Article

A Machine Learning Algorithm That Experiences Evolutionary Algorithm's Predictions – Application to Optimal Control.

Viorel Mînzu * and Iulian Arama

Posted Date: 12 December 2023

doi: 10.20944/preprints202312.0876.v1

Keywords: evolutionary algorithm; machine learning; optimal control; simulation.

Article

A Machine Learning Algorithm that Experiences Evolutionary Algorithm's Predictions – Application to Optimal Control

Viorel Mînzu 1,* and Iulian Arama 2

1 Control and Electrical Engineering Department, "Dunarea de Jos" University, 800008 Galati, Romania
2 Informatics Department, "Danubius" University, 800654 Galati, Romania; iulian.arama@univ-danubius.ro
* Correspondence: viorel.minzu@ugal.ro

Abstract: Using metaheuristics, such as the Evolutionary Algorithm (EA), within control structures is a realistic approach for certain optimal control problems. Their role is often predicting the optimal control values over a prediction horizon using a process model (PM). The big computational effort sometimes causes problems. Our work addresses a new issue: whether a machine learning (ML) algorithm could "learn" the optimal behaviour of the couple (EA, PM). A positive answer is given by proposing datasets apprehending this couple's optimal behaviour and appropriate ML models. Following a design procedure, a number of closed-loop simulations will provide the sequences of optimal control and state values, which are collected and aggregated in a data structure. For each sampling period, datasets are extracted from the aggregated data. The ML algorithm experiencing these datasets will produce a set of regression functions. Replacing the EA predictor with the ML model, new simulations are carried out, proving that the state evolution is almost identical. The execution time decreases drastically because the PM's numerical integrations are totally avoided. The performance index equals the best-known value. In different case studies, the ML models succeeded in capturing the optimal behaviour of the couple (EA, PM) and yielded efficient controllers.

Keywords: evolutionary algorithm; machine learning; optimal control; simulation

1. Introduction

Controlling a process subjected to a performance index is a usual task in process engineering. Theoretical control laws can be implemented in favourable situations where the process has certain mathematical properties. On the other hand, when the process has profound nonlinearities, or its model is uncertain, imprecise or incomplete, using metaheuristic algorithms (EA, Particle Swarm Optimization, etc.) (see [1–3]) within an appropriate control structure could be a realistic solution. Control engineering recorded many examples of using metaheuristics [4–9] owing to their robustness and capacity to cope with complex problems.

Generally speaking, the metaheuristic algorithm's role within a controller is to predict the optimal (quasi-optimal) control values. The predictor forecasts the optimal control sequence for a prediction horizon, and the controller decides the next optimal control value. A control structure adequate for this kind of controller is Receding Horizon Control (RHC) [10–13]. It can be used to solve Optimal Control Problems (OCPs) and includes a Process Model (PM) [14–16].

The authors have studied implementing the prediction module using EAs in previous works [16–18]. EAs proved a realistic solution due to many possibilities to reduce the predictor's execution time. The latter is the big challenge of this approach, mainly appropriate for slow processes with large sampling periods because of the predictions' computation time. Extending their applicability is a challenge, involving techniques and control structures diminishing the execution time [19–21]. One can consider that our work addresses the execution time's decreasing, but the proposed machine learning (ML) task (see [22–25]) largely exceeds this topic.

This work proposes an interesting issue: whether a machine learning algorithm could "learn" the optimal behaviour of the couple (EA, PM). A positive answer would have very favourable consequences for the controller implementation. The substitution of the couple (EA, PM) with an accurate ML algorithm could cause the predictions' computation time to decrease significantly and the controller's structure to be much simpler. In this context, the ML algorithm has to construct usable *models experiencing datasets catching the optimal behaviour of the couple* (EA, PM). Concretely, this work's main objective is to answer the above-mentioned question positively by proposing

- realistic datasets apprehending the optimal behaviour of the couple (EA, PM) and
- appropriate ML models.

Hence, this work will propose an ML model that can substitute the couple (EA, PM) inside the optimal controller while keeping the control performances. In other words, the ML model would be equivalent in a certain sense to the optimal behaviour of the EA plus PM; both entities are predictors [26]. According to our knowledge, this "intelligent equivalence" can be considered a new issue.

Before using the controller within the closed-loop system in real-time, a simulation program must validate the designed controller through the Process's evolution along the imposed control horizon; the Process and the PM are considered identical. The simulation's results are usually sequences of control and state variables' values along the control horizon that can be stored or recorded. This data is a mark of the system's evolution made up of the control profile (sequence of control values) and state trajectory (sequence of state variables' values). Repeating the simulation many times, we can aggregate these data and generate datasets for the ML algorithm. The simulations are conducted offline, so there is no execution time constraint.

An important remark is that we do not need data from the real Process to be included in the datasets. The predictor module predicts optimal trajectories using only the EA and PM, whose inter-influence must be captured by the datasets. When an accurate ML model replaces the initial predictor, the controller should behave quasi-identically within simulations, which is our desideratum.

This work will ascertain the previous considerations and propose an approach to construct the datasets and ML model, starting from the OCP to solve. Besides general considerations, we will apply the proposed methods to a specific OCP, exemplifying the proposed methods and algorithms to make the presentation easy to follow.

