Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Development and implementation of a framework for Model Predictive Control of discrete-event manufacturing systems

Andrea Cataldo

andrea.cataldo@stiima.cnr.it

Consiglio Nazionale delle Ricerche https://orcid.org/0000-0002-6862-8916

Riccardo Scattolini

Politecnico di Milano Dipartimento di Elettronica e Informazione: Politecnico di Milano Dipartimento di Elettronica Informazione e Bioingegneria

Daniele Gottardini

Politecnico di Milano Dipartimento di Elettronica e Informazione: Politecnico di Milano Dipartimento di Elettronica Informazione e Bioingegneria

Research Article

Keywords:

Posted Date: November 6th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11155395/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Development and implementation of a framework for Model Predictive Control of discrete-event manufacturing systems

Andrea Cataldo[1*], Riccardo Scattolini[2†] and Daniele Gottardini[2†]

[1*]Institute of Intelligent Industrial Technologies and Systems for Advanced Manufacturing (STIIMA), National Research Council (CNR), Via A. Corti, 12, Milan, 201330, Italy.
[2]Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB), Politecnico di Milano, Via Giuseppe Ponzio, 34, Milan, 20133, Italy.

*Corresponding author(s). E-mail(s): andrea.cataldo@cnr.it;
Contributing authors: riccardo.scattolini@polimi.it;
gottardinidaniele@gmail.it;
†These authors contributed equally to this work.

Abstract

This paper presents an open-source framework for Model Predictive Control (MPC) tailored for discrete-event manufacturing systems. Traditional MPC requires extensive mathematical modeling, which limits flexibility and adaptability. This research introduces a topology-driven approach where MPC controllers are synthesized automatically based on plant structure rather than on detailed equations. The framework, designed in Python and utilizing Google's CP-SAT solver, addresses the automatic synthesis of an MPC controller based on the latest available information on the plant's layout and state. It is adaptive and reconfigurable, meaning that it self-adapts to changes in the plant without the need of manual configuration. This topology-driven MPC offers a scalable and cost-effective solution to meet the demands of dynamic manufacturing environments.

Keywords: Model Predictive Control, Discrete-event systems, Manufacturing systems, Framework

1 Introduction

In the process industry, e.g., chemical, petrochemical, pulp and paper, Model Predictive Control (MPC), see [1], [2], has been recognized since the 1980's as the most effective methodology for the design of advanced control systems. More recently, the success of MPC has extended to the fields of robotics [3], swarm optimization [3], and avionics [4]. The reasons for the popularity of MPC are many. Underlying them, however, is the possibility of reformulating the control problem as an optimization one, in which to include explicit knowledge of the plant, constraints on process variables, as well as of considering different objectives in defining the cost function to be optimized, and the possibility of using models derived from simple plant tests. The advantages of MPC could be equally important in manufacturing industry, typically represented by discrete-event models. In fact, also in this case, reducing production time, energy consumption and enhancing sustainability would lead to terrific advantages for the modern industry. Various contributions have been proposed in this area in recent years for the solution of production scheduling problems, [5], [6],[7], for supply chain optimization [8], and for optimal routing in complex transport lines, see [9], [10].
Despite the notable exceptions reported in the literature, the use of MPC to control manufacturing systems still remains a challenging issue, and has been severely limited by the difficulty to develop simple and reliable mathematical models due to their inherent combinatorial characteristics, and to the presence of many logical constraints. For these reasons, the optimization problems to be solved at each event according to a moving horizon strategy, are particularly critical even for systems with small sizes.
A systematic procedure to handle these problems has been presented in [5], [9], where logical relations among variables have been transformed into booleans by means of the rules of propositional calculus. Then, a Mixed Logical Dynamical Model (MLD) of the system is obtained [11], and finally optimization is performed according to a Receding Horizon strategy. However, although this approach has proven to be effective also in real systems, see,[9] its implementations have been custom-oriented, rather than systematic, making it difficult to have a widespread diffusion of this approach.
The goal of this paper consists of describing the development of an open-source framework for Model Predictive Control (MPC) tailored for discrete-event manufacturing systems. Traditional MPC requires extensive mathematical modeling, which limits flexibility and adaptability. This research introduces a topology-driven approach where MPC controllers are synthesized automatically based on plant structure rather than on detailed equations. The framework, designed in Python and utilizing Google's CP-SAT solver, together with the adoption of a typical advanced control system design structured methodology, addresses the automatic synthesis of an MPC controller based on the latest available information on the plant's layout and state. It is adaptive and reconfigurable, meaning that it self-adapts to changes in the plant without the need of manual configuration. This topology-driven MPC offers a scalable, adaptive, and cost-effective solution to meet the demands of dynamic manufacturing environments.
The paper is structured as follows: in Section 2 the open-source framework architecture and the applied structured methodology for the design of the MPC algorithm are described. Section 3 details the elements that make up the framework and the

way to use them (paragraph 3.1 - 3.3), in addition to the actuation of the framework in terms of automation system (paragraph 3.4). In Section 4 simulation results are reported to assess the performance of the proposed methodology. Finally, some conclusions are drawn in Section 5, where also some hints for future hardware/software improvements are discussed.

2 Structured methodology for MPC and open-source framework architecture

The MPC approach for control design of discrete events systems basically consists in computing the future evolution of the control variable along a specified prediction horizon by solving an optimization problem, at any new discrete event, and given the knowledge of the system's state, see figure 1. The first value of the optimal solution is then applied to the plant and, at the next event, the overall procedure is repeated according to the so-called Receding Horizon (RH) approach, see [2].

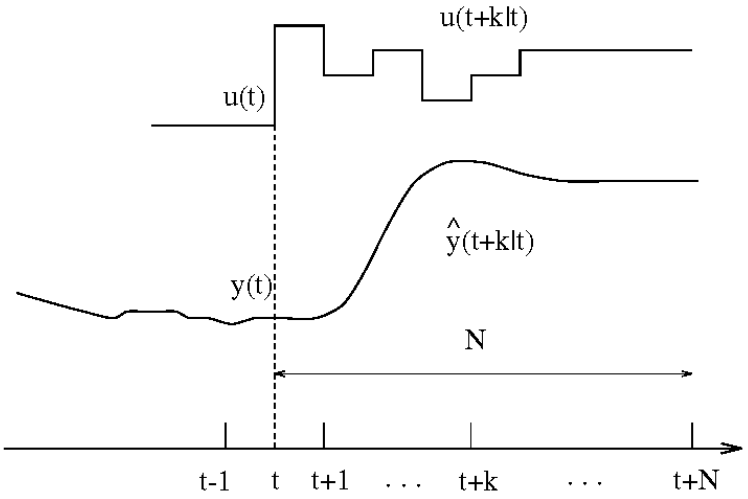


Fig. 1 MPC strategy.

