-op statement in case they leave it empty
} //******* PickOption End *******\\
{ //************* PickOption Start *************\\
/***popup:StopObjectProcess*/
/**Stop Object*/
if (/** \nCondition: *//***tag:condition*//**/newValue == 2/**/) {
    Object downobject = /** \nObject: *//***tag:object*//**/current/**/;
    int state = /** \nState: *//***tag:state*//**/STATE_STOPPED/**/;
    int id = /** \nID: *//***tag:id*//**/1/**/;
    double priority = /** \nPriority: *//***tag:priority*//**/0.00/**/;
    downobject.stop(state, id, priority);
}
} //******* PickOption End *******\\
{ //************* PickOption Start *************\\
/***popup:ResumeObjectProcess*/
/**Resume Object*/
if (/** \nCondition: *//***tag:condition*//**/newValue == 1/**/) {
    Object downobject = /** \nObject: *//***tag:object*//**/current/**/;
    int id = /** \nID: *//***tag:id*//**/1/**/;
    downobject.resume(id);
}
} //******* PickOption End *******\\
```

*Figure 3-16 Connection creation from the point of view of the FlexSim platform*

### 3.4.2 Virtual-to-Physical information flow

Up to now, the connection between the physical and virtual model has been introduced, in which the data goes from the robots to Node-Red, is processed, and then sent to FlexSim. Now the reverse process will be analyzed in which, on the basis of the data received and analyzed by the virtual model, multiple pieces of information will be sent towards the physical model to suggest actions to be performed during the process. The connection from the virtual to the physical model is based on the same protocols as before, but these must be adapted accordingly, as the data is now an output of the virtual system and an input of the physical one.

The connection between FlexSim and Node-Red will then be introduced first and then, respecting the direction of the data flow, the connection from Node-Red to the robots.


Emulation Properties

Connections | Variables

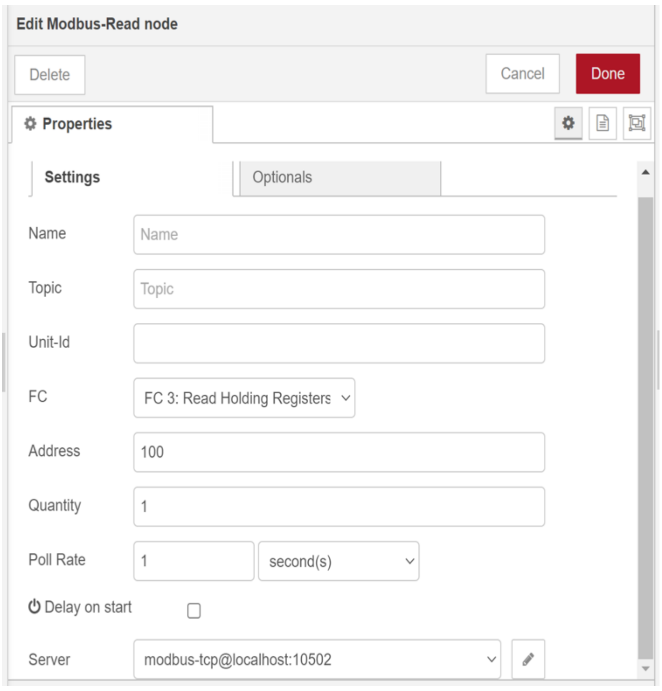
All Connections
All Variables
All Variable Types

UR3e 1Modbus Control Register
UR3e 2 Modbus Control Register
PT1 Modbus Control Register
PT2 Modbus Control Register
Modbus Sensor Register Queue1
Modbus Sensor Register Queue1
Modbus Sensor Register

| | |
|---|---|
| Alias | Modbus Sensor Register |
| Connection | Modbus TCP Connection |
| Register Type | Holding Register / 16-bit Int |
| Register # | 100 |
| Initial Value | 0 |
| Associated Object | /Counter processsed units |
| Write Events | |
| Event | On Entry |
| Write Value | Conditional Value |

Apply | OK | Cancel

Figure 3-17 Modbus Sensor Register that transmits the data about the number of units processed


Edit Modbus-Read node

Delete | Cancel | Done

Properties

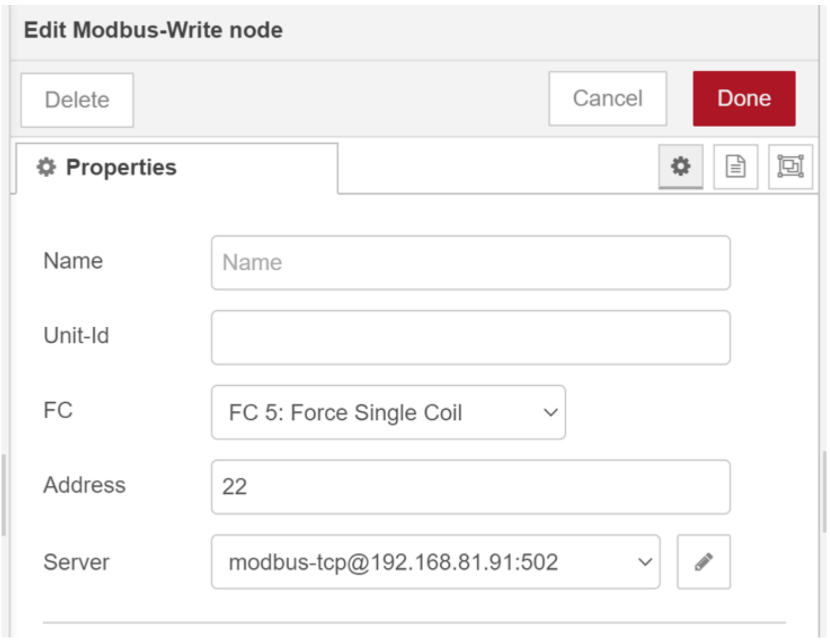
Settings | Optionals

| | |
|---|---|
| Name | Name |
| Topic | Topic |
| Unit-Id | |
| FC | FC 3: Read Holding Registers |
| Address | 100 |
| Quantity | 1 |
| Poll Rate | 1 second(s) |
| Delay on start | ☐ |
| Server | modbus-tcp@localhost:10502 |

Figure 3-18 Modbus Read node to receive information from FlexSim

This first connection makes use of Modbus Sensors Registers, which are PLC's inputs; this means that, when a PLC (or a server) receives a data through a Modbus sensor, that specific value is read from the FlexSim stored values, which are being collected during the execution of the model. In the case of the project in the laboratory, this connection was created to pass on the number of units processed during execution. As can be seen in Figures 3-17, the basic operation based on connection type, register number, and register type is the same as before, it simply occurs in the opposite direction. In this case, an *event* has been defined, via 'conditional value' tool, to specify when the information is to be passed or not: the condition specifies that when the counter, which is the associated object located within the virtual production line, perceives a number of units processed in the system greater than zero, it starts to keep count and gradually passes the value to Node-Red via register. From the point of view of Node-Red, as can be noticed from Figure 3-18, comparing it with the previous figure, the Modbus Read node has been set up as a Reading Holding Register, the address of which is 100, exactly the same as the register number that FlexSim uses to pass the value. At the bottom of the picture, you can see how, again, the connection is made via localhost.