We shall consider OCPs with a final cost that uses predictions to exemplify the equivalence mentioned above and implement ML controllers. Section 2 recalls the general approach to solving the OCPs using EAs developed in previous authors' works [16,19–21]. The Park Ramirez Problem (PRP) (see [21,27,28]) is a kind of benchmark problem already treated in this context, which is addressed as an example. This paper will report partially previous results for comparison and take over the EA predictor's implementation. Section 2 is mainly necessary because *the optimal control using the* EA *will supply the datasets for* ML. Although the aspects presented in this section are not among this paper's contributions, they introduce the notations and keep the discourse self-contained.

Section 3 answers the following three basic questions:

- What data do we need to capture the optimal behaviour of the couple (EA, PM)?
- How do we generate the datasets for the ML algorithm?
- What ML model can be used to design an appropriate controller (we will name it ML controller)?

These a general questions, each of which has subsumed aspects to clarify. Section 3.1, describing the proposed method, answers these questions succinctly; details will be given in the next sections. It also establishes a controller design procedure.

The starting point in our approach is that we have already solved the considered OCP, and the implemented controller has a prediction module using a specific EA. Section 3.2.1 describes an algorithm achieving the closed-loop simulation along the control horizon devoted to the EA predictor. The simulation will record the sequence of optimal control values - the optimal control profile - and the optimal trajectory (sequence of states). The two sequences can be regarded as a series of couples (state, control value), each couple associated implicitly with a sampling period.

Repeating the simulation M times (e.g. M equals 200), data concerning these optimal evolutions of the closed loop is collected and aggregated into a data structure presented in section 3.2.2. These aggregated data structure characterizes the optimal behaviour of the couple (EA, PM) globally, that is, for the whole control horizon. The extraction of datasets characterizing each sampling period from the aggregated data creates the premise to find a model of optimal behaviour for each sampling period. Section 3.2.3 describes how the aggregated data is split into datasets for each sampling period. Moreover, training and testing datasets are created.

In section 3.3, the ML models are constructed according to an important choice. The ML model will be a set of regression functions; a linear regression function will be determined for each sampling period [29–31]. The first reason for this choice is the model's simplicity, which is important, especially when the control horizon is large. Secondly, the control law will be directly determined. This fact is appropriate for the controller implementation, which is now straightforward. Section 3.3.1 proposes simple models with linear terms for each state variable. In contrast, section 3.3.2 uses the stepwise regression strategy to generate regression functions, which are allowed to include nonlinear terms as the interactions. The ML model succeeds in reproducing the optimal behaviour of the EA predictor with a high accuracy.

The simulation of the closed-loop system is the way to test the generalization aptitude of the new predictor after its implementation. In its first part, Section 4 describes an algorithm that simulates the closed-loop system using the ML controller along the control horizon. The second part compares the simulation results to those anteriorly obtained with the EA predictor for the PRP case [21]. The state evolutions are practically identical, and the performance index equals the best of M evolutions, which is true for the two types of regression functions.

The simulation results (in the PRP case and other case studies not presented in this paper) proved that the proposed ML models succeeded in apprehending the optimal behaviour of the couple (EA, PM) and engendered efficient controllers.

We consider that our work has the following findings:

- the interesting issue itself, which is to find an ML model experiencing the datasets generated by an EA (or another metaheuristic), trying to capture the latter's optimal behaviour.
- the dataset's construction as a dynamic trace of the EA predictor, aggregating the trajectories and CPs.
- the dataset extraction for each sampling period as a premise to find a temporally distributed ML model.
- the design procedure for the ML controller and all associated algorithms (simulation and models' construction algorithms).
- the *outstanding decrease in the ML controller's execution time*.

Special attention was addressed to the implementation aspects such that the interested reader can find support to apprehend and eventually reproduce parts of this work or use it in other projects. With this aim in view, all algorithms used in this work are implemented, the associated scripts are attached as supplementary materials, and all the necessary details are given in the Appendixes.

2. Optimal control using Evolutionary Algorithms.

This section recalls the general approach to solving the OCPs using EAs developed in anterior papers [19,21]. The minimal elements presented here introduce the notations and keep the discourse self-contained.

2.1. Optimal control problems with a final cost

The structure of an OCP being well-known, we consider in the sequel only the defining elements adapted to the problem taken as an example in this section. Rigorous mathematical details will be avoided to simplify the presentation.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 December 2023 doi:10.20944/preprints202312.0876.v1

2.1.1. Process Model

In our approach, the controller includes a process model constituted by algebraic and ordinary differential equations:

$$\begin{cases} \dot{X}(t) = [f_1(X,U,W) \cdots f_n(X,U,W)]^T \\ g_i(X,U,W) = 0, \quad i = 1,\cdots,p \end{cases} \text{, where} \tag{1}$$

$X(t) = [x_1(t) \cdots x_n(t)]^T$ - a vector with n state variables;

$U(t) = [u_1(t) \cdots u_m(t)]^T$ - a vector with m control variables;

An example of a Process Model is equation (7) from subsection 2.1.4

2.1.2. Constraints

There are many constraint types, but we mention only those used in the case study presented in this paper.