In particular:

1. the future outputs in a determined prediction horizon N_{RH} are predicted at each discrete event t using the process model. These predicted outputs $y(t+k|t)$ [1] for $k = 1 \dots N$ depend on the known values up to t, past inputs and outputs, and on the future control signals $u(t+k|t)$, $k = 0 \dots N-1$, which are those to be computed and sent to the system;

[1] The notation indicates the value of the variable at event $t+k$ calculated at t.

2. the set of future control signals is calculated by optimizing a given criterion to keep the process as close as possible to the reference trajectory $w(t+k)$, which can be the setpoint itself or a close approximation of it. This criterion usually takes the form of a quadratic function of the errors between the predicted output signal and the predicted reference trajectory. The control effort is included in the objective function in most cases. An explicit solution can be obtained if the criterion is quadratic, the model is linear and there are no constraints; otherwise an iterative optimization method has to be used. Some assumptions about the structure of the future control law are also made in some cases to simplify the optimization problem, such as that it will be constant from a given instant onward;
3. the control signal $u(t|t)$ is sent to the process whilst the calculated control signals $u(t+i|t)$, $i > 0$ are rejected because at the next event $y(t+1)$ is already known and step 1 is repeated with this new value and all the sequences are brought up to date. Thus the term $u(t+1|t+1)$ is calculated, which in principle will be different from the $u(t+1|t)$ because of the new information available, using the receding horizon concept.

In order to implement this strategy, a basic structure shown in figure 2 can be used. A model is used to predict the future plant outputs, based on past and current values

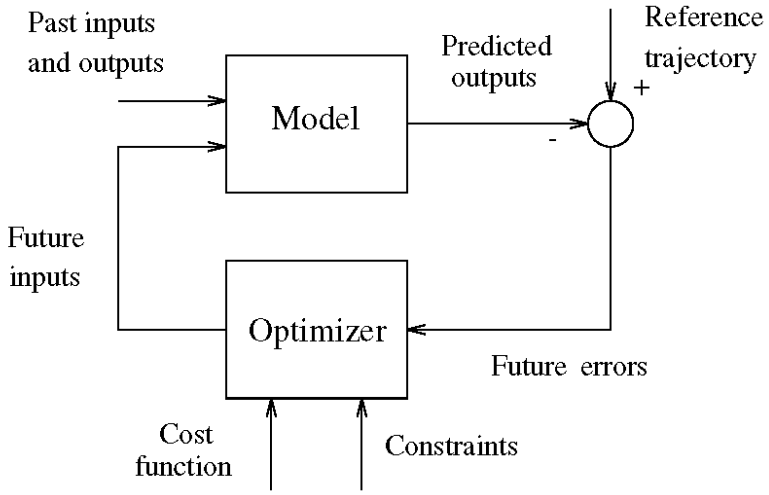


Fig. 2 MPC schema.

and on the proposed optimal future control actions. These actions are calculated by the optimizer taking into account the cost function, where the future tracking error is considered, as well as the constraints. The process model plays, in consequence, a decisive role in the controller. The chosen model must be able to capture the process dynamics to precisely predict the future outputs and must be simple to implement and understand.

The optimizer is another fundamental part of the strategy as it provides the control actions. If there are no constraints and the cost function is quadratic, its minimum can be obtained as an explicit linear function of past inputs and outputs and the future reference trajectory. In the presence of constraints, the solution has to be obtained by more computationally demanding numerical algorithms. The size of the optimization problems depends on the number of variables and the prediction horizons used. However, the amount of time needed for constrained problems can be of various orders of magnitude higher than the one needed for the unconstrained case and the bandwidth of the process to which constrained MPC can be applied is considerably reduced.

The implementation of MPC requires to specify the following elements:

- *Mathematical representation of the production plant*, which, at a given discrete event, for a given future control sequence, and for the current state of the plant, allows to compute the future evolution of the system along the prediction horizon. In the developed framework, the Plant Topology Modeling Language (PTML) has been defined. This language contains two different plant representations: the first one is topological and based on the interconnection of pre-specified blocks by means of arcs. These blocks can be machines, buffer nodes, item containers, ports etc., see 3.1. The second representation consists of the textual characterization of each block in terms of static or dynamic equations, or logic propositions, and allows to automatically represent blocks in terms of a MLD form which, properly connected, form the required overall MLD model of the plant.
- *Control dynamic constraints and cost function*. In order to complete the MLD model of the plant, and given its topology i.e. the interconnections among blocks, it is required to specify the control and state constraints, the cost function to be minimized, and the optimization parameters.
- *Synthesis of the MPC controller*. Given the specified optimization problem, the value of the optimal control solution is computed with the CP-SAT optimizer and the actual control value is selected according to the RH strategy;
- *Plant control*. The computed optimal control solution is actuated by means of the adopted hardware/software framework architecture, described in the following paragraph 3.4.

At the next event, the overall procedure is repeated. A more detailed description of the implementation of all these steps is reported in the following Section 3. However, at this stage a remark is due. The availability of a reliable mathematical model can be used in two ways. First, the model is required in the MPC algorithm to compute the future evolution of the system state when the MPC regulator is used on-line to control the system. Second, the model can be used in an off-line implementation to mimic the behavior of the system. In this way, the overall control structure can be thoroughly tested before implementing it in a real environment.

3 Open-source framework implementation

In this Section, the main characteristics of the approach are described with more detail.

3.1 Plant Topology Modeling Language

PTML allows to define the manufacturing components, e.g., items, item containers, control units, control actions. The blocks of PTML are now described. In order to detail the Plant Topology Modeling Language (PTML), a basic plant composed of 2 processing machines (NAME: MachineA, MachineB), feeded by 2 tending robots (NAME: ManiA, ManiB) , an input buffer node (NAME: InputBuffer) and an output buffer node (NAME: OutputBuffer) is modeled and represented in figure 3.

- **Items**: entities associated with a physical object movable in the production plant.

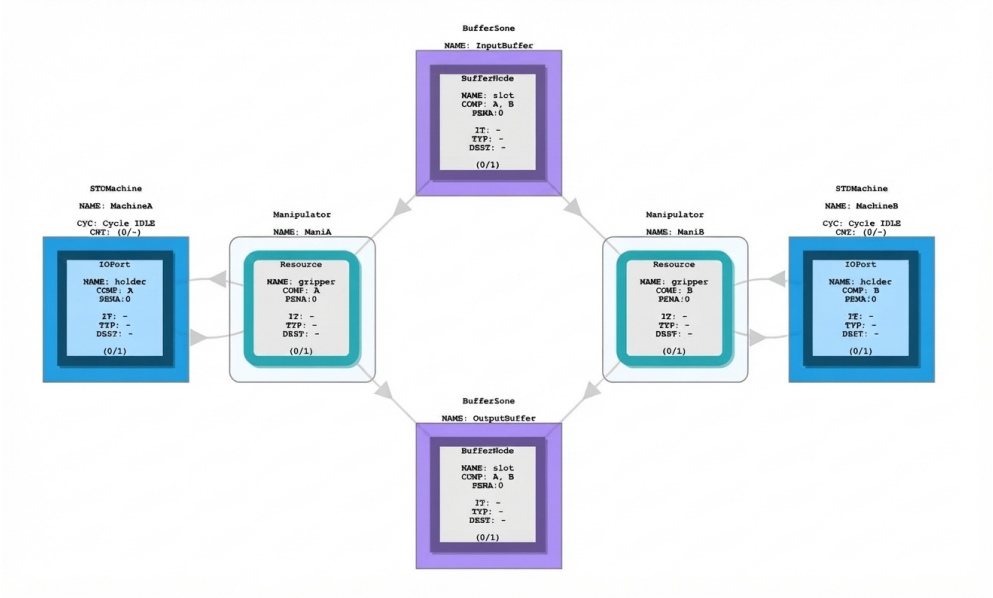


Fig. 3 Example of a simple plant modeled using the PTML.

