Available online at www.sciencedirect.com

ScienceDirect

IFAC Papers Online CONFERENCE PAPER ARCHIVE

IFAC PapersOnLine 55-2 (2022) 396–401

Safeguarded optimal policy learning for a smart discrete manufacturing plant

Roberto Boffadossi *, **Fabio Bonassi** *, **Lorenzo Fagiano** *, **Riccardo Scattolini** *, **Andrea Cataldo** **

* *Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Italy (e-mail: name.surname@polimi.it).*
** *Institute of Intelligent Industrial Technologies and Systems for Advanced Manufacturing (STIIMA), National Research Council, Milano, Italy (e-mail: andrea.cataldo@stiima.cnr.it)*

Abstract: An approach to safely learn and deploy, at fast rate, a given optimization-based controller for the routing problem in smart manufacturing is presented. The considered application features a large number of integer decision variables, combined with nonlinear dynamics, temporal-logic constraints, and hard safety constraints. The approach employs a neural network as feedback controller, trained using a data-set of state-input pairs collected with the optimization-based controller. A safeguard mechanism checks whether the input computed by the neural network is feasible or not, in the latter case the optimization-based controller is called. Results on a high-fidelity simulation suite indicate a strong decrease of average computational time combined with a negligible loss of plant performance.

Keywords: Advanced Manufacturing, Nonlinear Model Predictive Control, Machine Learning for Control, Safe Learning, Neural Networks

1. INTRODUCTION

Reconfigurable manufacturing systems (RMS) are one of the main building blocks of the Industry 4.0 paradigm, see e.g. Kang et al. (2016); Zheng et al. (2018); Cataldo et al. (2019); Koren and Shpitalni (2010); Koren et al. (2018). They enable real-time adaptation and re-routing of parts in order to respond to market changes, to improve the resilience and the energy efficiency of the process. However, the increase of flexibility implies a more complex decision-making problem to achieve optimal operation, giving a new impulse to a research topic spanning the last few decades, see Gupta et al. (1989); Byrne and Chutima (1997); Das and Nagendra (1997); Kouiss et al. (1997); Peng and Chen (1998); Saygin et al. (2001); Vargas-Villamil and Rivera (2000, 2001); Moro et al. (2002); Souier et al. (2010); Bucki et al. (2015); Cataldo et al. (2015). In this work we consider a discrete manufacturing plant equipped with a flexible, modular transportation system. The integer commands issued to each transportation module must be chosen to optimize a long-term performance criterion, while guaranteeing satisfaction of operational safety constraints, both static and temporal-logic ones.

In recent research efforts, we developed different solutions for this class of problems, all based on a hierarchical decomposition and real-time optimization using Model Predictive Control (MPC), as presented in Cataldo and Scattolini (2016); Fagiano et al. (2020); Boffadossi et al. (2021). MPC makes it possible to operate the plant optimally according to the chosen performance criterion, which can be set by the user to achieve various trade-offs between throughput and energy consumption, while guaranteeing satisfaction of constraints. On the other hand, these on-line optimization-based approaches may require a significant time to reach a satisfactory solution of the receding horizon routing problem, whose complexity increases significantly with the size of the plant and the considered prediction horizon. Such a computational overhead, if prolonged over time, may be not acceptable in real-world operation where plant throughput is one of the main performance indexes. To cope with this drawback, the idea we propose in this paper is to learn and replicate the behavior of the MPC law with a universal approximator (a feedforward Neural Network in our specific case). Technically, this can be seen as a variant of approximate MPC approaches (Johansen (2004); Canale et al. (2010)) which however have never been used for problems entailing a nonlinear integer program, as the one considered here. Moreover, differently from the literature, which is mainly concerned with off-line training of the approximator and then on-line usage as feedback controller, we propose a safeguarded batch-training approach, presented in Figure 1. In this novel strategy, a Safeguard Mechanism (SM) oversees the behavior of the MPC approximator (i.e, the Feed-Forward Neural Network), by checking whether the input to the plant is compatible with the operational constraints and with optimization-based operation. If an incompatibility is found, the MPC routine is called to compute and apply a feasible input, and the state-input data is added to a training set. The latter is then used periodically to re-train the approximator, thus gradually approaching the behavior of the MPC law. In other words, the MPC law is

⋆ This research was funded by a grant from the Italian Ministry of Foreign Affairs and International Cooperation (MAECI), project "Real-time control and optimization for smart factories and advanced manufacturing".