Control horizon: $t \in [t_0, t_{final}]$; $t_0 = 0$, (2)

Initial state: $X(0) = X_0 \in \mathbf{R}^n$. (3)

Bound constraints: $u_j(t) \in \Omega_j \triangleq \left[u_{\min}^j,\ u_{\max}^j\right]$; $j = 1,\cdots,m$; $0 < t < t_{final}$. (4)

The values $u_{\min}^j$, $u_{\max}^j$ are the technological bounds of the variable $u_j(t)$.

If T is the sampling period of the control system, we can divide the control horizon into H sampling periods:

$$t_{final} = H \times T.$$

2.1.3. Cost function

The problem is to determine the control function $U(\cdot)$ optimizing (max or min) a specific cost (objective) function *J*, whose general form is given below:

$$J(U(\cdot), X_0) = \int_0^{t_{final}} L(X(\tau), U(\tau))d\tau + J_{final}. \tag{5}$$

The function L determines the integral component (Lagrange term) of function J, while J_{final} (Mayer term) rewards (or penalizes) the final state (in most cases).

Remark 1. *When the final cost is present, and the controller makes predictions, whether or not there is an integral term, the prediction horizon must end at the final time, involving the biggest computational complexity.*

Given remark 1, we consider only the final cost, a situation suited to our OCP (see next section).

$$J\left(U(\cdot), X_0\right) \triangleq J_{final} = J\left(X\left(t_{final}\right)\right).$$

The problem's solution is the function $U(\cdot)$ that engenders the cost function's optimal value. This value, J_0, is called the performance index:

$$J_0 = \max_{U(\cdot)} J\left(X\left(t_{final}\right)\right) \triangleq \max_{U(\cdot)} J\left(U(\cdot), X_0\right). \tag{6}$$

2.1.4. An example of OCP with a final cost

Park–Ramirez problem (PRP) is a kind of benchmark problem ([21,27,28]) that can exemplify a final cost OCP. The nonlinear process models a fed-batch reactor, which produces secreted protein. This problem has been addressed in many works to study integration methods.

Process Model (PM)

$$\begin{aligned}
\dot{x}_1 &= g_1 \cdot (x_2 - x_1) - \frac{u}{x_5} \cdot x_1 . \\
\dot{x}_2 &= g_2 \cdot x_3 - \frac{u}{x_5} \cdot x_2 . \\
\dot{x}_3 &= g_3 \cdot x_3 - \frac{u}{x_5} \cdot x_3 \\
\dot{x}_4 &= -7.3 \cdot g_3 \cdot x_3 + \frac{u}{x_5} \cdot (20 - x_4) \\
\dot{x}_5 &= u \\
g_1 &= \frac{4.75 \cdot g_3}{0.12 + g_3} \\
g_2 &= \frac{x_4}{0.1 + x_4} \cdot e^{-5.0 \cdot x_4} \\
g_3 &= \frac{21.87 \cdot x_4}{(x_4 + 0.4)(x_4 + 62.5)}
\end{aligned} \tag{7}$$

The state vector $X(t) = [x_1(t) \cdots x_5(t)]^T$ regroups the following physical parameters: $x_1(t)$ - concentration of secreted protein, $x_2(t)$ - concentration of total protein, $x_3(t)$ - density of culture cell, $x_4(t)$ - concentration of substrate, $x_5(t)$ -holdup volume.

It holds: n=5; m=1; $U(t) = u(t) \in \mathbf{R}$.

Constraints

Control horizon:	$t \in [t_0, t_{final}]$; $t_0 = 0$, $t_{final} = 15h$.
Initial state:	$X(0) = X_0 = [0,\ 0,\ 1,\ 5,\ 1]^T \in \mathbf{R}^5$.
Bound constraints:	$u(t) \in \Omega \triangleq [0,\ 2]$; $0 < t < t_{final}$.

Performance Index

$$J_0 = \max_{u(t)} J\left(x\left(t_{final}\right)\right) = \max_{u(t)} X_1(t_{final}) \cdot X_5(t_{final})$$

An open-loop solution can not be used in real-time because the Process and PM have different dynamics (even when there are small differences); this will produce unpredictable efficiency. We want to generate a controlled optimal process (a closed-loop solution) starting from a given X_0 whose final cost should be J_0.

2.2. A discrete-time solution based on EAs

A control structure that can generate the optimal solution is RHC ([16,21]). Its controller includes the process model and a prediction module (see Figure 1). The last one predicts, at each moment *kT*, the optimal control sequence until the final time. Then, the controller outputs this sequence's first element as the optimal value and inputs the next Process's state.

The prediction module using an EA proved a realistic solution due to many possibilities to reduce its execution time [19–21].