Each item is distinct, i.e. it is associated with one and only one object and has a unique identification number, to be trackable along its production cycle.

- **Item Containers**:, virtual representation of devices that can hold an item. In the adopted graphical representation, item Containers are described by the internal rectangles shown in figure 3.

- **Nodes**: specialization of Item Containers, parts of Buffer zones. The purpose of a Nodes is just to hold an Item for buffering purposes or short and long term storage.

- **Buffer Nodes**: specialization of Nodes, used for short-term storage / buffering. A buffer node can contain at most one Item.

-**Source Nodes**: specialization of Nodes acting as "spawners" of Items. A source node is virtually identical to a Buffer Node, but it is only used to bring back to the production graph an Item that was parked in a Buffer Zone. So, Items can only be picked from a Source Node, to park them back one should use a Sink Node.

-**Sink Nodes**: specialization of Nodes acting as "holes". When an Item is moved to a

Sink Node it gets destroyed (figuratively, it only gets untracked by the plant model). Each time an Item is moved into a Sink Node, a counter is increased by one, so that the plant supervisor can control how many Items can be cumulatively placed into a Sink Node.
- **Resources**: specialization of Item Containers, used to model any physical holder part of a transportation system (conveyors, robots, etc.). They are used to move Items from/to a Node or a Machine port, or to another Resource.
- **Machine ports**: specialization of Item Containers used to model the Machine physical interfaces able to exchange Items.
- **IO ports**: specialization of Machine ports. They can be used to model machine interfaces where the loading/unloading point coincide. During processing, the Item inside a IO port is locked and then released once the cycle has finished.
- **Input ports**: specialization of Machine port. They can be used to model unidirectional machine loading interfaces. After a machine finishes processing, the Item inside an Input port disappears.
- **Output ports**: specialization of Machine ports. After a machine finishes processing, an Item appears in an Output port. Output ports are used in machines that assemble or disassemble objects. When an object is disassembled or assembled, its final shape (and thus Item type) can change, so the software framework should know what kind of item it gets transformed into. If the number of Item types allowed by an Output port is more than one, the choice is not unique, so it is required to specify one and only one Item type for each Output port.
- **Virtual ports**. specialization of Machine ports. Their behavior is similar to the one of IO ports one, with a major difference: the Item is not moved to the port but only placed on it. It means that, the Resource that is placing the Item in a Virtual port does not get freed afterwards: both the Resource and the Virtual port share the Item for the entire machine's processing duration. It can be used to model things like a microscope or a laser marker, in which a workpiece can be directly placed underneath it without being released from the manipulator. The Virtual port only acts as an image of the Resource that is loading it. All the item's information is temporarily mirrored to the Virtual port, and it is then cleared when the machine finishes processing.

Example 1 - PTML modeling
In order to describe with a simple example how the PTML works, consider a plant composed by two processing machines (NAME: MachineA, MachineB), feeded by two tending robots (NAME: ManiA, ManiB), an input buffer node (NAME: InputBuffer), and an output buffer node (NAME: OutputBuffer), and with PTML representation reported in figure 3. Specifically, the two machines are modeled by two Item Containers. In addition, there are two IO ports (because the Item is loaded and unloaded from just only one picking point), two Item Containers, two Resources (the two robots), one Item Container of type Buffer Node (InputBuffer), in which the Item to be machined is placed, and finally one Item Container of type Buffer Node (OutputBuffer), in which the machined Item can be picked up.

- **Control Units**: abstract entities used to model low level control systems. For

example, in a manufacturing environment, a PLC can be programmed to handle a robot that has many grippers, sensors and actuators. From an MPC control point of view, the coordination of each sensor/actuator is not part of the optimization challenge, so all the low level logic is masked by the low level controller, and only a small set of interactions can be executed, called **Control actions**. In the previous worked example, Control Units are represented by the external rectangles shown in figure 3.
- **Machines** types of Control Unit that can require one or more discrete events to process Items. Each Machine is modeled by a FSM (Finite State Machine) that iterates through a 3-states cycle:

$$\text{Idle} \Rightarrow \text{Processing} \Rightarrow \text{Finished} \Rightarrow \text{Idle}$$

Machines are normally in the *Idle* state, loading them triggers a transition to the *processing* state and a counter is started. After a certain number of events have elapsed (dictated by the machine processing time), the Machine FSM transitions to the *Finished* state, and, unloading the Item, the FSM triggers a transition back to the *Idle* state. A Machine is composed of one or more Machine ports (IO port, Input port, Output port or Virtual port).

- **Manipulators**: type of Control Unit whose purpose is to move items. Its Item Containers are the Resources and they move in group. For example, a robot with two independent grippers can be modeled as a Manipulator with two Resources. Moving in group means that the robot cannot load or unload two distinct machines at the same time, because both the Resources are built on the same end effector.

- **Buffer zones** type of Control Units that only act as buffers or storages for Items. Potentially, they can be considered a grid of slots in which placing Items in queue, without using any sensor or actuator. Each Buffer zone is composed of one or more Nodes, that are just virtual images of physical object containers. For example, a plant with a deposit station could be modeled by a Buffer zone, where objects (Items) are placed after the machining process, so as a temporary storage for the next processing steps.
- **Control Actions**: abstraction for commands to a plant to move or place an Item from an Item Container to another one. At a given discrete event k, each Control Action can either be active or inactive, represented by the state of the associated boolean variable. If it is active, then it will be executed during the evolution to the discrete event $k + 1$. Control actions are represented by a directed edge (arrow) that connects two different Item Containers.Going back to the example of figure 3, Control Units are represented by the larger (external), colored rectangles, Item Containers are represented by the smaller (internal) rectangles, while Control Actions are represented by direct edges connecting two Item Containers.
Once all the PTML elements have been presented, it can be useful to show examples on how topological rules can be applied to model a plant in the proper way.

Example 2 - Topological rules

Topological Rule 1.14 (Input ports):

It is only possible to move Items to an Input port and not from it. A Control Action cannot have an Input port as origin.
In fact, a Control Action is used to move an Item from two different Control Units, getting the Item from the Output port of the first Control Unit and releasing it to the Input port of the second ones.