10.1016/j.ifacol.2022.04.226

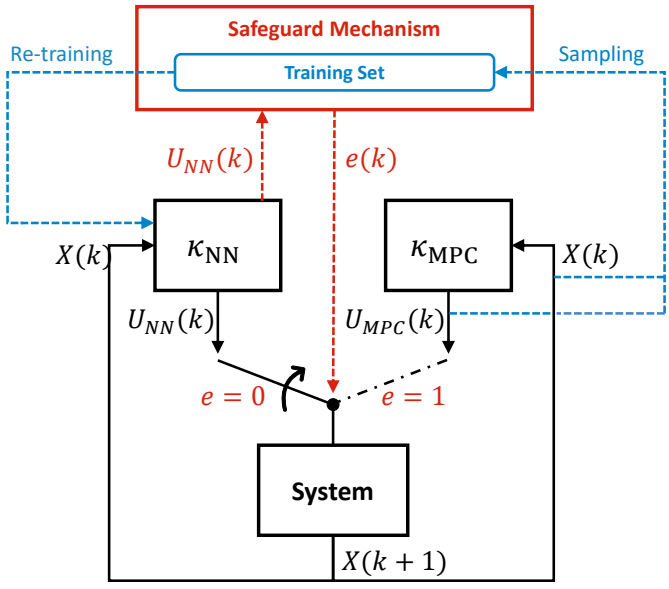

Fig. 1. Proposed control structure, where κ_{MPC} and κ_{NN} are the MPC and the FFNN control laws, respectively.

used to guarantee operational safety and as an "instructor" for the approximator, which is then able to replicate the decision-making algorithm at much faster rate.
As a result, we are able to dramatically reduce the computational effort over time, since after a few training batches the approximator can reproduce the MPC law with high accuracy, and the latter must intervene only sporadically. At the same time, we retain safety guarantees and performance optimality.

2. PROBLEM DESCRIPTION AND BACKGROUND

2.1 Problem description

We model the plant as a directed graph with N_n nodes, see Fig. 2 for a simplified example. Each node $h = 1, \dots, N_n$ can be a transportation one, a machine, or an input or output station. Furthermore we denote by L the number of nodes in the graph corresponding to a machine or a input/output station. A fixed number N_p of pallets travels on the plant. At each discrete time k, each pallet i is at a given node h_i and has a target g_i among N_t possible ones. The possible targets are either machines or input/output stations. Pallets are emptied at the output stations and must move to an input station to load a new part. When a part is first loaded on a pallet, a target machine is set, corresponding to the first job that must be executed on that part. Then, a new target is set after the execution of each job. The new target may be a-priori uncertain, as it generally depends on the outcome of the previous job(s). After all suitable jobs have been executed on a part, the target is the output node. Finally, each pallet $i = 1, \dots, N_p$ has also a total time counter t_i, equal to the number of time steps elapsed since the last loading operation (i.e., the time since when the part currently traveling on pallet i has entered the plant), and a machine indicator m_i. The latter is equal to 1 when the pallet is in a machine node and the current job has finished (i.e, the pallet can be unloaded from that machine), otherwise it is equal to 0. The state of each pallet is denoted with:

$$\boldsymbol{x}_i(k) = [h_i(k),\, g_i(k),\, t_i(k),\, m_i(k)]^T,$$

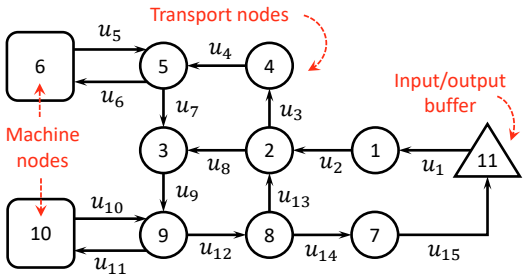

Fig. 2. Example of a graph representation of the plant

where $\cdot^T$ is the matrix transpose operation, and the state of the whole plant is:

$$\boldsymbol{X}(k) = [\boldsymbol{x}_1(k)^T, \dots, \boldsymbol{x}_{N_p}(k)^T]^T \in \mathbb{N}^{4N_p}. \quad (1)$$