To use an EA, we append to our OCP the *discretization constraint*:

$$U(t) = U\ kT) \text{, for } k \cdot T \le t < (k+1) \cdot T;\ k = 0, \cdots, H-1 .$$

So, the control variables are step functions. For the sake of simplicity, the time moment $k \cdot T$ will be denoted k in the sequel. For example, inside the sampling period $[kT,(k+1)T)$, the control vector is

$$U(kT) \equiv U(k) \triangleq \left[u_1(k),\cdots,u_m(k)\right]^T$$

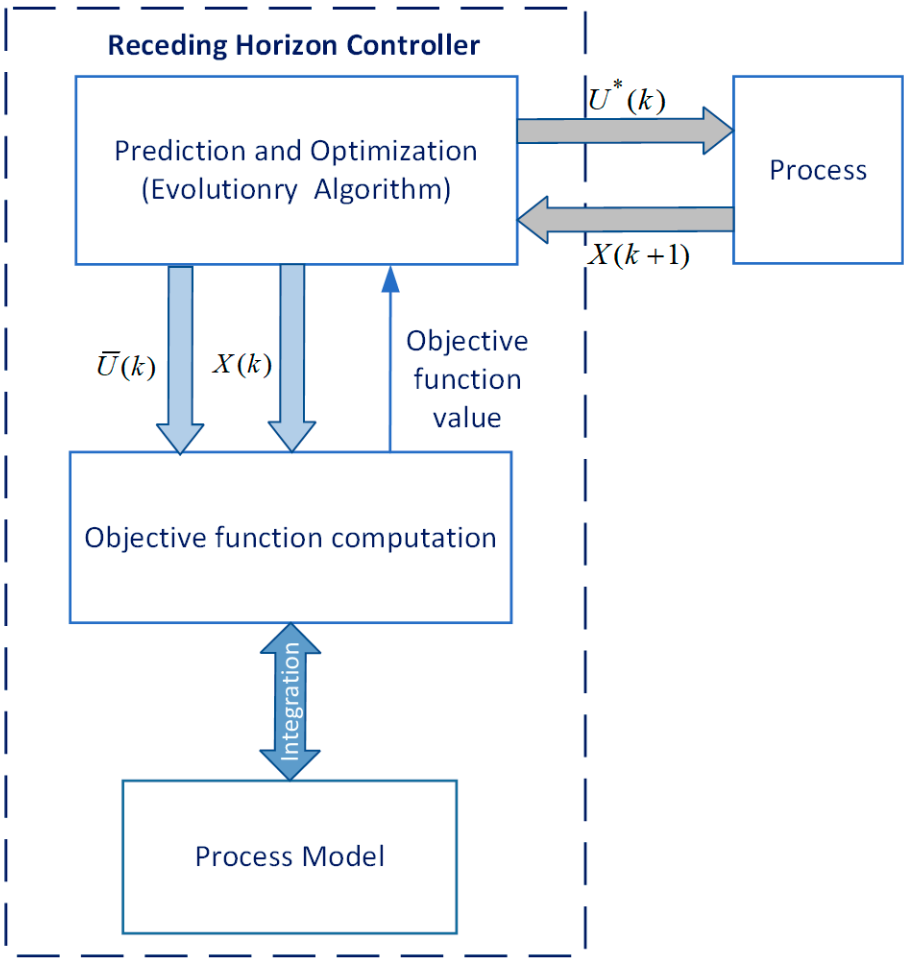

Receding Horizon Controller
Prediction and Optimization (Evolutionry Algorithm)
$U^*(k)$
Process
$X(k+1)$
$\bar{U}(k)$
$X(k)$
Objective function value
Objective function computation
Integration
Process Model

Figure 1. The control structure using EAs.

We name a "control profile" (CP) a complete sequence of H control vectors, $U(0),\ U(1),\ \cdots\ U(H-1)$. It will generate the transfer diagram drawn in Figure 2.

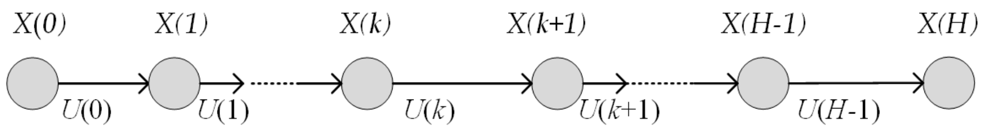


Figure 2. The state trajectory yield by a control profile.

The EA yields candidate predictions over prediction horizons and evaluates the cost function *J*. For the sampling period [*k*, *k*+1), a candidate prediction is a control sequence having the following structure:

$$\bar{U}(k) = \langle U(k),\ldots,U(H-1)\rangle . \tag{8}$$

The vector $X(k)$ is the process's current state. It is also the initial state for the candidate prediction with *H*-*k* elements. This fact justifies the appellation of "Receding Horizon Control". Using equations (1) and (8), the EA also calculates the corresponding state sequence (with *H*-*k*+1 elements):

$$\overline{X}(k) = \langle X(k), \ldots, X(H) \rangle . \tag{9}$$

At convergence, the EA returns (to the controller) the optimal prediction sequence, denoted $\overline{V}(k)$:

$$\overline{V}(k) \triangleq \arg\max_{\overline{U}(k)} J(\overline{U}(k), X(k)) = \langle V(k), \ldots, V(H-1) \rangle . \tag{10}$$

Finally, the first value of the sequence, *V*(*k*), becomes the controller's best output, denoted $U^*(k)$, sent toward the process:

$$U^*(k) \triangleq V(k) . \tag{11}$$

Remark 2. *The optimal control* $U^*(k)$ *is also a function of the current state* $X(k)$*, which does not appear as a distinct argument in (11) to keep the notation simple and easy to follow. Nevertheless, this dependence is essential for the machine learning models as well.*

The other values of the sequence $\overline{V}(k)$ are forgotten, and the controller will treat the next sampling period [*k*+1, *k*+2).