For the connection between Node-Red and the robots it was instead sufficient to change the type of nodes. Whereas until now the Modbus Write node was used to send information to FlexSim, it is now used to send information to the robots, as the connection is the opposite. Figure 3-19 shows an example of a Modbus Write node in which the IP address to which the information is to be sent is specified; obviously in this case it is no longer localhost but the specific address of one of the two robots.



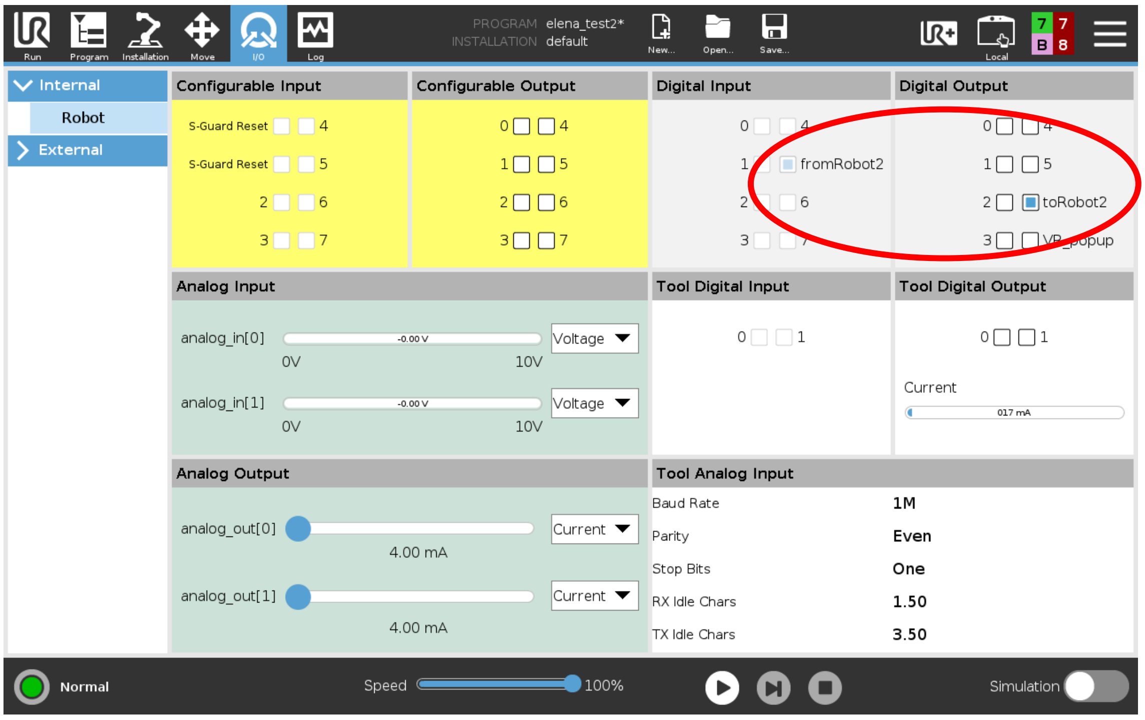
| | |
|---|---|
| Name | Name |
| Unit-Id | |
| FC | FC 5: Force Single Coil |
| Address | 22 |
| Server | modbus-tcp@192.168.81.91:502 |

*Figure 3-19 Modbus Write register addressed to Robot 1*

### 3.4.3 Communication between robots

The communication between the two robots was established with the aim of exchanging messages between them more efficiently using digital input and digital output. The purpose of this type of communication is to set the status of the robot as operational or non-operational, based on a specific piece of data received. In more detail, the robot is activated and deactivated by setting an internal parameter to 'high' or 'low', when receiving the signal in the form of digital input; within the program to be executed by the robot, it is then sufficient to insert a piece of code in order to wait for a value passed through the other robot's digital output.

As illustrated in Figure 3-20, in the I/O control panel of robot 1, it can be seen that the connection to robot 2 is established: among the digital inputs the bit 'fromRobot2' is active, exactly the same as for the signal 'toRobot2' among the digital outputs. What happens when using this type of connection is that from Node-Red a Boolean value is sent through a specific register to Robot 1 which passes the signal to Robot 2 in the form of digital input/output. Although not shown in the image, the same happens in the opposite direction, so passing information from Robot 2 to Robot 1.



PROGRAM elena_test2*
INSTALLATION default

| Configurable Input | Configurable Output | Digital Input | Digital Output |
|---|---|---|---|
| S-Guard Reset ☐ ☐ 4 | 0 ☐ ☐ 4 | 0 ☐ ☐ 4 | 0 ☐ ☐ 4 |
| S-Guard Reset ☐ ☐ 5 | 1 ☐ ☐ 5 | 1 ☐ ■ fromRobot2 | 1 ☐ ☐ 5 |
| 2 ☐ ☐ 6 | 2 ☐ ☐ 6 | 2 ☐ ☐ 6 | 2 ☐ ■ toRobot2 |
| 3 ☐ ☐ 7 | 3 ☐ ☐ 7 | 3 ☐ ☐ 7 | 3 ☐ ☐ VR_popup |

Analog Input
analog_in[0] -0.00 V Voltage 0V 10V
analog_in[1] -0.00 V Voltage 0V 10V

Tool Digital Input
0 ☐ ☐ 1

Tool Digital Output
0 ☐ ☐ 1
Current 017 mA

Analog Output
analog_out[0] Current 4.00 mA
analog_out[1] Current 4.00 mA

Tool Analog Input

| | |
|---|---|
| Baud Rate | 1M |
| Parity | Even |
| Stop Bits | One |
| RX Idle Chars | 1.50 |
| TX Idle Chars | 3.50 |

Speed 100%

*Figure 3-20 Digital Input/Output panel on robot's Teach Pendant*

## 3.5 Logic

Up to this point, all implementations made on a practical level on the physical and virtual system have been introduced, so that the functioning of each entity underlying the project is clear. In this section, however, the logic of the system will be explained; taking into account how the entities are made and how they communicate with each other, the functioning of the models will then be shown at a broader and less detailed level, with reference to what the objectives of this work are, i.e., how the system must behave in order to guarantee flexible scheduling.