The graph edges indicate the moves between two nodes that are physically achievable by the plant's transportation modules. A Boolean command $u_i, i = 1, \dots, N_u$, is associated to each edge and takes the value 1 if the corresponding move must be actuated at time k and 0 otherwise, see Fig. 2. Denoting with $\boldsymbol{U}(k) = [u_1(k), \dots, u_{N_u}(k)] \in \{0,1\}^{N_u}$ the vector of all commands, the plant's dynamics can be described as:

$$\boldsymbol{X}(k+1) = f(\boldsymbol{X}(k), \boldsymbol{U}(k)) \quad (2)$$

The control strategy shall compute $\boldsymbol{U}(k)$ at each k in order to move all pallets towards their targets while avoiding gridlocks, i.e., situations in which no pallet can move anymore due to the presence of the other ones, and satisfying operational constraints. The latter are plant-specific and include, for example, the inability to trigger two or more specific transitions at the same time (static constraints), or the need to wait for a job to finish before making a move out of the corresponding machine (temporal-logic constraints). Moreover, the control system shall optimize given economic criteria, for example a trade-off between throughput (number of parts that are unloaded per time unit) and power consumption (number of commands actuated). Even for relatively small plants, this problem features a large number of Boolean optimization variables together with both static and temporal-logic state and input constraints, making it difficult to compute the global optimum in reasonable time.

2.2 Background: Hiearchical Predictive Routing Control

In Fagiano et al. (2020) and Boffadossi et al. (2021), we presented a design method that involves the definition of a number of sensible sequences of nodes across the plant. Then, a hierarchical decomposition with two levels is adopted:

i) *Predictive path allocation.* At high level, a receding horizon controller (re-)assigns the sequences to each pallet in order to avoid gridlocks, satisfy operational constraints, and maximize the plant performance. This is done by solving a finite horizon optimal control problem (FHOCP) with initial state equal to $\boldsymbol{X}(k)$. We denote such an optimization problem as $\mathcal{P}(\boldsymbol{X}(k))$.

ii) *Path following strategy.* At low level, a set of general rules computes $\boldsymbol{U}(k)$ to satisfy operational and temporal-logical constraints and, whenever possible, advance each pallet along the sequence assigned by the high level.

In problem $\mathcal{P}(\boldsymbol{X}(k))$, the system is simulated for a chosen prediction horizon N starting from the current state $\boldsymbol{X}(k)$, under the action of the low-level path following strategy. The predicted behavior is optimized by exploring a decision-tree, which represents all the possible sequences that can be assigned to the pallets at each future time.

The described strategy, named Hierarchical Predictive Routing Control (HPRC), can be seen as a feedback policy of the form:

$$\boldsymbol{U}_{\text{MPC}}(k) = \kappa_{\text{MPC}}(\boldsymbol{X}(k)), \tag{3}$$

defined on the following set:

$$\mathcal{F} = \{\boldsymbol{X} : \mathcal{P}(\boldsymbol{X}) \text{ is feasible.}\}, \tag{4}$$

In nominal conditions, if N is chosen large-enough (namely larger than the time required for all pallets on the plant to conclude at least one full trip between input and output nodes), the HPRC achieves the following property (*recursive feasibility*):

$$\boldsymbol{X}(k) \in \mathcal{F} \Rightarrow f(\boldsymbol{X}(k), \boldsymbol{U}_{\text{MPC}}(k)) \in \mathcal{F}, \tag{5}$$

Moreover, the MPC law is such that:

$$\boldsymbol{U}_{\text{MPC}}(k) \in \mathcal{U}(\boldsymbol{X}(k)), \forall k, \tag{6}$$

where $\mathcal{U}(\boldsymbol{X}(k))$ is the set of all input vectors that satisfy the static and temporal logic constraints of the plant, which in part depend on its current state $\boldsymbol{X}(k)$. Properties (5)-(6) imply, in turn, that all safety constraints are always satisfied and that no gridlock ever occurs.

The HPRC law obtained extremely good results with a manageable computational time on a real plant of medium size, see Boffadossi et al. (2021) for the details. However, the involved optimization program is an integer nonlinear one, such that the global solution can be found only through complete exploration of all possible combinations of decision variables. In Boffadossi et al. (2021) we approached this problem by introducing ad-hoc strategies to quickly find a feasible candidate solution, tolerating possible sub-optimality. Yet, in some application cases the computational time may still be not negligible for the sake of economic performance. Moreover, such a hierarchical decomposition and tailored optimization routines do not avoid an exponential growth of complexity on larger plants, but simply alleviate it. Indeed, all optimization-based approaches, such as also the one of Cataldo and Scattolini (2016), share this same drawback, motivating the research efforts aimed to learn κ_{MPC} from data to avoid or reduce the need to solve $\mathcal{P}(\boldsymbol{X}(k))$ on-line. At the same time, the guaranteed safety properties of the predictive controller shall be retained. We present next our approach to achieve safe learning in this context.