Topological Rule 1.26 (Manipulators): *A Manipulator must contain at least one Resource.*
This means, for example, that a robot can have one, two or whatever end effectors, but not no end effector, otherwise it cannot operate any item movement.

3.2 Dynamic constraints, state evolution and cost function minimization

The definition of a set of control constraints, evolution rules and cost function to be optimized is necessary for the formulation of a suitable MLD model able to describe the pallet movement in the considered plant. To this end, and starting from the PTML model, it is necessary to define a set of control constraints and evolution rules for each Item Container and Control Unit of the PTML plant model. Each constraint is formally described as an equation or a logic proposition, so that it can later be translated into a form suitable for the numerical solver. This control constraints and evolution rules definition represent the fundamental core of the whole software framework, since they are used to convert the topological model into a problem suitable for optimization.

Dynamic Constraints: Several conditions must be defined to represent the evolution of the plant. These conditions can be formally defined by means of propositional calculus, see [11–13], and linear programming rules, see [14], which allow to provide a rigorous mathematical MLD formulation of the optimization problem.

Example 3 - Dynamic Constraints
As an example, a dynamic constraint is reported for clarity of the presentation.

Dynamic Constraint 2.1: *If a container is empty, no Control Action to pick an Item from it can be activated.*

$$\left[\kappa_i^N(k) = 0\right] \implies \left[\sum_{j \in \Lambda_i^{OUT}} \sigma_{ij}(k) = 0\right], \forall i \in C$$

where:

- $\mathbf{N}$ *is the prediction horizon length used in the MPC formulation;*
- $\mathbf{C}$ *is the set of all Item Containers;*
- $\kappa_{\mathbf{i}}^{\mathbf{N}}(\mathbf{k})$ *is the number of Items in Item Container* i *at* k*;*

- $\sigma_{\mathbf{ij}}(\mathbf{k})$ *is the boolean activation value of the Control Action moving the Item from Item Container* i *to Item Container* j *at* k*;*
- $\mathbf{\Lambda_i^{IN}}$ *is the set of Item Containers for which there exists a Control Action going into the Item Container* i*;*
- $\mathbf{\Lambda_i^{OUT}}$ *is the set of Item Containers for which there exists a Control Action coming from the Item Container* i*.*

According to the Dynamic Constraint 2.1, if an Item Container is empty, no Control Action can be issued to move an Item from that Item Container to another one. This is implemented by means of logical-mathematical proposition "if Item Container i *contains 0 Items, this implies that the sum of the Control Action activation coming from that Item Container must be equal to zero". Since the Control Action values can only have value* 0 *or* 1*, all the Control Action values must be equal to zero. By using the propositional calculus, see [11–13], and the linear programming rules, see [14], such logical-mathematical proposition is implemented as a linear equation and inequalities, suitable to compose the MLD formulation of the optimization problem.*

State evolution Once the plant state is known at a given k and the constraints are specified, then the next state is computed based on the Control Actions value.

Example 4 - State evolution
Consider two Buffer Nodes x *and* y*, the first contains an Item while the other is empty. It is chosen to move the Item from* x *to* y*, so that the Control Action* σ_{xy} *is set to* 1*. In order to model this desired plant behavior, the following linear equation can be formulated:*
$\forall i \in C^{N-BN}$

$$\begin{aligned}\kappa_i^{ID}(k+1) = \kappa_i^{ID}(k) \\ + \sum_{j \in \Lambda_i^{IN}} \kappa_j^{ID}(k)\, \sigma_{ji}(k) \\ - \sum_{j \in \Lambda_i^{OUT}} \kappa_i^{ID}(k)\, \sigma_{ij}(k)\end{aligned}$$

where:

- $\mathbf{C^{N-BN}}$ *is the set of Buffer Nodes;*
- $\kappa_{\mathbf{i}}^{\mathbf{ID}}(\mathbf{k+1})$ *is the integer ID of the Item present in the Item Container* i *at* $k+1$*;*
- $\kappa_{\mathbf{i}}^{\mathbf{ID}}(\mathbf{k})$ *is the integer ID of the Item present in the Item Container* i *at* k*;*
- $\mathbf{\Lambda_i^{IN}}$ *is the set of Item Containers for which there exists a Control Action going to the Item Container* i*;*
- $\mathbf{\Lambda_i^{OUT}}$ *is the set of Item Containers for which there exists a Control Action coming from the Item Container* i*;*

- $\sigma_{\mathbf{ij}}(\mathbf{k})$ *is the boolean activation value of the Control Action going from the Item Container i to the Item Container j at k;*

If the suggested equation is applied to both the Buffer Nodes, then the resulting set of two state evolution equations will be able to describe the movement of the Item from Buffer Node x to Buffer Node y or vice versa.

Cost function minimization
In order to compute the optimal Control Actions, a suitable cost function must be selected. In this work, it has been formulated as a linear combination of weighted variables where the weights can be tuned to modulate the relevance of the different summation terms. Then the problem optimization can be completed by minimizing the cost function

$$\min_{x \in X} F = \min_{x \in X} \sum_{i=1}^{h} w_i f_i(x)$$

where x represents the state variable, X is the state vector, F is the cost function, h is the number of terms of the summation, f_i the different terms, multiplied by their own weight.

Example 5 - Cost function minimization

Assume for example that the objective of the control design consists of minimizing the number of machines processing Items at k, then a suitable cost function is:

$$F = \sum_{k=0}^{N} \sum_{m \in U^{MH}} \left(1 - \phi_m^{S1}(k)\right)$$

where:

- $\mathbf{U^{MH}}$ *is the set of Machines;*
- $\phi_{\mathbf{m}}^{\mathbf{S1}}(\mathbf{k})$ *is a boolean variable that is 1 if the Machine m is in the* processing *state at k.*

3.3 Optimization problem formulation with CP-SAT

Given the MLD model of the plant under control, the selected cost function, and the current state $x(k)$, the CP-SAT optimizer is used to compute the optimal solution to the optimization problem. This implies to enumerate with a unique integer number the state variables in $x(k)$, i.e Items, Item Containers, Control Units, and Control Actions Then, For each of these components, a set of associated variables is generated and duplicated at any k in which they appear.
The overall set of dynamic constraints, state evolution equations and the cost function,

can be rearranged in this form:

$$\begin{aligned} x_{k+1} =& f(x_k, \delta_k, \alpha_k), \\ y_k =& h(x_k, \delta_k, \alpha_k), \\ \\ \mathcal{P} =& \{P_1(x_k, \delta_k, \alpha_k),\ P_2(x_k, \delta_k, \alpha_k), \\ & \dots, \\ & P_j(x_k, \delta_k, \alpha_k)\} \end{aligned}$$

where:

- $\mathbf{x_k}$ is the vector of state variables at k;
- $\delta_\mathbf{k}$ is the vector of Control Actions at k;
- $\alpha_\mathbf{k}$ is the vector of auxiliary variables at k;
- $\mathbf{y_k}$ is the vector of output variables at k, used to define objectives;
- $\mathbf{f}$ is a non-linear function computing the state evolution;
- $\mathbf{h}$ is a non-linear function computing the output variables;
- $\mathbf{P_j}$ is the j-th constraint;
- $\mathcal{P}$ is the set of constraints to be satisfied.