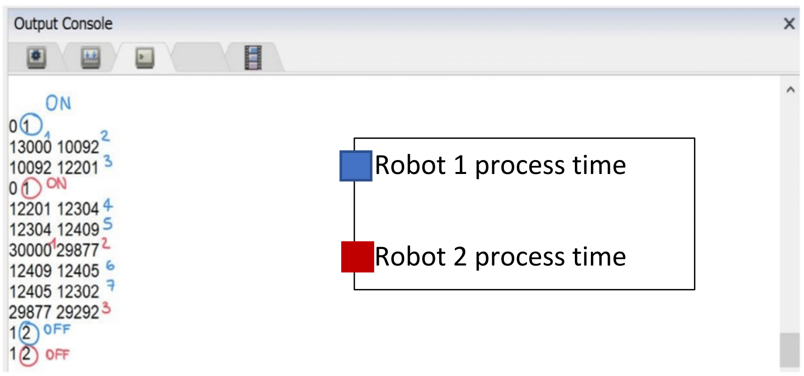
It is important to emphasize that, taking into account the fact that the system must be extremely reactive in order to guarantee instantaneous changes, the largest part of the data collection phase with the subsequent reworking and construction of the logic was done on Node-Red, as it provides a much less restrictive environment than FlexSim, both in terms of code and programming, but also because of the delay in receiving information that cannot be reduced due to force majeure, which will be explained next. The latter was indeed used mainly for the virtual visualization of the process, and for the analysis of the system that can be done ex-post, such as the one related to the machinery utilization, throughput, cycle time and factors like these ones.

As mentioned earlier, it is important to devote a few words to the reason why the entire process was not carried out on FlexSim. The purpose of this type of process is to generate a time-varying scheduling; to do this, it was decided to assign tasks to one robot rather than the other based on a real-time analysis of the process time and the number of tasks executed. The process time, as mentioned above, was calculated with a timer that is activated and then deactivated at the end of the movement of the robotic arm. Node-Red is able to make the decision in real time, as the calculation for the allocation of the remaining tasks takes into account the number of units processed by the critical resource, and as there is no need in that environment for a virtual representation. On FlexSim, however, as digital visualization is necessarily included, the delay becomes unavoidable: the virtual model will execute ‘unit 1’ with the calculated process time when at the same time the real robot is already starting to process ‘unit 2’.

Going beyond the premises just made, the logic of the entire system will now be introduced. At the beginning of the process both the two environments are connected, and FlexSim and Node-Red are on listening and ready to start. A start signal is sent from Node-Red to both robots, which exchange it with each other via digital input and output, and begin processing together each with the relative speed, which varies according to the replicated scenario. The robots are programmed in repeated loops five times, as their mission is to process five units each.

The robots' program is written so that they send messages at specific times of execution. Specifically, at the end of each movement, so at the moment in which the last step of the part of the program contained in the loop is executed, the robots send via digital output a value to Node-Red, which in turn reprocesses it to derive the number of units processed up to that instant and the processing time, using the method explained above.  With the first signal received, Node-Red transforms it into a specific number, 1, so that it can send it to FlexSim to communicate that the process has begun and that, consequently, the digital representation of it should also begin.

Meanwhile, the robots continue to process, continuing to send the signal at each task end. When the first timer stops, the time obtained is instantaneously passed to FlexSim, which through the emulation tool passes it to the virtual machines as process time. The values received from the virtual platform are shown in a specific panel, the *output console*, which prints the old value received, or the first of all, on the left and the next new values on the right. Thus, in each row the values for only one robot of the two are shown.



```
0 1
13000 10092
10092 12201
0 1
12201 12304
12304 12409
30000 29877
12409 12405
12405 12302
29877 29292
1 2
1 2
```

Figure 3-21 Output Console in FlexSim

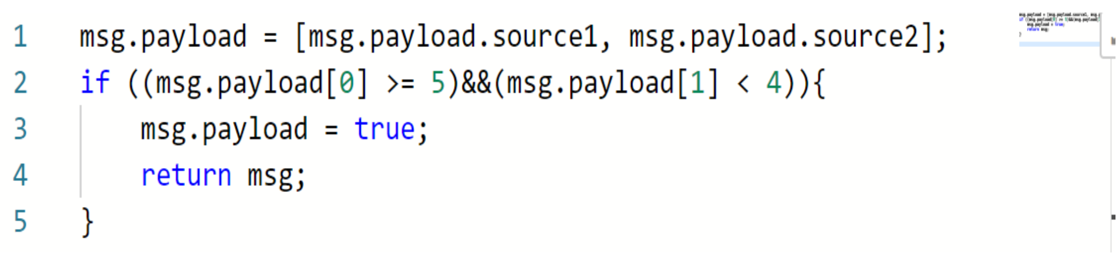
Figure 3-21 shows the FlexSim panel in which the received values are shown: the numbers 0 and 1 show the change of state of the two robots from 'waiting for signal' to 'operational', while the change to number 2, which tends to be at the end of the process unless unexpected breakdowns are avoided, means 'machine stopped'. The other values are the process times expressed in milliseconds.

Returning to the real system, it is now important to differentiate various scenarios that can happen:

- The first one is the best case, in which no time failure occurs, and the two robots process their five assigned units and then stop. In this case Node-Red finds no problem in the received data and consequently will send the stop signal to the robots the moment it counts ten processed units. The program on Node-Red was written to allow a certain margin of error, which is considered tolerable. As slight differences in the execution speed of the two robots may exist in the real case, the program does not recognize a 'time failure' until there is at least a mismatch of an entire processed unit.
- The second scenario is one in which time failure occurs instead. Since the system is monitored in real time, what happens is that Node-Red, thanks to a specially created piece of code, recognizes a delay in task execution by a resource. The code, after one of the two robots has finished its five tasks, evaluates, based on the number of processed units received, whether to have the stopped resource start the loop again. If according to the algorithm this action is necessary, a special node will send a boolean value *'true'* to the robot in order to give it the signal to start the loop from the beginning. This procedure is illustrated in Figure 3-22.