3. SAFEGUARDED OPTIMAL POLICY LEARNING

To decrease the computational effort, we propose to learn κ_{MPC} with a Feed-Forward Neural Network (Bengio et al. (2017)), denoted with κ_{NN}:

$$\boldsymbol{U}_{\text{NN}}(k) = \kappa_{\text{NN}}(q(\boldsymbol{X}(k))), \;\; \kappa_{\text{NN}} \approx \kappa_{\text{MPC}}, \tag{7}$$

where function $q(\cdot)$ represents an encoding procedure that we describe later on (Section 3.1). The choice of a feed-forward network stems from the fact that κ_{MPC} is a static system, i.e. it has no internal state. A FFNN consists of an ordered sequence of layers with position $\ell = 1, \dots, N_l$, where $\ell = 1$ is the input layer and $\ell = N_l$ the output one. Each layer has a number d_ℓ of neurons which are chosen by the user, with the exception of d_1 (detailed in Section 3.1) and d_{N_l} which is equal to the size of the output, i.e. N_u in our case. Any intermediate layer, i.e. $\ell = 2, \dots, N_l - 1$, consists in an affine transformation of the previous layer's output, which is then passed through a nonlinear activation function, except for the last layer that features a linear activation function. For more details on FFNNs, the interested reader is addressed to Bengio et al. (2017).
Function κ_{NN} is trained using a data-set $\mathcal{D}(k)$ of state-input pairs:

$$\mathcal{D}(k) = \left\{ (\boldsymbol{X}^{(i)}, \boldsymbol{U}^{(i)}_{\text{MPC}}),\, i = 1, \dots, M(k) \right\}. \tag{8}$$

The number $M(k)$ of data pairs increases over time during system operation, as explained later on.

To effectively train and deploy κ_{NN} in the considered feedback control system, the following aspects must be solved:

1) How to efficiently build the training set, avoiding the presence of redundant states?
2) How to ensure that safety constraints are always satisfied notwithstanding the approximation errors?
3) How to smoothly insert κ_{NN} in the control loop, and gradually improve the quality of the data-set $\mathcal{D}(k)$ (hence of the training process) over time?

We describe next our solution to these points.

3.1 State encoding

The plant state $\boldsymbol{X}$ is not best suited to be used as input argument of a FFNN. The reason is that different $\boldsymbol{X}$ values can actually correspond to the same plant condition, least some aspects that are irrelevant from the standpoint of optimal routing. This is apparent for example by swapping the targets, nodes, and priorities of two pallets h and l: the corresponding two states are different (in particular, the entries pertaining to pallet h are swapped with those of pallet l, see (1)), but the value of U_{MPC} resulting from the solution of problem $\mathcal{P}(\boldsymbol{X}(k))$ is the same. Another example pertains to the total time values t_i, whose numerical values are used only to establish priorities among parts, so that infinitely many values can lead to the same ordering and hence the same control input. In such a situation, an encoding procedure shall be carried out to sort out the ambiguities, see Bengio et al. (2017). We decide to adopt a binary state encoding procedure in this work, see Franco (2006) and Potdar et al. (2017):

$$\tilde{\boldsymbol{X}}(k) = q(\boldsymbol{X}(k)), \tag{9}$$

where $\tilde{\boldsymbol{X}}(k)$ is the encoded version of state $\boldsymbol{X}(k)$. The encoding procedure $q(\cdot)$ is defined by Algorithm 3.1.

The obtained vector $\tilde{\boldsymbol{X}}(k)$ features only Boolean entries. Denoting with $N_{b,n} = \lceil \log_2(N_n + 1) \rceil$ and $N_{b,t} = \lceil \log_2(N_t + 1) \rceil$ the minimum number of bits required to represent in base-2 the values of N_n and N_t, respectively, the size of $\tilde{\boldsymbol{X}}(k)$ is equal to

$$d_1 = N_p(N_{b,n} + N_{b,t}) + L. \tag{10}$$

This is the number of nodes in the first layer of κ_{NN}. Note that the information on which specific pallet is headed to

which target or has the highest priority is discarded by this encoding, since it is not relevant for the sake of optimally allocating the plant commands, as discussed above.