This form allows for a direct translation into CP-SAT variables and constraints generation, leading to a numerical model that can be solved to obtain an optimal control law.

3.4 Automation system implementation

Scope of the hardware/software architecture is to allow the MPC control algorithm running. In order to implement the MPC controller in a manufacturing scenario, the framework is integrated in a layered automation system with MES, SCADA, and PLC levels, see figure 4.

3.4.1 MES layer

The Manufacturing Execution System (MES) is the top layer of the control system. The planner program has a broad overview on the list of pieces to be manufactured, and the list of processes that each piece must go through. This layer is the one keeping the workload under control and deciding the production schedule. If, lets say, 1000 pieces are expected to be manufactured through the day, it makes no sense to ask the plant controller to optimize them all at once. Keeping the number of items in the plant bounded at any discrete event is crucial to control the plant's load and the MPC computational effort, so a priority order can be defined beforehand and progress at most N items at once. As each item completes a step, the MES checks its status and updates the item's next destination, sending this information directly to the SCADA layer.

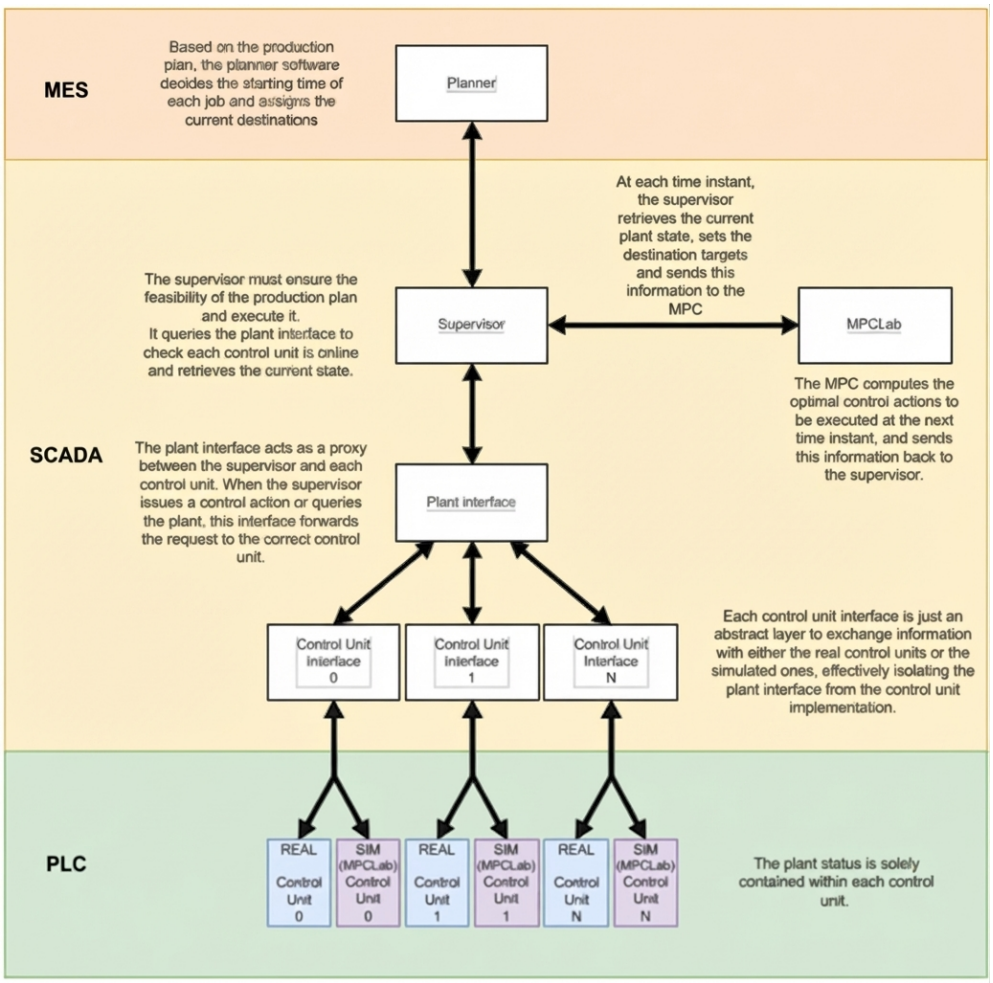


Fig. 4 Global automation system architecture.

3.4.2 SCADA layer

The SCADA layer then handles the items movement, driving them to the appropriate machine or buffer based on the MES' instruction. At the core of the SCADA layer there is the so called Supervisor, the software responsible for actuating the production plan provided by the top layer planner. Once it receives production instructions, it must translate them into commands that are sent to the plant in order to manage its working function. Before sending such commands, the Supervisor checks that the current plan is actually feasible, that means the current Items exist and their destinations are reachable. In fact, the plant layout can be variable. For example, a Control

Unit might go offline together with the Items contained into it, or an actuator could be on failure, situation which disable the related Control Action. To retrieve the current plant layout and status, the Supervisor exchange information with the so called Plant Interface, that is responsible both for providing functions and data streams to gather information from the plant, and to forward commands, like Control Actions, computed by the Supervisor. The plant interface connects each Control Unit through a Control Unit Interface that is a software module instantiated for each Control Unit, which acts as an abstract interface between the generic Control Unit implementation and the Plant Interface. The list of Item Containers, the list of Items, the list of available Control Actions are solely contained within each Control Unit. The Plant Interface merges together all information from each Control Unit in order to assemble the production plant, so to send the global plant situation to the Supervisor. This kind of configuration allows the complete separation between the PLC layer (often called also field layer) and the Supervisor layer; in this way, the Control Unit implemented into the PLC layer can either be real or simulated, so from the SCADA layer perspective, it doesn't make a difference. This means that the software functions used to gather data and to send commands to the Control Units are relayed, by the Control Unit interface, to real plant devices or simulated ones, so this approach allows for a comprehensive testing of the MPC framework with a simulated plant, making the commissioning plant phase extremely easy. Once the Supervisor retrieves all the information from the Plant Interface, then the MPC algorithm can be run so to get the next optimal control actions to be sent to the plant. The Supervisor has the authority to send any command to the Control Units, regardless of the commands were proposed by the MPC controller or not. This decoupling mechanism consists of an important automation system requirement, allowing manual intervention in case of need. In fact, nevertheless the MPC controller is a smart and adaptive algorithm able to adapts itself in case of plant faults, it should always be possible to bypass it in case a manual sequence of commands needs to be issued, like for example during maintenance or during testing of new Control Actions that are not yet part of the production graph.