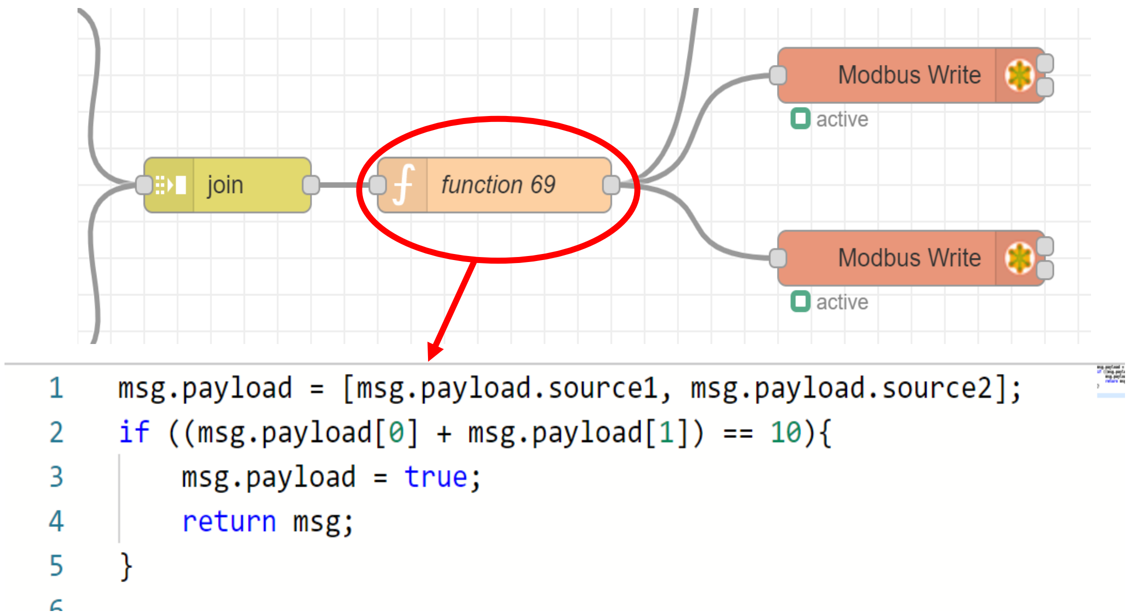
The last action performed by the program is to recognize when it is time to send the stop signal to the robots. In the scenario where no time failure occurs, theoretically the problem should not exist as the two robots are programmed on a cycle of five repetitions. In the case of time failure, on the other hand, there is a need to send a stop to the two robots, as the Robot 1 would otherwise unnecessarily complete a full five-unit cycle, when in reality those needing to be processed might be fewer. Consequently, a Node-Red piece of code was written which, based on the number of processed units it processes moment by moment, evaluates when it is necessary to send the stop signal.

Since the visualization is delayed on FlexSim, it is conceptually incorrect to send the stop signal at the same time as the one sent to the robots. To solve this problem, an object renamed *counter units* was inserted into the virtual environment, with the sole purpose of reading the units output by the two virtual robots and sending the value at each instant of time to Node-Red, which, similarly to the process just described, when it recognizes that all units have been processed, also sends the stop to FlexSim.



```
msg.payload = [msg.payload.source1, msg.payload.source2];
if ((msg.payload[0] >= 5)&&(msg.payload[1] < 4)){
    msg.payload = true;
    return msg;
}
```

Figure 3-22 Node-Red code that detects a time delay and sends a Boolean value 'true'



```
msg.payload = [msg.payload.source1, msg.payload.source2];
if ((msg.payload[0] + msg.payload[1]) == 10){
    msg.payload = true;
    return msg;
}
```

Figure 3-23 Node that sends the stop message and its code

# 4 Analysis of scenarios and data collected

As mentioned in the previous chapters, three different scenarios will be presented in order to have a basis for comparison between the scenario with the integration of the digital twin, which was processed in the laboratory, and two others which were processed theoretically, representing the bast case and the worst case.

In this chapter, the two limiting cases will be analyzed in more detail, providing an a priori scheduling linked to the execution of the tasks with the corresponding process times (also estimated a priori, based on the process times of the robots should they process at 100% of their capacity). For each of these scenarios will then be calculated:

- TH: the throughput of the line, which represents the number of items processed by the system per unit of time
- CT: the average cycle-time of the line, which represents the actual time required to process all the units in the system, in this case the 10 units
- Utilization of the robot: it is associated with the fraction of time the machine is in use (taking into account any setup or breakdowns, as well as the simple time it processes).

  In this content, in the practical case with the application of DT technology, one should notice an increase in utilization for Robot 1, which will have more units to process than in the other cases.

Based on the collected data and calculated variables, a comparison will then be made at the end of the chapter; the expected result of the laboratory test, if completed correctly, should represent an intermediate solution between the best and worst case. This should happen because, firstly, it is not possible, in the presence of a time failure linked to one of the two robots, to have a performance such as to completely cancel out the delay, and consequently such as to guarantee results equal to the best case. At the same time, thanks to the application of the digital twin, which aims to 'mitigate' damages in the system, the total time for the process of the entire batch in dynamic conditions should be better than in the case in which the DT system is absent (non-flexible scheduling conditions).

## 4.1 Best Case scenario: no time failure

This scenario represents the best possible outcome related to the production line. The two robots are identical in form, equipped with the same tools (a gripper attached to the last joint), and have the same characteristics, so it can be assumed that under optimal conditions their speed, and therefore their process time, will also be the same, within a range of uncertainty due to natural variability.

The process time considered will therefore be 13,000 milliseconds per item, when no time failure occurs. Consequently, the fixed allocation of tasks to be accomplished will be as follows:

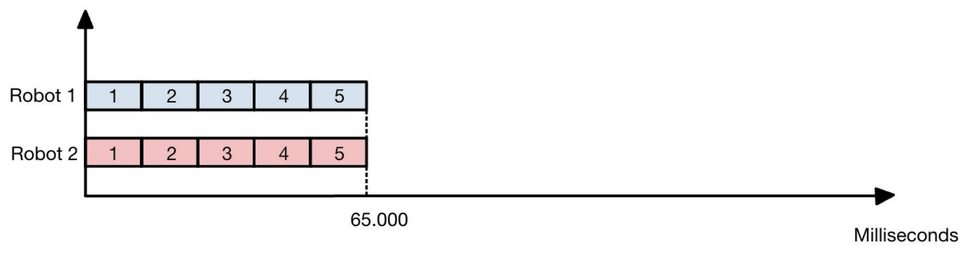


*Figure 4-1 Fixed Scheduling related to the Best-Case scenario*

As it can be seen, there is no bottleneck in this process, due to the absence of a slower resource. The total estimated time for processing the ten units of the batch is therefore 65,000 milliseconds, which corresponds to the processing time of a single unit multiplied by the five units assigned to each robot. Next, in Figure 4-2, it will be shown the FlexSim dashboard, obtained by running the simulation on the same system used in the practical case, but assigning process times a priori.

In the collection of results, the variables introduced earlier can be found: throughput is represented in the box named *'composite throughput per hour'*, and cycle-time is instead indicated as *'Composite Staytime'*, which is evaluated as the average staytime related to both robots.

The state bars box in Figure 4-2 indicates the time the robots were actually operating out of the total time. In this case, as there is no disparity in the process of the assigned units, the state bar will be 100% for both.