Algorithm 3.1 - State encoding procedure

1) Establish the priorities among pallets by ranking the values t_i, $i = 1, \dots, N_p$ from largest to smallest, and correspondingly build the auxiliary node/target vector:
$$\boldsymbol{Y} = \left[[h_{i_1}, g_{i_1}], \dots, [h_{i_{N_p}}, g_{i_{N_p}}]\right]^T,$$
such that $t_{i_j} > t_{i_{j-1}}$, $\forall j \in [2, N_p]$. Vector $\boldsymbol{Y}$ now features $2N_p$ integer variables (i.e. the nodes and targets of each pallet, ordered by priority);
2) Convert each entry of $\boldsymbol{Y}$ in base-2. We denote the resulting vector of Boolean variables as ${}_2\boldsymbol{Y}$;
3) Assuming that the plant features L machines identified by node indexes $l_1, \dots, l_L$, build a further vector of Boolean variables $\boldsymbol{O} = [o_1, \dots, o_L] \in \{0,1\}^L$ such that:
$$o_j = \begin{cases} 1 & \text{if } h_i = l_j \wedge m_i = 1 \text{ for some } i = 1, \dots, N_p \\ 0 & \text{otherwise} \end{cases}$$
4) Return $\tilde{\boldsymbol{X}} = \left[{}_2\boldsymbol{Y}^T, \boldsymbol{O}^T\right]^T$.

3.2 Safeguard Mechanism

The neural controller κ_{NN} is inevitably prone to errors when the state is not part of the employed training set. If the generalization capability of the network is high (i.e., it is able to correctly compute the input sequence in face of previously unseen state values), we can expect such errors to occur rarely. However, the considered application is safety-critical: even one error can lead to damage to the plant and/or to long downtime periods. In order to retain the safety guarantees provided by the predictive controller, we therefore introduce a Safeguard Mechanism (SM), see Fig. 1, given by Algorithm 3.2.

Algorithm 3.2 - Safeguard Mechanism

At each time step k:
1) Compute $\boldsymbol{U}_{\text{NN}}(k)$ as in (7);
2) Feasibility check:
if $\boldsymbol{U}_{NN}(k) \in \mathcal{U}(\boldsymbol{X}(k)) \wedge f(\boldsymbol{X}, \boldsymbol{U}_{NN}(k)) \in \mathcal{F}$
then set $e(k) = 0$ and go to step 4)
else set $e(k) = 1$ and go to step 3)
3) Compute the MPC input $\boldsymbol{U}_{\text{MPC}}(k)$ (3) by solving problem $\mathcal{P}(\boldsymbol{X}(k))$ and update the training data-set as:
$$\mathcal{D}(k) = \mathcal{D}(k-1) \cup (\boldsymbol{X}(k), \boldsymbol{U}_{\text{MPC}}(k)) \qquad (11)$$
4) If $e(k) = 0$ then apply the input $\boldsymbol{U}_{\text{NN}}(k)$ to the plant, else apply the input $\boldsymbol{U}_{\text{MPC}}(k)$

At step 2) of this Algorithm, other features of the control input can be verified as well to discard low-performing solutions: for example, a null control action is always feasible but not wanted. Note that whenever step 3) is executed, the cardinality of the training data set increases by one, i.e. $M(k) = M(k-1) + 1$, see (8).

3.3 Training Process

When the number $M(k)$ of data points in the training set is small, we can expect poor generalization performance of the FFNN. On the other hand, building a-priori a vast training set is not a good strategy, since the number of possible states exponentially increases with the number of nodes, pallets, and machines, and one would need to solve the MPC optimization for each one. Moreover, many states may result to be irrelevant because never actually reached during optimal plant operation. To cope with this problem, we propose a two-step procedure (Procedure 3.3) to train a reasonably well-performing controller and to gradually improve it over time.

Procedure 3.3 - Continual learning

1) In the first phase, an initial data-set of $\underline{M}$ pairs is generated with the plant controlled by the MPC law (either in simulation, possibly with more parallel runs, or on the real plant). A first instance of κ_{NN} is trained in background with the obtained initial data-set. Then, at each subsequent time step, the input produced by the MPC is compared with that of the FFNN: an error counter $\varepsilon(k)$ is incremented by one each time the FFNN produces an input that is different from that of the MPC law. Any state-input pair $(\boldsymbol{X}(k), \boldsymbol{U}_{\text{MPC}}(k)) \neq (\boldsymbol{X}(k), \boldsymbol{U}_{\text{NN}}(k))$ is added to the training set, increasing its cardinality $M(k)$ by one. Whenever $M(k) = \underline{M} + \Delta M$, where ΔM is a user-chosen integer, we set $\underline{M} = M(k)$ and run a new training procedure to update κ_{NN}, while resetting the error counter, $\varepsilon(k) = 0$. When the rate $\varepsilon(k)/K$, where K is a suitably-defined time window, falls below a chosen threshold, the next phase is triggered.