The controller equipped with the EA constructs the optimal CP for the given initial state $X_0=X(0)$ and the entire control horizon, concatenating the optimal controls $U^*(k)$ $k = 0, \cdots, H-1$. It forces the system to pass through a sequence of "optimal states", the optimal trajectory $\Gamma(X_0)$:

$$\Omega(X_0) \triangleq \langle U^*(0),\ U^*(1), \ldots,\ U^*(H-1) \rangle \tag{12}$$

$$\Gamma(X_0) \triangleq \langle X_0, X^*(1), \ldots, X^*(H) \rangle . \tag{13}$$

These two sequences completely characterize the optimal evolution of the closed loop over the control horizon. Theoretically, the optimal cost function will reach the value J_0 if the process and its model are identical. Practically, this value will be very close to J_0, such that $\Omega(X_0)$ is a quasi-optimal solution of our OCP.

The flowchart of the prediction module is drawn in Appendix A. The interested reader can find the main characteristics of the implemented EA in [21] or the supplementary materials appended to this work. In our implementation, the EA's code is presented in the script `RHC_Predictor_EA.m`. The initial population is generated using the control variables' bounds. The cost function is coded within the file `eval_PR_step.m`. All scripts are included in the folder `ART_Math`, as the other functions called by the predictor and implementing the EA's operators and the PM.

3. Controller based on machine learning

3.1. The general description of the proposed method

The PRP and other problems of this kind allow the validation of the designed optimal controller by simulating the control loop.

Simulation of the control loop is an important design tool in this context, which can supply the sequence $\Omega(X_0)$ and $\Gamma(X_0)$ ((12) and (13)) that describe the quasi-optimal evolution of the loop.

Repeating the control loop simulation M times, we obtain M different optimal (actually quasi-optimal) couples (CP – trajectory), even if the initial states would be identical due to the EA's stochastic character. Moreover, in the case of PRP, the initial state can be perturbed to simulate the imperfect realization of the initial conditions when launching a new batch. Let us consider a lot of the M simulations illustrated in Figure 3.

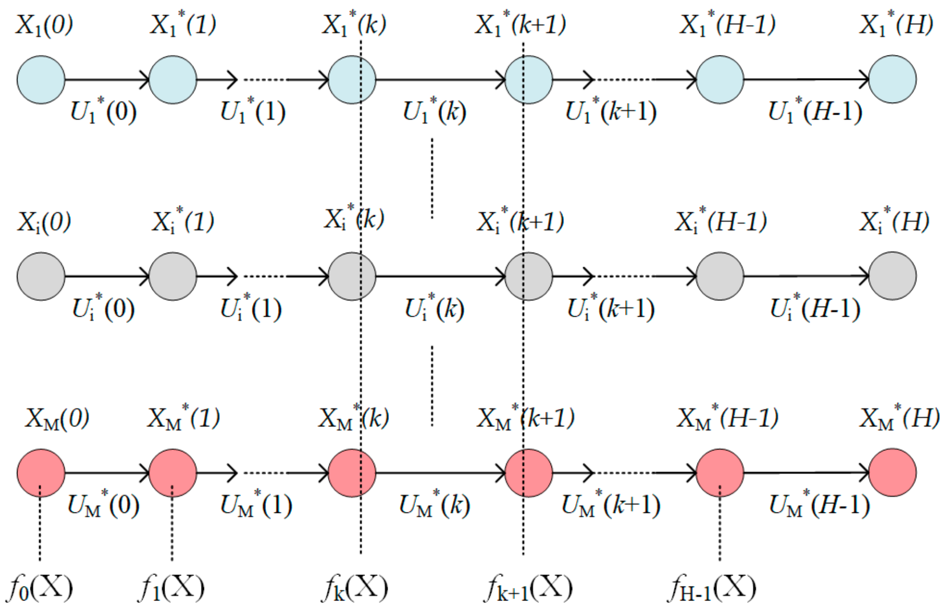


Figure 3. A set of M quasi-optimal trajectories produced by control loop simulation using a controller based on the EA and MP.

At step *k* of the control horizon, the controller must predict the optimal control output (sent toward the process) using its predictor module based on the EA and MP described before. Data concerning the same step have some common aspects:

- The initial process's state $X_i^*(k),\ 1 \le i \le M$ is input data for the EA.
- The prediction horizon is the same: $H-k$.
- The PM is the same.
- The M simulations use the same EA.

The EA calculates and returns the optimal control vector $U_i^*(k),\ 1 \le i \le M$. A dataset including the simulation results for step *k* can be constructed as follows:

The dataset gets the look of a table due to the transposition operator. If M is big enough, this dataset collects an essential part of the EA's ability to predict optimal control values for step *k*. It would be useful to answer the question: How can we generalize this ability for other current state values the process could access at step *k*? A machine learning algorithm is the answer. For example, linear regression (see [29,31]) can construct a function $f_k : \mathbf{R}^n \to \mathbf{R}^m$ for each $k,\ k=0,\ 1,\ldots,\ H-1$ using a dataset like that presented before.