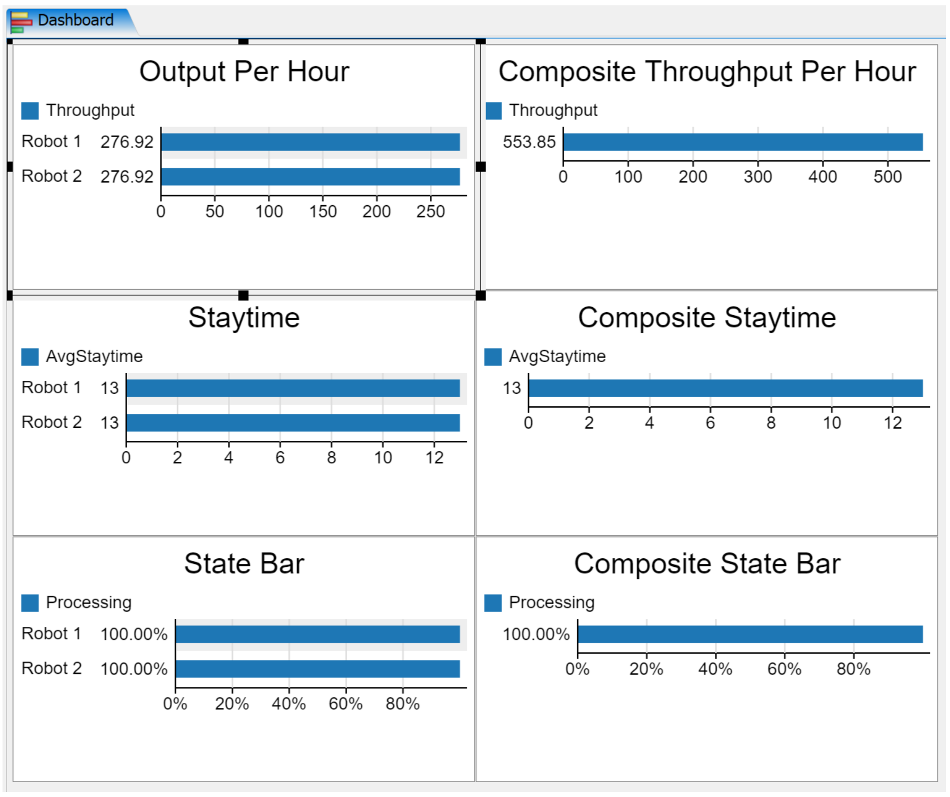


*Figure 4-2 Dashboard on FlexSim for the Best Case Scenario*

## 4.2 Worst Case scenario: time failure without DT detection

The worst-case scenario represents the worst possible outcome, in this case following a time failure. In this case, the situation presented involves one of the two robots, namely Robot 2, processing the units with a significantly longer process time than Robot 1.

It has been estimated that the process time of Robot 2 under these conditions is 2.5 times longer than expected, thus being around 32.500 milliseconds.

The peculiarity of this scenario is the absence of the digital twin. The absence of this technology means that when a time failure occurs, there is nothing capable of detecting it, and therefore that no corrective action is planned to mitigate the situation.

Next, a flexible task allocation scheduling will be proposed when Robot 2 turns out to be slower; as can be seen in Figure 4-3, as there is no DT system to handle the reallocation of tasks in the line, Robot 1 will process the 5 units planned within its pre-scheduled loop, and so will Robot 2, but taking more than twice as long.

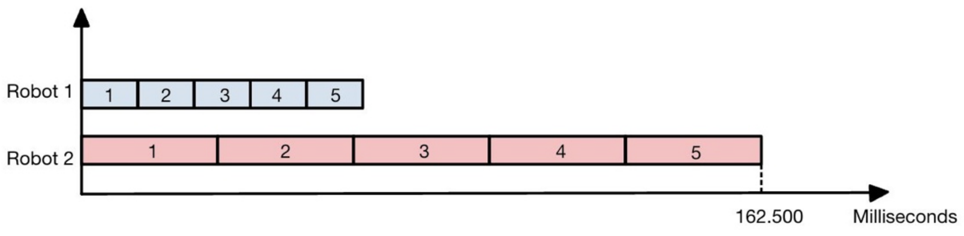


*Figure 4-3 Fixed Scheduling for the Worst-Case scenario*

In this case, it can therefore be seen that the second resource, UR3e 2, represents a bottleneck in the production line, bringing the total process time from the 65.000 milliseconds of the best-case scenario to 162.500 milliseconds, due to the lack of response to changes in the physical system.

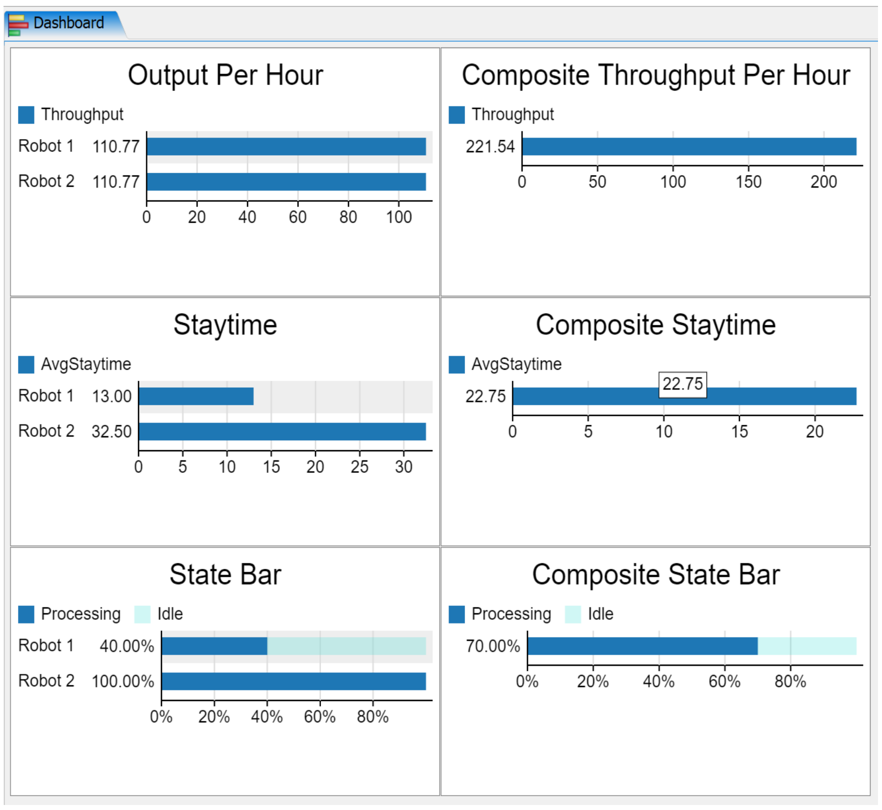


*Figure 4-4 Dashboard on FlexSim for the Worst Case Scenario*

Figure 4-4 shows the dashboard for the worst-case scenario, structured in the same way as the one seen above. Starting with the analysis of throughput, it can be seen that it is the same despite the differences in production times between the two robots. The value 110,77, expressed in items per hour, is in fact derived from the bottleneck resource, which in this case exists and is represented by robot 2. The slowest resource among those that constitute a production line is in fact the one that defines the hourly production volumes. The system throughput will therefore be 221,54 items per hour, and takes into account the contribution of both robots. Robot 1 could have an extremely higher throughput but, due to the fixed scheduling, when it finishes processing the 5 units assigned to it, it remains unused until Robot 2 finishes its batch. Based on theoretical concepts, at this point it can be predicted that, most likely, with the use of flexible scheduling, one could exploit the unused Robot 1 to assign some units of the slower robot to it, in order to increase the final throughput of the system.