2) In Phase 2, the system is ran under the Safeguard Mechanism of Algorithm 3.2. In a way similar to Phase 1, whenever $M(k) = \underline{M} + \Delta M$, a new training procedure is ran in background using the whole data-set. When the resulting function κ_{NN} is deployed we set $\underline{M} = M(k)$ and reset the error counter $\varepsilon(k)$ to zero.

Phase 1 of Procedure 3.3 can be seen as running Algorithm 3.2 while forcing $e(k) = 1$, i.e. letting the MPC law control the plant while the FFNN learns its behavior. Phase 2 corresponds instead to the proposed safeguarded learning approach, where it is expected that most of the time the FFNN computes a safe and well-performing control action, and the MPC law must intervene only sporadically. This expectation is confirmed by our tests on a high-fidelity simulator of a real-world plant, described next.

4. CASE STUDY: DE-MANUFACTURING PLANT

We consider a de-manufacturing pilot plant, located in the laboratory of STIIMA-CNR in Milan, see Fig. 4 and Colledani et al. (2014). It is designed to recover second-hand electronic boards and it consists of four machines: M1 is a load/unload robot cell, M2 a testing machine, M3 a reworking machine, and M4 a dismantling one. The process is characterized by the following routine: first, the pallet is moved to the robot cell, that loads a new electronic board on it (or replaces the board if the pallet is not empty). Then, the first target is the testing machine. The test outcome can be one of three: A) the board works properly; B) the board can be repaired; C) the board must be dismantled. In the first and third cases, the target becomes M1 (respectively M4) and the board is unloaded from the plant, while in case B) the pallet is sent to the

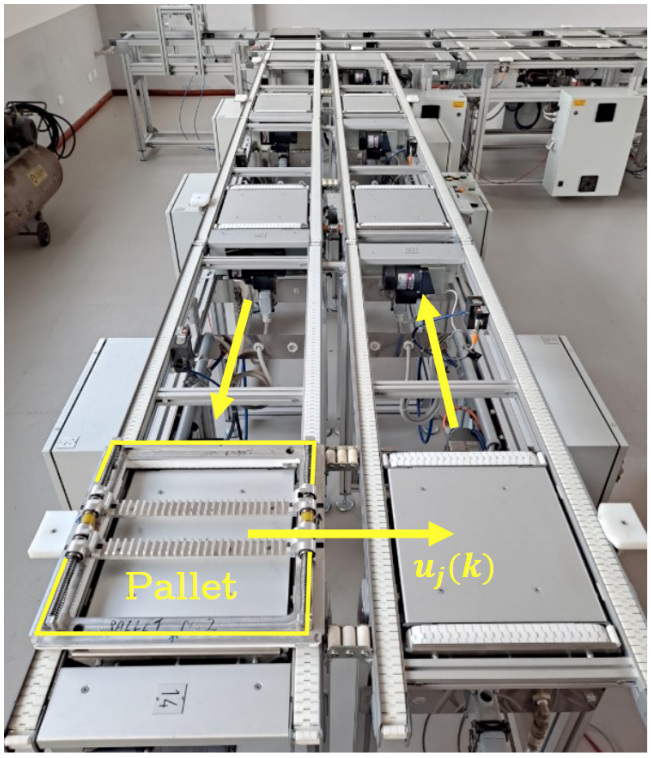

Fig. 3. Real plant on which the proposed control architecture has been tested.

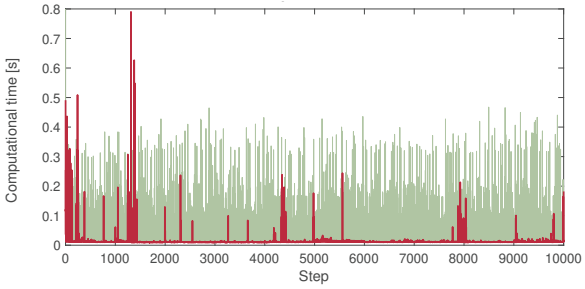

Fig. 4. Comparison between the computational time required by the proposed control architecture (red line) and that required by the MPC (light-green line).

reworking machine M3, and subsequently it is tested again in M2 to check if the fixing was successful or the part must be definitively discarded. The handling system consists of $N_u = 53$ control signals $N_n = 36$ nodes (including the $L = 4$ machines). We consider five pallets in the test, i.e., $N_p = 5$. The HPRC has been designed with a prediction horizon $N = 50$ steps, see Boffadossi et al. (2021). The control structure that we propose in this paper has been validated directly on the real plant, while the results of the long-rage simulation discussed in the following lines have been obtained using a high-fidelity model.