Remark 3. *The linear regression function* f_k *models how the EA determines the optimal prediction at step k. The set of functions* $\Phi = \{f_k \,|\, k=0,\ 1,\ldots,\ H-1\}$ *is the couple (EA–PM) machine-learning model. The behaviour of the EA, which, in turn, depends on the PM, is captured by the set of functions* Φ.

Hence, it would be possible to successfully replace the predictor with EA by this set of functions and obtain a faster controller.

Logically, a few steps lead us to a design procedure for the optimal controller.

Design procedure

1. Implement a program to simulate the closed loop functioning over the control horizon (*H*) using the controller based on the EA. To simplify the presentation, we will call it in the sequel

ControlLoop_EA. The output data are the quasi-optimal trajectory and its associated control profile ($\Omega(X_0)$ and $\Gamma(X_0)$).

2. Repeat M times the module ControlLoop_EA to generate and save M quasi-optimal trajectories and their control profiles.
3. Extract, for each step *k*, datasets similar to Table 1 using the data saved at step 2.
4. Determine the machine-learning model (for example, the set of functions Φ) experiencing the M trajectories and control profiles.
5. Implement the new controller based on ML. It will be called ML_controller in the sequel.

Simulation of the closed loop using the new controller

6. Write a simulation program called ControlLoop_ML for the closed loop working with the new controller. This simulation will test the feasibility of the proposed method, the quality of the quasi-optimal solution, the performance index, and the execution time of the new controller.

Table 1. Dataset for step *k*.

X^T	U^T
$\left(X_1^*(k)\right)^T$	$\left(U_1^*(k)\right)^T$
……	……
$\left(X_M^*(k)\right)^T$	$\left(U_M^*(k)\right)^T$

The set of functions Φ can determine the optimal CP starting from a given state X_0, following the transfer diagram from Figure 2, and applying the functions $f_0, f_1, \ldots, f_{H-1}$ to the current state successively:

$$U_1^* = f_0(X_0);\; U_2^* = f_1(X_1);\; \ldots\; U_{H-1}^* = (X_{H-1}). \tag{14}$$

Remark 4. *The design procedure shows that the new controller is completely designed offline. No data collected online from the process is necessary. The result of this procedure is the new Controller, ML_controller, which could be used in real-time after a robustness analysis.*

3.2. Dataset generation for machine learning model

The first two steps of the design procedure will be described in this section, trying to keep generality. Only some aspects will refer to the PRP to simplify the presentation.

3.2.1. The simulation of the closed-loop system based on EA predictions

This subsection corresponds to step 1 of the design procedure. Figure 4 shows the flowchart of the simulation program for the closed loop (the script `ControlLoop_EA.m`).