The composite average staytime is 22.750 milliseconds per item, as this is an average calculated over the lot size. With regard to the state of the machinery, it can be seen that the utilisation of Robot 1 is drastically lower than that of Robot 2, as after processing its part of the batch of 5 units, the robot that works as intended remains unused until the end of the process.

## 4.3 Practical Worst Case: time failure detection through DT system

The worst practical case is the scenario that occurs when, in the model run in the real system, the time failure that occurs is detected by the Digital Twin system and consequently changes are implemented to the behavior of the model in order to minimize the negative consequences.

This scenario has been proposed in the laboratory following the logic introduced in the previous chapter: in fact, it will be observed that, following the failure, the speed of Robot 2 will be approximately 2.5 times less than that of Robot 1, which will instead remain unchanged with respect to the conditions predicted a priori.

Thanks to the operation of the Digital Twin explained above, the results of this practical case shown below will confirm the initial hypothesis, i.e. that although it is impossible to obtain results equal to those of the best-case, the variables obtained at the end of the experiment, obviously considering time failure conditions, will be better than those of the theoretical wort-case.

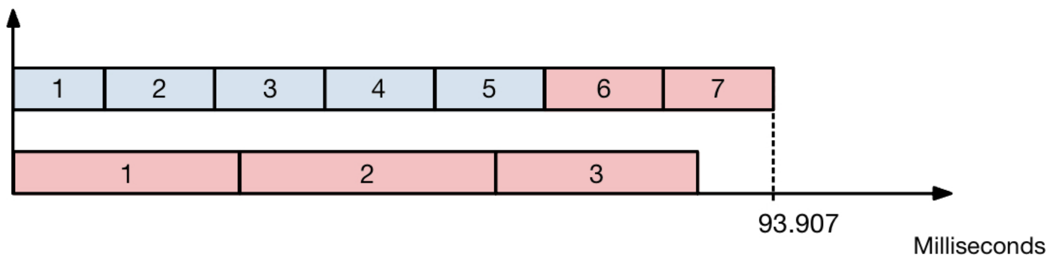


*Figure 4-5 Flexible scheduling obtained by applying the DT to the model*

*Table 4-1 Collected process times and average process times in milliseconds*

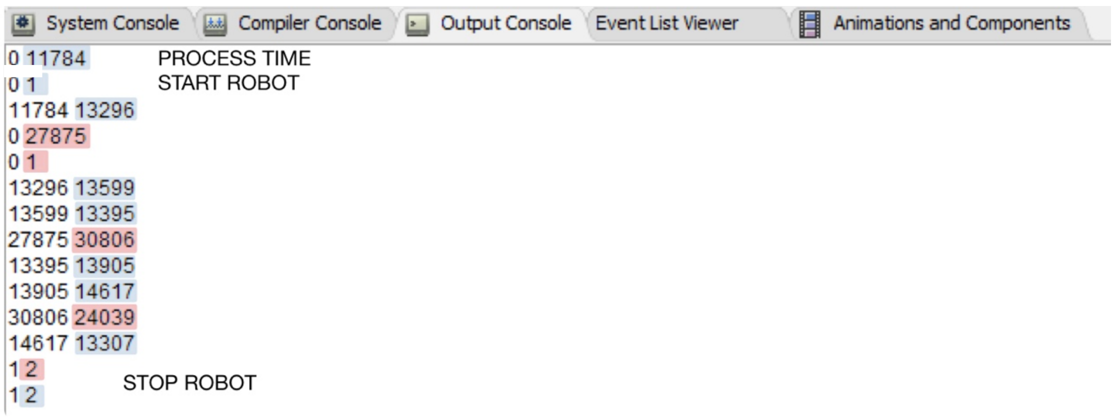
| | unit 1 | unit 2 | unit 3 | unit 4 | unit 5 | unit 6 | unit 7 |
|---|---|---|---|---|---|---|---|
| Robot 1 | 11784,6681 | 13296,7121 | 13599,8162 | 13395,6975 | 13905,3247 | 14617,7153 | 13307,4644 |
| Robot 2 | 27875,739 | 30806,8568 | 24039,2658 | | | | |

| | Avg process time |
|---|---|
| Robot 1 | 13452,3426 |
| Robot 2 | 27573,9539 |

In Figure 4-5 it can be noticed how the scheduling changed due to the real-time detection of the change in the physical system. The scheduling shown in the picture is representative of the data collected during the launch of the test in the laboratory, shown in Table 4-1. It can be seen that in this case the critical resource is Robot 1, which determines the total process time of 93.907,3983 milliseconds.