We adopt a FFNN with $N_l = 4$ layers, featuring $d_1 = 49$ encoded input arguments, $d_2 = 200$ nodes, $d_3 = 100$ nodes and $d_4 = 53$ outputs, according to Section 3.1 and to the characteristics of the plant. We firstly trained the FFNN in Phase 1 of Procedure 3.3. When the accuracy of the network reached a set threshold of 1.5%, the resulting training $\mathcal{D}$ is found to contain 71.9% of the total explored states, indicating that in the remaining 28.1% of state

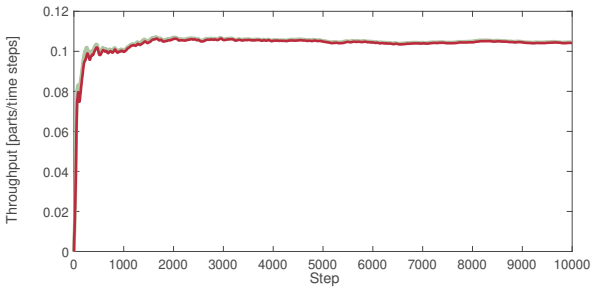

Fig. 5. Comparison of the throughput achieved by the proposed control architecture (red line) and that scored by the MPC (light-green line).

values the network was able to correctly generalize the learned behavior. Then, we switched to Phase 2 and tested the control architecture for 10^4 steps, setting $\Delta M = 40$.

As depicted in Fig. 4, the computational time required by the proposed control architecture is significantly lower compared to the computational time required by the MPC. In particular, the mean computational time is $0.054s$ when the MPC is controlling the plant, while it is five times less when the control is performed by the SM. Furthermore, the two cases have a different distribution of values, for the SM approach the computational time is greater than $0.1s$ only for the 0.94% of time steps, while for the MPC approach the percentage increases to 17.91%. The computational advantages achieved by the proposed approach would be even more apparent in presence of a larger number of nodes, machines, or more extensive tree-search routines. In fact, in this case, the computational burden of MPC would increase significantly, while that of the described architecture would remain almost unchanged. Remarkably, the aforementioned reduction of the computational cost does not negatively affect the system throughput which, as shown in Fig. 5, is almost equivalent to the throughput scored by the MPC. Considering the overall throughput obtained at the end of the test, in case of the SM approach the value is less than 0.0004 [parts/time steps] w.r.t the MPC approach, corresponding just to a decrease of about 0.38%.

Eventually, it is worth noticing that, owing to the continual learning approach, both the throughput mismatch and the computational time of the proposed control architecture, which is related to the frequency with which the MPC is executed by the SM as a fallback controller, improve during the plant operation.

5. CONCLUSIONS

In this paper, the use of Neural Networks (NNs) to approximate Model Predictive Control (MPC) laws applied to discrete-time manufacturing system has been discussed. The aim of this approach is to address the implementation issues of MPC, namely its computational burden, while preserving its optimal performances and constraint satisfaction capabilities. When the NN is used to control the real plant, safety of operations is ensured by a Safeguard Mechanism which switches to the MPC whenever a potential safety violation is detected. A continual learning strategy has then been proposed to gradually improve the FFNN performance. The reported results indicate that the

proposed architecture preserves the performances of MPC, measured by the system's throughput, while drastically reducing the computational time required. Future research will address the improvement of the Safeguard Mechanism, by decoupling it from the MPC law in order to keep fast computational times. This can be obtained for example by adopting a safe heuristic to correct the NN command when the MPC runs into high computational peaks, and compute the optimal input with MPC in the background only for the sake of NN improvement.

REFERENCES

Bengio, Y., Goodfellow, I., and Courville, A. (2017). *Deep learning*, volume 1. MIT press Massachusetts, USA:.

Boffadossi, R., Fagiano, L., Tanaskovic, M., Cataldo, A., Tanaskovic, M., and Lauricella, M. (2021). Advanced hierarchical predictive routing control of a smart demanufacturing plant. *2021 European Control Conference (ECC)*, 1774–1779.

Bucki, R., Chramcov, B., and Suchánek, P. (2015). Heuristic algorithms for manufacturing and replacement strategies of the production system. 21(4), 503–525.

Byrne, M. and Chutima, P. (1997). Real-time operational control of an FMS with full routing flexibility. *International Journal of Production Economics*, 51(1), 109 – 113. Raising the Competitive Edge in Manufacturing.

Canale, M., Fagiano, L., and Milanese, M. (2010). Efficient model predictive control for nonlinear systems via function approximation techniques. *IEEE Transactions on Automatic Control*, 55(8), 1911–1916. doi: 10.1109/TAC.2010.2049776.