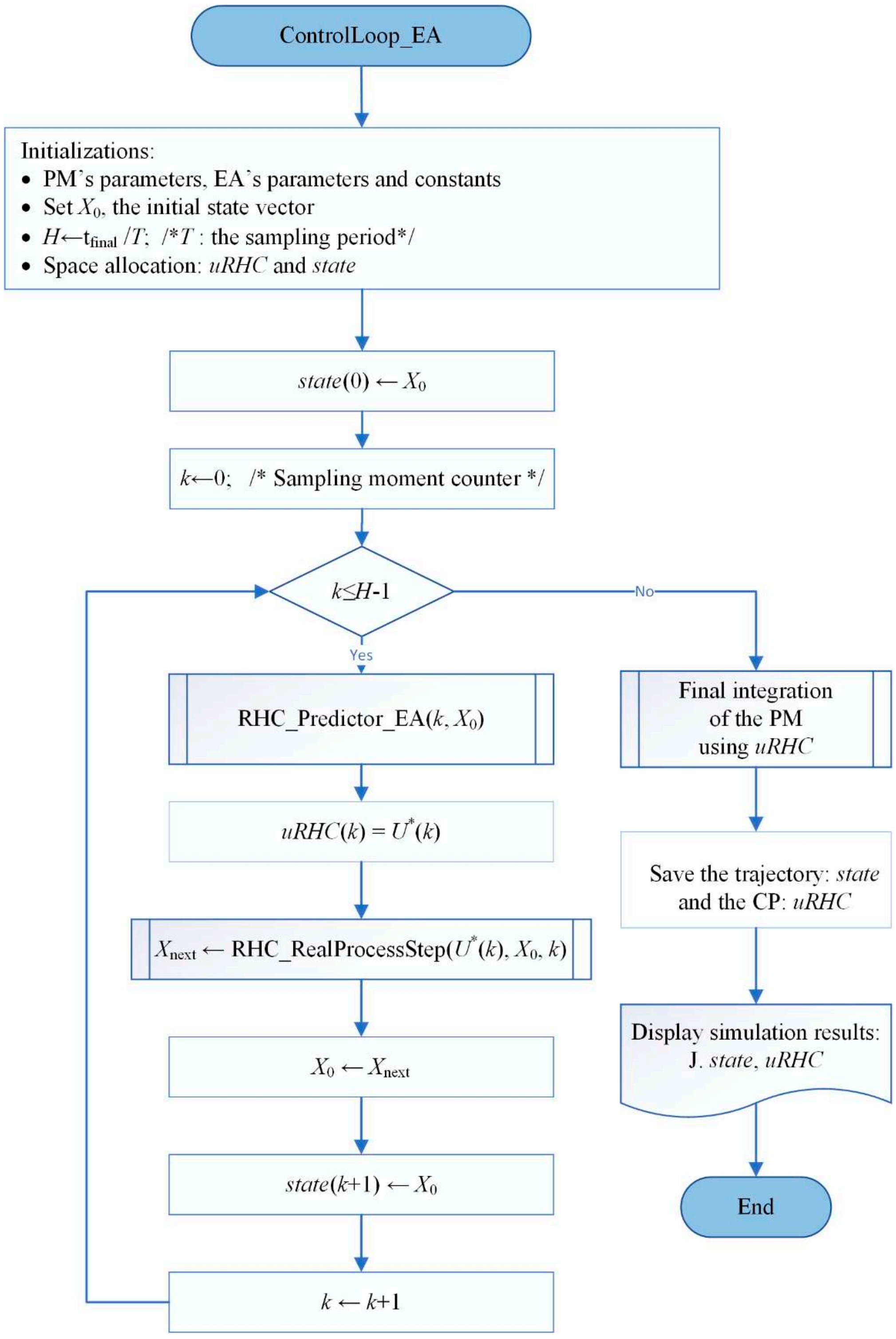

ControlLoop_EA
Initializations:
PM's parameters, EA's parameters and constants
Set X_0, the initial state vector
$H \leftarrow t_{final}/T$; /*T : the sampling period*/
Space allocation: $uRHC$ and $state$
$state(0) \leftarrow X_0$
$k \leftarrow 0$; /* Sampling moment counter */
$k \leq H-1$
No
Yes
RHC_Predictor_EA(k, X_0)
$uRHC(k) = U^*(k)$
$X_{next} \leftarrow$ RHC_RealProcessStep($U^*(k)$, X_0, k)
$X_0 \leftarrow X_{next}$
$state(k+1) \leftarrow X_0$
$k \leftarrow k+1$
Final integration of the PM using $uRHC$
Save the trajectory: $state$ and the CP: $uRHC$
Display simulation results: J, $state$, $uRHC$
End

Figure 4. The closed loop simulation to generate a quasi-optimal trajectory and its CP.

At every moment k, the controller calls the predictor based on the EA, `RHC_Predictor_EA`. The last one returns the optimal control value $U^*(k)$, which is used by the function `RHC_RealProcessStep` to determine the next Process's state.

The optimal control value and the optimal states are stored in the matrices *uRHC* (H x m) and *state* (H x n), respectively, having the structure presented in Figure 5.

state $(H \times n)$	*uRHC* $(H \times m)$
$(X(0))^T$	$(U^*(0))^T$
$(X^*(1))^T$	$(U^*(1))^T$
$\vdots$	$\vdots$
$(X^*(H-1))^T$	$(U^*(H-1))^T$

Figure 5. The matrices for the quasi-optimal trajectory and its CP.

Hence, the optimal CP and trajectory are described by the matrix *uRHC* and *state*, respectively, which are the images of $\Omega(X_0)$ and $\Gamma(X_0)$ sequences (see (12) and (13)).

In the case of the PRP, an example of matrices describing a quasi-optimal evolution is given in Figure 6. Notice that, this time, m=1 and the 16th state is the final one.

	state (16×5)					*uRHC*
k	x_1	x_2	x_3	x_4	x_5	$u^*(k)$
0	0	0	1.0000	5.0000	1.0000	0.1670
1	0.0000	0.0000	1.1567	4.9577	1.1670	0.0963
2	0.0000	0.0000	1.4411	3.3848	1.2633	0.4659
3	0.0000	0.0000	1.4203	5.1790	1.7292	0.6577
4	0.0000	0.0000	1.3877	6.6436	2.3869	0.2543
5	0.0000	0.0000	1.6920	4.7333	2.6412	0.7624
6	0.0000	0.0000	1.7723	4.7999	3.4035	1.4234
7	0.0000	0.0000	1.6862	6.0952	4.8270	1.0932
8	0.0000	0.0000	1.8552	5.1567	5.9201	1.2947
9	0.0000	0.0000	2.0543	3.9366	7.2148	2.0000
10	0.0000	0.0000	2.1671	3.3443	9.2148	0.3982
11	0.0332	0.1579	2.6170	0.0949	9.6130	0.9041
12	0.5415	0.8658	2.6204	0.1392	10.5170	0.7409
13	1.1599	1.5773	2.6326	0.0977	11.2580	0.8112
14	1.8118	2.2431	2.6386	0.1001	12.0690	1.1940
15	2.4045	2.7541	2.6401	0.1468	13.2630	-

Figure 6. An example of matrices for the optimal trajectory and its CP (PRP case).