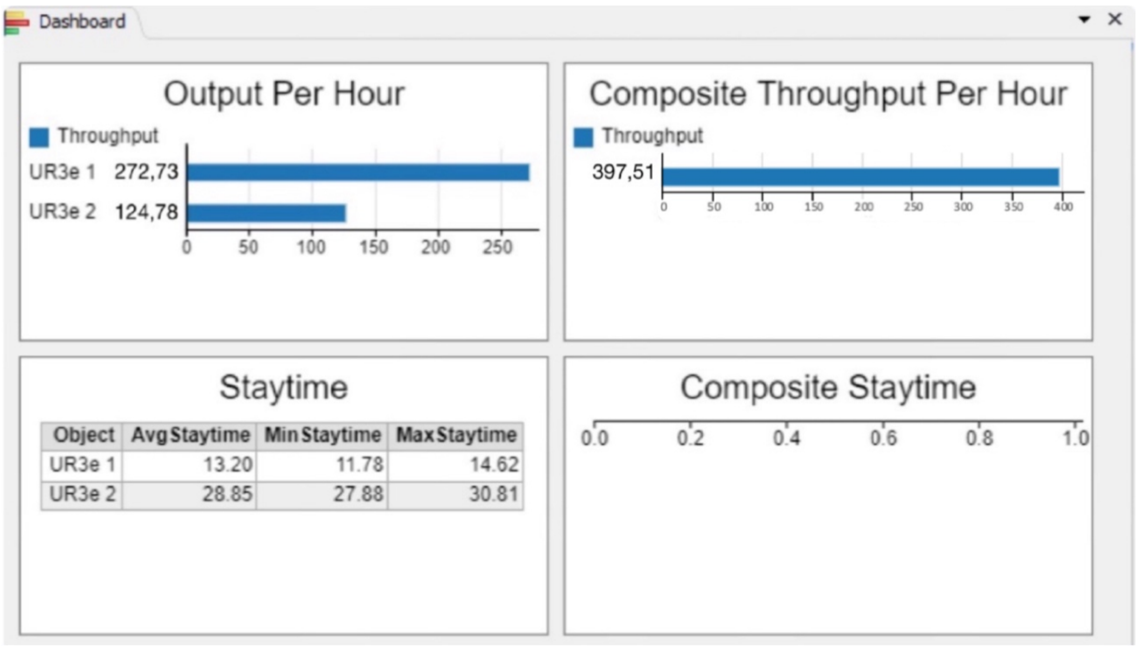
What can be seen from the various tests carried out in the laboratory, however, is that the variability of process times for each individual task is rather high for both, but particularly for Robot 2. Analyzing the average process times, it can be seen that the average time related to the machine two is the one that deviates the most from the values predicted a priori. In this case, in fact, the process time related to Robot 2 is not 2.5 times that of Robot 1, but only 2,06 times.

### 4.3.1 FlexSim Dashboard for the Practical Case



```
0 11784        PROCESS TIME
0 1            START ROBOT
11784 13296
0 27875
0 1
13296 13599
13599 13395
27875 30806
13395 13905
13905 14617
30806 24039
14617 13307
1 2
1 2        STOP ROBOT
```

*Figure 4-6 Output console on FlexSim with the values received during the process*



| Object | Avg Staytime | Min Staytime | Max Staytime |
|---|---|---|---|
| UR3e 1 | 13.20 | 11.78 | 14.62 |
| UR3e 2 | 28.85 | 27.88 | 30.81 |

*Figure 4-7 Dashboard on FlexSim*

The process dashboard on FlexSim, which collects the data received from the physical system, will now be illustrated. This data collection will have to be analyzed more specifically as it is certainly error-prone due to the structure of the system. As introduced earlier, the replication of the physical process on the platform is delayed by a whole process time; this problem cannot be solved as it is necessary for the physical robot UR3e to make a whole movement before transmitting the time taken. An a priori estimate would in fact render the entire project a mere simulation, and not an emulation of the real system.

The final dashboard will then provide values that deviate slightly from those observed in Node-Red, merely because the reception of delayed values of an entire process time, relative to each robot, means that on FlexSim the process does not start at the same time for both machines, thus leading to idle times for one of them that do not correspond exactly to the real ones. These results will be taken up in one of the next sections, so that they can be compared with those collected on the Node-Red platform to verify their accuracy.

### 4.3.2 Node-Red Dashboard for the Practical Case

Node-Red's dashboard, unlike that of FlexSim which has just been introduced, should not be subject to errors (except for the intrinsic margin of error due to measurement, which is in any case negligible) as it receives the values directly from the real system, and thus from the two robots at the end of each task completed.

On Node-Red's platform, as the real system proceeds with the execution of the process, the times associated with the completion of a task, start signals when Robot1's intervention is required, and stop signals when a total of 10 processed units is detected, are evaluate through the use of the timer function, and then transmitted.

Node-Red, differently from FlexSim, which was created for the purpose of analyzing and simulating industrial processes, does not have a predefined set of process variables. Hence, while on FlexSim there is a palette of output visualizers, which by default includes the Throughput and Cycle-Time graphs of the process, on Node-Red the entire calculation must be programmed.

- Throughput

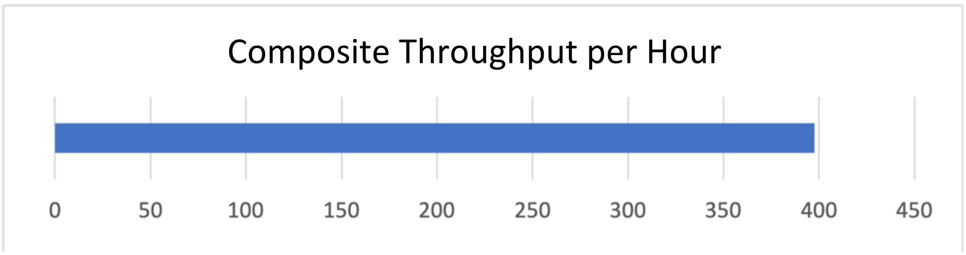


*Figure 4-8 Composite Throughput per Hour*

The throughput calculation consists of obtaining the number of items that, on average, are produced by the system in a unit of time; In this case, both throughput per hour and throughput per second will be shown in Table 4-1, the first one to ensure an effective comparison with FlexSim, and the second one because it is more consistent with the system's timing.

The first throughput calculated is for each machine, based on the process times received. For Robot 1, the average process time is 13415,3426 milliseconds, while for the Robot 2 the average process time is 27573,9539 milliseconds. Thus, the volume produced in one hour by each machine will be equal to:

*Table 4-2 Throughputs related to each robot*

| | items per hour | items per second |
|---|---:|---:|
| Robot 1 | 268,35 | 0,0745 |
| Robot 2 | 130,56 | 0,0363 |

Consequently, deriving the compound throughput per hour, it is equal to 398,9075 items per hour.

- Cycle Time

By analyzing the data collected on Node-Red at the end of the process, it is possible to derive not only the average and relative stay times of each individual machine, but also the cycle time of the complete system, which is useful for describing the time taken (on average) by the ten units of the batch to be processed.

*Table 4-3 Staytime of the two robots*

| Staytime in seconds | |
|---|---:|
| Robot 1 | 13,415 |
| Robot 2 | 27,574 |

### 4.3.3 Comparison of results

In order to prove that the model is a coherent representation of what happens in the real system, it must be verified that there is a correspondence between the number and order of task execution, and between the system variables.

Beginning with task execution, it was confirmed that the processing order of the items on FlexSim, despite the delay, is consistent with the order of execution in the physical system; indeed, both models show seven units processed by robot 1, and 3 by robot 2, in the same chronological order at the end of the process.

As previously mentioned, the performance results of the entire system in terms of Throughput and Cycle Time will now be compared to verify that, despite the interference caused on the data by the structure of the FlexSim model, the results are still consistent and contained within a tolerable margin of error.