3.4.3 PLC layer

Finally, PLC layer consists of the framework part in which the low level logic is coded. Each Control Unit is responsible for reading its own sensors and controlling its own actuators, completely masking its internal dynamics to the high level control system. As mentioned before, all information regarding the plant layout (list of Item Containers and available Control Actions) and plant status (list of Items and where they are laying in the plant) is just only contained, distributively, in each Control Unit. In case a Control Unit fails while it is handling one or more Items, then the Control Unit and the Items managed by it become unavailable. At each polling cycle executed by the Supervisor, each Control Unit sends updated information about its status, effectively achieving a dynamic layout that can adapt in real-time with changes. Most importantly, each Control Unit is also responsible for the integrity

check of the Control Actions received from the Supervisor. In fact, in case of a bad manual commands sequence or a bug in the MPC controller, unfeasible Control Actions could be sent to a Control Unit. It is even possible, and a very likely eventuality, that during the interval time occurring between the instant time in which the Supervisor has polled the Control Unit state and the instant time in which the Control Actions are sent, the plant state has changed. So, it is crucial that, when a Control Unit receives a command, it checks whether it is feasible or not; if not, the Control Unit can reject the command marking it as unfeasible and triggering a plant pause, or asking the Supervisor to poll again, skipping one iteration in the control loop. On this layer, another problem is addressed: it is not uncommon that a Control Unit can contain many Item Containers and many Items, but only a set of them need immediate optimization. Many Items can be tracked but "parked" for a long time before the MES layer decides that they need progressing. In this scenario, it doesn't make sense to inform the Supervisor about every Item and Item Container, as they would greatly increase the computational complexity of the optimization problem. Furthermore, control units can contain manufacturing material that is stack-able (more than 1 item in the same location) or not tracked yet.

3.4.4 Control Unit layering

To tackle this, a layered structure has been introduced for each control unit as shown in figure 5.

Slots VS Item Containers. The physical Item Containers are modeled separately, with the name of Slots. If an Item physically exists in a Control Unit, it must be contained in one and only one Slot. Then, each Item can be mapped to an Item Container to expose it to the Supervisor. This effectively creates a mask, in which Item Containers are just images of the physical Slots: each Item in the Control Unit is uniquely tracked and associated with a Slot, but one can choose whether to also map it to an Item Container or not. With this structure, the MES layer can decide when it is the right time to expose a new Item in the plant (by mapping the hidden Slot with an Item Container) and progress its production cycle, while maintaining the size of the graph bounded and the optimization problem within computational feasibility.
Items VS Stock. Another scenario that is common in manufacturing plants, consists of having multiple identical workpieces within a storage unit that, at a certain event, need to be picked one by one to be assembled in more complex parts. For example, a Control Unit might have a storage of screws contained in a bin that does not need to be tracked one by one until they are picked by a robot. In this scenario, it can be easier to just track the whole batch of parts with an identifier and the number of pieces, and to convert them to tracked items just as they need to be used. To embed such structure in the software framework, the concept of Stock has been introduced. A Stock can be assigned to one or more Slots in different quantities, as shown in figure 5. Stock components cannot be directly mapped to Item Containers, as they are not uniquely tracked. To expose a Stock component to the plant, it must firstly be

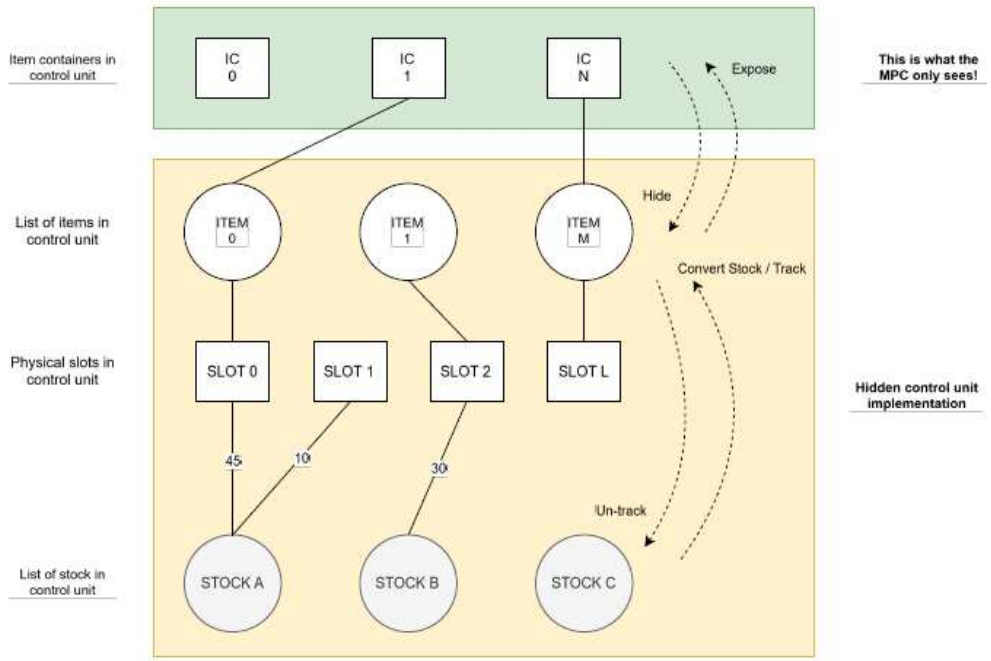


Fig. 5 Control Unit layering.

converted into an Item, each of them characterized by a unique identification number to be used from the Supervisor, the MPC and the MES layer. After the Item has been generated, it can be exposed by mapping it to an Item Container, so making it visible and movable on the plant. In this way, the number of Items can be kept as low as the number of pieces that actually need to be tracked in the plant.
Based on the described Control Unit layer, the upper control layers have access to three main interactions:

- **Tracking / Un-tracking**: operations that convert Stock to Items and vice versa, to control the active pieces in the plant;
- **Exposing / Hiding**: operations that map or un-map an Item to an Item Container, in order to make them movable and controllable by the Supervisor, or to park them indefinitely;
- **Control Actions**: the actual commands, sent by the Supervisor, to move an Item from an Item Container to another one.

According to the PTML and Control Unit layering specification, just only Buffer Zones are intended to contain Stock or have different mappings between Item Containers and Slots. On the other hand, Manipulators and Machines have unique, and well defined, Resources and Ports, and they are not meant to contain stock material. So, for

these two classes of Control Units, just only Control Actions are implemented, without the possibility of controlling the tracking and exposing of Items. Instead for Buffer Zones, interactions can be combined to tackle real-world scenarios, very common in a manufacturing plant.

Scenario I: Picking one unit of a Stock from a Buffer Zone. One machine requires a piece to be processed, that is currently inside a Buffer Zone Control Unit, stocked in a storage Slot.

1. Spawn the Item: convert one unit of the wanted Stock from one of the Slots that contain it, assigning an unique tracking ID;
2. Expose the Item: map the Item to a Source Node in the Control Unit. In this way, the Item is now available for the Supervisor;
3. Pick the Item: call a proper sequence of Control Actions. It can be programmed manually or automatically computed by the MPC by setting the correct destination of the item. Actions 1 and 2 are directly issued by the MES layer, based on the production plan, and relayed through the Supervisor, while action 3 is issued by the Supervisor only.