Cataldo, A., Morescalchi, M., and Scattolini, R. (2019). Fault tolerant model predictive control of a demanufacturing plant. *The International Journal of Advanced Manufacturing Technology*, 9(12), 4803–4812.

Cataldo, A. and Scattolini, R. (2016). Dynamic pallet routing in a manufacturing transport line with model predictive control. *IEEE Transactions on Control Systems Technology*, 24(5), 1812–1819. doi: 10.1109/TCST.2015.2507062.

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

Colledani, M., Copani, G., and Tolio, T. (2014). Demanufacturing systems. *Procedia CIRP*, 17, 14–19. Variety Management in Manufacturing.

Das, S.K. and Nagendra, P. (1997). Selection of routes in a flexible manufacturing facility. *International Journal of Production Economics*, 48(3), 237 – 247.

Fagiano, L., Tanaskovic, M., Mallitasig, L.C., Cataldo, A., and Scattolini, R. (2020). Hierarchical routing control in discrete manufacturing plants via model predictive path allocation and greedy path following. In *2020 59th IEEE Conference on Decision and Control (CDC)*, 5546–5551. doi:10.1109/CDC42340.2020.9303933.

Franco, L. (2006). Generalization ability of boolean functions implemented in feedforward neural networks. *Neurocomputing*, 70(1), 351–361. Neural Networks.

Gupta, Y.P., Gupta, M.C., and Bector, C.R. (1989). A review of scheduling rules in flexible manufacturing systems. *International Journal of Computer Integrated Manufacturing*, 2(6), 356–377. doi: 10.1080/09511928908944424.

Johansen, T. (2004). Approximate explicit receding horizon control of constrained nonlinear systems. *Automatica*, 40, 293–300.

Kang, H.S., Lee, J.Y., Choi, S., Kim, H., Park, J.H., Son, J.Y., Kim, B.H., and Noh, S.D. (2016). Smart manufacturing: Past research, present findings, and future directions. *International Journal of Precision Engineering and Manufacturing-Green Technology*, 3, 111–128.

Koren, Y., Gu, X., and Guo, W. (2018). Reconfigurable manufacturing systems: Principles, design, and future trends. *Frontiers of Mechanical Engineering*, 13(2), 121–136. doi:10.1007/s11465-018-0483-0.

Koren, Y. and Shpitalni, M. (2010). Design of reconfigurable manufacturing systems. *Journal of Manufacturing Systems*, 29(4), 130–141. doi: 10.1016/j.jmsy.2011.01.001.

Kouiss, K., Pierreval, H., and Mebarki, N. (1997). Using multi-agent architecture in fms for dynamic scheduling. *Journal of Intelligent Manufacturing*, 8, 41–47.

Moro, A.R., Yu, H., and Kelleher, G. (2002). Hybrid heuristic search for the scheduling of flexible manufacturing systems using petri nets. *IEEE Transactions on Robotics and Automation*, 18(2), 240–245.

Peng, C. and Chen, F. (1998). Real-time control and scheduling of flexible manufacturing systems: An ordinal optimisation based approach. *The International Journal of Advanced Manufacturing Technology*, 14, 775–786.

Potdar, K., Pardawala, T.S., and Pai, C.D. (2017). A comparative study of categorical variable encoding techniques for neural network classifiers. *International Journal of Computer Applications*, 175, 7–9.

Saygin, C., Chen, F., and Singh, J. (2001). Real-time manipulation of alternative routeings in flexible manufacturing systems: A simulation study. *The International Journal of Advanced Manufacturing Technology*, 18, 755–763.

Souier, M., Hassam, A., and Sari, Z. (2010). *Metaheuristics for Real-time Routing Selection in Flexible Manufacturing Systems*, 221–248. Springer London, London.

Vargas-Villamil, F.D. and Rivera, D.E. (2000). Multilayer optimization and scheduling using model predictive control: application to reentrant semiconductor manufacturing lines. *Computers & Chemical Engineering*, 24(8), 2009 – 2021.

Vargas-Villamil, F.D. and Rivera, D.E. (2001). A model predictive control approach for real-time optimization of reentrant manufacturing lines. *Computers in Industry*, 45(1), 45 – 57.

Zheng, P., Wang, H., Sang, Z., Zhong, R.Y., Liu, Y., Liu, C., Mubarok, K., Yu, S., and Xu, X. (2018). Smart manufacturing systems for industry 4.0: Conceptual framework, scenarios, and future perspectives. *Frontiers of Mechanical Engineering*, 13, 137–150.