*Table 4-4 Comparison between FlexSim and Node-Red*

| | | Node-Red | FlexSim |
|---|---|---|---|
| Throughput | [item/s] | 0,1108 | 0,1104 |
| Cycle Time | [s] | 90,55 | 90,64 |

The data shown in Table 4-4 are for the specific system, which processes ten units. As can be seen, the model shows slight differences in total performance, attributable to the reasons listed above such as the delay in receiving the process time, which creates a time lag between the two models. More specifically, there is a variation of 0.09 seconds for every 10 units processed, which corresponds to the 0,1 percent of the Cycle Time. This absolutely negligible value demonstrates a certain consistency of the model.

# 5 Conclusions

The case study concerning the application of a digital twin to a production line was implemented with the intention, not so much of creating an exact copy in geometry and movements of the two robots, but rather a virtual system rather faithful to the physical one, which was however able to recognize variations and suggest changes in real time, with the aim of remedying any time failures of the line.

The main objective was to demonstrate that *flexible scheduling* could be achieved, capable of reallocating tasks dynamically between different resources to achieve optimal results in critical situations. The results obtained provide strong evidence to prove the assumptions made a priori: it can be seen that, firstly, the scheduling of the Worst Practical Case scenario is a viable solution to time failure in terms of managing the remaining tasks, and in terms of process variables. As a demonstration of this, in fact, it can be seen that in the three various scheduling scenarios proposed, the one in the practical case is in the middle, with the total process time of the ten units clearly reduced, proving that, in the failure situation presented, it is clearly better than the Worst Case scenario, analyzed without the implementation of the Digital Twin.

In conclusion, it is therefore possible to state that various benefits can be achieved through the application of a Digital Twin system in manufacturing. Among the main ones, it is worth mentioning improvements in terms of performance, increasing the use of resources that, under normal conditions, would have a lower utilization since they would not be used to compensate for a delay in production, and above all improvements in terms of time, since the non-recognition of a time failure would entail negative consequences in economics terms for the companies.

It would therefore be good, with a view to future implementations, to continue testing it on additional machines, thus expanding the production line, in order to have an overview to fully understand its potential, which has so far been explored in a narrower environment.

## References

[1] Yibing Li, Zhiyu Tao, Lei Wang, Baigang Du, Jun Guo, Shibao Pang – Digital twin-based job shop anomaly detection and dynamic scheduling, Elsevier Ltd, 21 August 2022.

[2] Scot Kim, 21 Lessons From Successful Digital Twin Implementations for Manufacturing, Gartner, 21 December 2021.

[3] Simon Jacobson, Janet Suleski, Hype Cycle for Manufacturing Operations Strategy, Gartner, 2022.

[4] Alfonso Velosa, Quick Answer: 4 Technical Prerequisites for Successful Digital Twins Implementation in Manufacturing, Gartner, 9 March 2022.

[5] Fuquiang Zhang, Junyan Bai, Dongyu Yang, Qiang Wang – Digital Twin data-driven proactive job-shop scheduling strategy towards asymmetric manufacturing execution decision, Scientific Reports, 2022.

[6] Qi Yan, Hongfeng Wang, Fang Wu – Digital Twin enabled dynamic scheduling with preventive maintenance using a double layer Q-learning algorithm, Elsevier Ltd., 2022.

[7] Douxi Yan, Weinan Sha, Dwen Wang, Jiafreng Yang, Shenghui Zhang – Digital twin driven variant design of a 3C electronic product assembly line, Scientific Reports, 2022.

[8] Frank Siqueira, Joseph G. Davis – Service computing for Industry 4.0: State of the Art, Challenges, and Research Opportunities, ACM Computing Surveys, October 2021.

[9] Ping Chong Chua, Seung Ki Moon, Yen Ting Ng, Huey Yuen Ng – A surrogate model to predict production performance in Digital Twin-based Smart Manufacturing, Journal of Computing and Information Science in Engineering, June 2022.

[10] Jiming Li, Yingfenz Zhang, Cheng Qian – The enhanced resource modeling and real-time transmission technologies for Digital Twin based on QoS considerations, Elsevier Ltd, December 2021.

[11] Ding Zhang, Jiewu Leng, Min Xie, Hong Yan, Qiang Liu – Digital twin wnabled optimal reconfiguration of the semi-automatic electronic assembly line with frequent changeovers, Elsevier Ltd, 19 March 2022.

[12] Robert Woitsch, Anna Sumereder, Damiano Falcioni – Model-based data integration along the product and service lifecycle supported by digital twinning, Elsevier Ltd, 7 April 2022.

[13] Robert Kazala, Slawomir Luscinski, Pawel Straczynski, Albena Taneva – An Enabling Open-Source Technology for Development and Prototyping of Production Systems by Applying Digital Twinning, MPDI, 23 December 2021.

[14] Alvaro Garcia, Anibal Bregon, Miguel A. Martinez-Prieto - Towards a connected Digital Twin Learning Ecosystem in manufacturing: Enablers and challenges, Elsevier Ltd, 14 July 2022.

[15] Jakob Bonsch, Matthes Elstermann, Andreas Kimming, Jivka Ovtcharova – A subject-oriented reference model for Digital Twins, Elsevier Ltd, 9 August 2022.

[16] Maulshree Singh, Rupal Srivastava, Evert Fuenmayor, Vladimir Kuts, Yuansong Qiao, Niall Murray, Declan Devine – Applications of Digital Twin across Industries: a Review, MDPI, 4 June 2022.

[17] Hainfan Jiang, Shengfeng Qin, Jianlin Fu, Jian Zhang, Guofu Ding – How to model and implement connections between physical and virtual models for digital twin application, Elsevier Ltd, 2 June 2020.

[18] Mohammad Azarian, Hao Yu, Wei Deng Solvang – A Simulation-Based Approach for Improving the Performance of a Manufacturing System, IEEE, 11 January 2021.

[19] Wang Y. R., Chen A. N. – Production logistics simulation and optimization of industrial enterprise based on FLEXSIM, 2016.

[20] Shahd Yaser, Nour Abdelatif, Irene Fahim, Youssef Emad, Abdelrahman Saleh, Sally S. Kassem – FlexSim Simulation to Enhance Productivity of a Production Cell: A Case Study, 2021.

[21] https://www.flexsim.com/plc-emulation/

[22] https://docs.flexsim.com/en/21.2/Reference/Tools/Emulation/Emulation.html

[23] https://nodered.org/docs/user-guide/

[24] https://www.universalrobots.com/articles/?filter_Manuals[]=98757&filter_Manuals[]=98758&filter_Manuals[]=98761

[25] https://www.universal-robots.com/products/ur3-robot/

[26] https://www.modbus.org

## List of Figures

## List of Tables