Scenario II: Park an Item into a Buffer Zone for long term buffering. An Item needs to be placed into a Buffer Zone for a long period (much longer than the prediction horizon of the MPC) without losing track of it.

1. Move the Item: call a proper sequence of Control Actions that take the Item to a Sink Node in the destination Control Unit. A Sink Node has an in-built mechanism such that, when an Item is moved into it, the Item is automatically hidden. By doing this, the Item will be still tracked and placed in a Slot, but hidden from the Supervisor and not movable any more.

Scenario III: Pick a parked Item. This scenario is similar to scenario I, but the item already exists.

1. Expose the Item: map the Item to a Source Node in the Control Unit. In this way, the Item is now available to the Supervisor;
2. Pick the Item: call a proper sequence of Control Actions to move the Item to the desired destination.

Scenario IV: Refill a Stock. An Item needs to be moved to a Buffer Zone to refill a Slot with Stock. This scenario is similar to scenario II, but the tracking of the Item must be lost.

1. Move the Item: call a proper sequence of Control Actions that take the Item to a Sink Node.
2. Un-track the Item: convert the Item to a unit of the wanted Stock, effectively making the Item disappear.

With this structure, at each cycle, the Supervisor software in the SCADA layer queries all Control Units about their state, builds an updated plant graph and merges information on scheduling coming from the MES layer, sends it to the MPC framework and waits for an optimal solution to be computed. After that, it forwards the

computed solution to the interested Control Units, waiting for their acknowledgment of a successful execution, to repeat the cycle right afterwards.

4 Simulation runs for framework validation

To test the whole framework, simulations on a dummy plant have been run. The test plant is composed of 27 Control Units, 44 Item Containers and 96 possible Control Actions.

Problem size - prediction horizon

Unfortunately, mixed integer problems are proven to be NP-Hard, that loosely means that the computational complexity grows exponentially (or at least faster than any high order polynomial) with the size of the optimization problem. Figure 6 shows the trend of the number of Variables and Constraints related to the prediction horizon values.

Size of the model vs. prediction horizon length

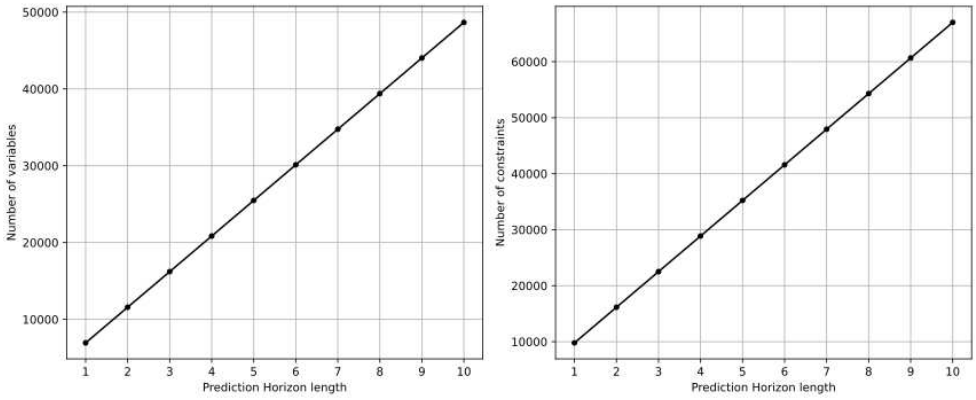


Fig. 6 CP-SAT model size as a function of the prediction horizon length.

In figure 7 the relation between the computation time needed to solve the optimization problem and the prediction horizon, value is detailed. The linear y-axis plots highlights a rapid exponential growth, confirming the NP-Hardness of the problem. In order to more clearly highlight the growing effect, different number of Items have been considered in the test run. Since the time to compute a solution must be bounded in order to be applied to a real industrial application, and the computational power is limited, keeping the problem size bounded is crucial. This leads to an appropriate choice of the prediction horizon length.

Computation time vs. prediction horizon length

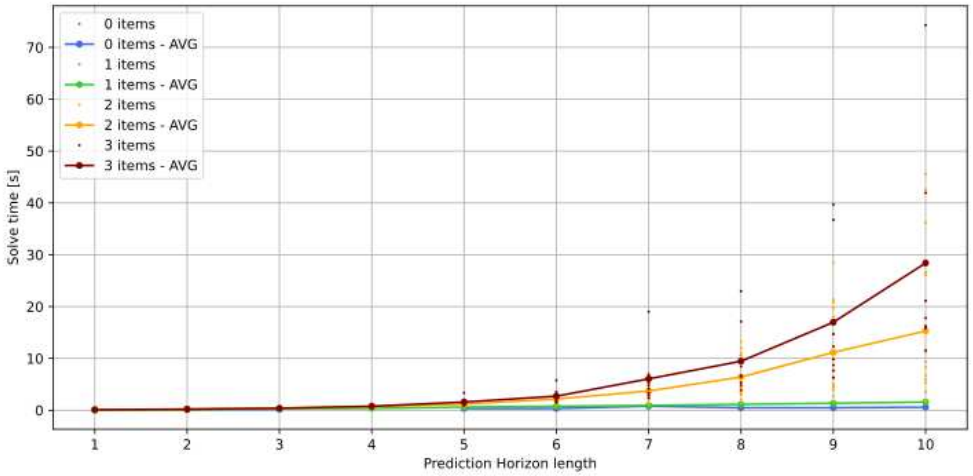


Fig. 7 CP-SAT model computation time as a function of the prediction horizon length.

Problem size - number of Items

In addition, the number of Items present in the plant influences the number of possible solutions, since the problem is of a combinatorial nature. Several experiments were run using a sample plant to gather trends in performance. In figure 8 the trend of the computation time related to the number of Items involved in the test run, according to different values of the prediction horizon, is showed by using linear y-axis plots.

Computation time vs. active number of items

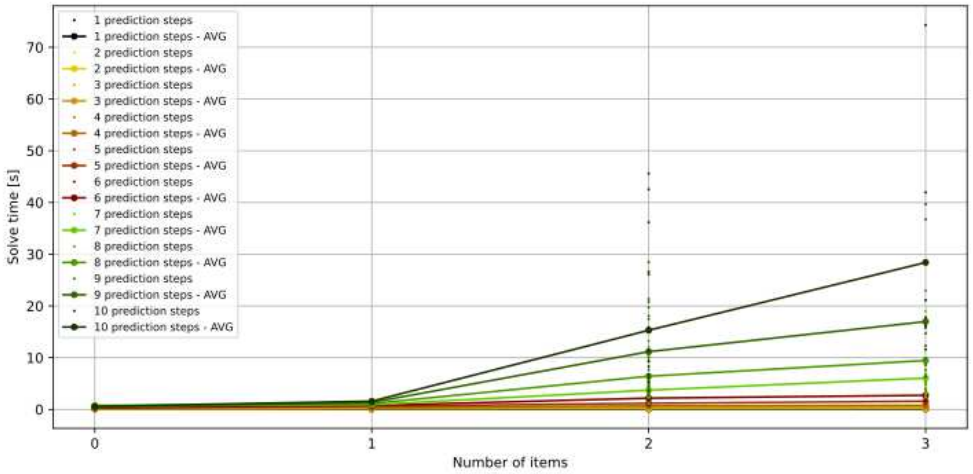


Fig. 8 CP-SAT model computation time as a function of the active number of items.

Deadlock

In order to complete the consideration about the test, it is important to spend few words on possible deadlock that can occur during a plant control. A deadlock in a manufacturing plant occurs when two or more processes, machines, or parts wait on each other in a circular dependency, resulting in a complete halt in production. Each element in the cycle holds a resource that another one needs, but none can proceed until the others release their resources, creating a "traffic jam" in the production line. For example, Machine A has a part that Machine B needs to work on, and Machine B has a part that Machine A requires. If neither machine can release the part it holds until it receives the part it needs, they'll remain stuck indefinitely, unable to proceed with production. In the context of the MPC developed in this work, a dead-lock can have a broader meaning: in certain scenarios, the controller could not predict long enough to find a solution that makes the plant evolve. It can exist an input sequence that unblocks the plant from a stuck state, but the prediction horizon might be shorter than the sequence, effectively leading to a halted state. A unique, deterministic approach to avoid any type of deadlock for any generic plant layout could not be developed yet, and it will be part of future developments. However, some measures proved effective in simulation runs:

- Proper scheduling: The MES layer has the power to instruct the Supervisor software which Items can progress and which not. At the MES level, the whole production plan of each part is known, so it makes sense to pre-schedule each part in a way that does not generate circular dependencies. The scheduling approach is not unique and is not part of this framework, as it is too much plant-specific;
- Limiting the number of exposed Items: Again, the MES layer has a broader overview on the production plan, so it is important that the feeding rate of Items is less or equal than the plant's throughput capabilities. Feeding more items than it can process leads to an overcrowding, increasing computational demand and accumulating items nearby their destination, progressively leading to a deadlock;
- Proper tuning of the cost function: Weights and penalties set in the cost function critically influence the controller's behavior. Paths that have higher risk of deadlocks can be penalized to prefer safer alternatives, and the permanence of Items on critical containers can be avoided as well. Choices are endless, and based on the specific plant's layout, one can tune the cost function to achieve the intended behavior.

5 Conclusions

In this paper the development of an open-source framework for Model Predictive Control (MPC) tailored for discrete-event manufacturing systems has been presented. The developed MPC framework, designed in Python and utilizing Google's CP-SAT solver, has proven capable of handling each step from the automatic generation of an optimization problem to the actual solution finding, making it a free and versatile choice in manufacturing plants that are structured compatibly with the developed modeling language. Simulation results confirmed the exponential dependence between the optimization problem computation complexity and the prediction horizon /

number of Items. This implies the optimization problem size bounding, that can be obtained by limiting the prediction horizon (deadlock could occur) ora by reducing the number of Items exposed to the Supervisor. In fact,it can occur that only a set of Items need immediate optimization, and many other can be hidden to the Supervisor, so to reduce the optimization problem size and then decrease the MPC computational complexity. At the time of writing of [15], the framework has been implemented in a real contact lenses manufacturing plant, to test it on the field. Further improvements are planned to improve the optimization problem formulation, changing the conceptual model conceived to describe the Item movement and machining, so to reduce the optimization problem size. Another research theme regards changing the optimization problem format so to focus on solver's performances and capabilities increasing.

References

[1] Alba, C.B.: Model Predictive Control. Springer, London (2012)

[2] Rawlings, J.B., Mayne, D.Q.: Model Predictive Control: Theory and Design, 1st edn. Nob Hill Publishing, Madison, Wisconsin, USA (2009)

[3] Katayama, S., Murooka, M., Tazaki, Y.: Model predictive control of legged and humanoid robots: models and algorithms. Advanced Robotics **37**(5), 298–315 (2023)

[4] Qu, S., Zhu, G., Su, W., Shan-Min Swei, S., Hashimoto, M., Zeng, T.: Adaptive model predictive control of a six-rotor electric vertical take-off and landing urban air mobility aircraft subject to motor failure during hovering. Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering **236**(7), 1396–1407 (2022)

[5] Cataldo, A., Perizzato, A., Scattolini, R.: Production scheduling of parallel machines with model predictive control. Control Engineering Practice **42**, 28–40 (2015)

[6] Rokhforoz, P., Fink, O.: Distributed joint dynamic maintenance and production scheduling in manufacturing systems: Framework based on model predictive control and benders decomposition. Journal of Manufacturing Systems **59**, 596–606 (2021)

[7] Wenzelburger, P., Allg"ower, F.: Model predictive control for flexible job shop scheduling in industry 4.0. Applied Sciences **11**(17), 8145 (2021)

[8] Alessandri, A., Cervellera, C., Gaggero, M.: Predictive control of container flows in maritime intermodal terminals. IEEE Transactions on Control Systems Technology **21**(4), 1423–1431 (2012)

[9] Cataldo, A., Scattolini, R.: Dynamic pallet routing in a manufacturing transport line with model predictive control. IEEE Transactions on Control Systems Technology **24**(5), 1812–1819 (2016)

[10] Cataldo, A., Morescalchi, M., Scattolini, R.: Fault-tolerant model predictive control of a de-manufacturing plant. The International Journal of Advanced Manufacturing Technology **104**, 4803–4812 (2019)

[11] Bemporad, A., Morari, M.: Control of systems integrating logic, dynamics, and constraints. Automatica **35**(3), 407–427 (1999)

[12] Cavalier, T., Pardalos, P., Soyster, A.: Modeling and integer programming techniques applied to propositional calculus. Computers & Operations Research **17**(6), 561–570 (1999)

[13] McKinnon, K.I.M., Williams, H.P.: Constructing integer programming models by the predicate calculus. Annals of Operations Research **21**(21), 227–246 (1989)

[14] Williams, H.P.: Model Building in Mathematical Programming. Fifth Edition. Wiley, Chichester (2013)

[15] Gottardini, D.: Development and implementation of a framework for Model Predictive Control of discrete-events manufacturing systems. Master's thesis, Politecnico di Milano (2024)

Statements & Declarations

The authors declare that no funds, grants, or other support were received during the preparation of this manuscript.

The authors have no relevant financial or non-financial interests to disclose.

All authors contributed to the study conception and design. Material preparation, data collection and analysis were performed by Andrea Cataldo, Riccardo Scattolini and Daniele Gottardini. The first draft of the manuscript was written by Andrea Cataldo and all authors commented on previous versions of the manuscript. All authors read and approved the final

manuscript.