3.2.2. Aggregation of datasets concerning M optimal evolutions of the closed loop

This subsection corresponds to step 2 of the design procedure. The controller's optimal behaviour learning process needs data from an important number of optimal evolutions. Practically, the program `ControlLoop_EA.m` will be executed repeatedly M times (e.g. M=200) in a simple loop described in the script `LOOP_M_ControlLoop_EA.m`. The objective is to create aggregate data structures and store the optimal trajectories and their CPs.

Figure 7 illustrates possible data structures, a cell array for M tables storing the trajectories and a matrix storing their CPs.

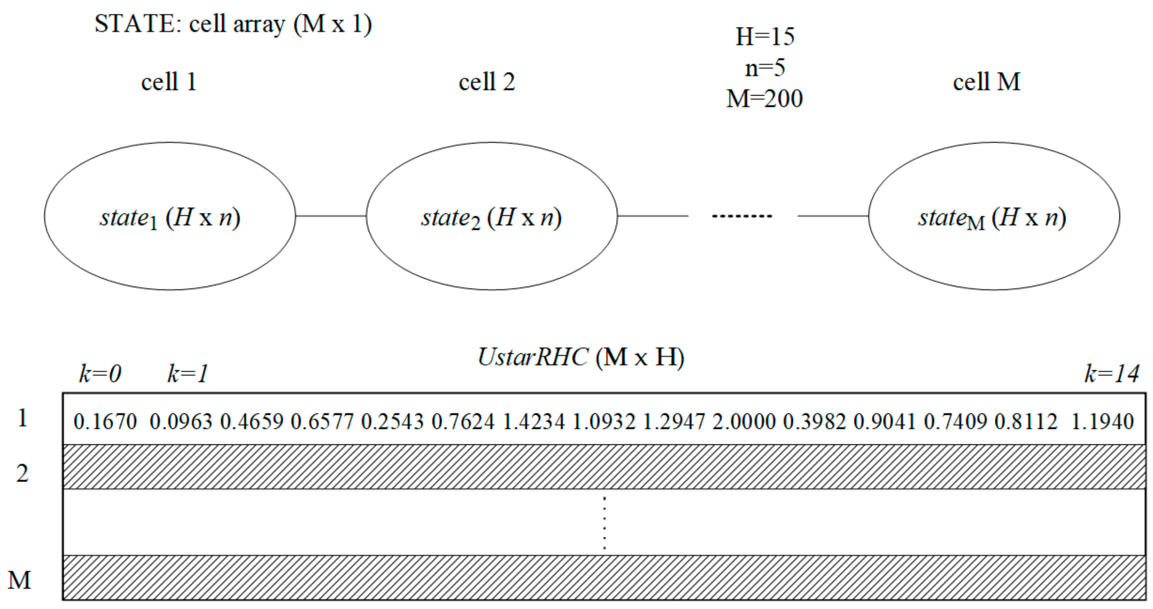


Figure 7. STATE and *UstarRHC*: data structures reprezenting the M optimal trajectories and CPs.

The optimal CPs could also be stored in a cell array, but in the PRP case (*m*=1), a matrix (Mx*H*) can store these values simpler. We call it *UstarRHC* (200 x 15); on each line, it memorizes the transposed vector *uRHC* from Figure 6.

The performance index values are also stored in a column vector *JMAT* (M x 1) for different analyses. All these data structures could be saved in a file for ulterior processing.

3.2.3. Extraction of datasets characterizing each sampling period

This subsection details step 3 of the design procedure. For each sampling period *k*, a dataset similar to Table 1 is extracted from *STATE* and *UstarRHC* data structures.

Considering *k*, $0 \le k \le H-1$, is already fixed, we generate a matrix $SOCSK \in \mathbf{R}^{M\times(n+m)}$ that gathers the **s**tates and **o**ptimal **c**ontrol values concerning step ***k*** from all the M experiences. Line *i*, $1 \le i \le M$, concatenates data from experience *i*:

$$SOCSK_i \leftarrow \left[\left(X_i^*(k)\right)^T \ \left(U_i^*(k)\right)^T \right].$$

Using the data structures defined before, it holds:

$$SOCSK_i \leftarrow \left[STATE_i(k,1:n) \ \ UstarRHC(i,k) \right] \quad .$$

STATEi designates the ith component of the cell array *STATE*. In the PRP case, we have:

$$SOCSK = \begin{bmatrix} x_1(k)^1 & x_2(k)^1 & x_3(k)^1 & x_4(k)^1 & x_5(k)^1 & u(k)^1 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ x_1(k)^i & x_2(k)^i & x_3(k)^i & x_4(k)^i & x_5(k)^i & u(k)^i \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ x_1(k)^M & x_2(k)^M & x_3(k)^M & x_4(k)^M & x_5(k)^M & u(k)^M \end{bmatrix} .$$

Remark 5. *Because the Controller's optimal behaviour has repercussions on each sampling period, the learning of its optimal behaviour will be split at the level of each interval [k, k+1).*

Figure 8 gives the flowchart of the data processing to obtain datasets for training and testing the machine learning algorithm at the level of each moment *k* (*datakTrain* and *datakTest*). To save these datasets for each *k*, we will use cell array *DATAKTrain* ($H\times 1$) and *DATAKTest* ($H\times 1$).

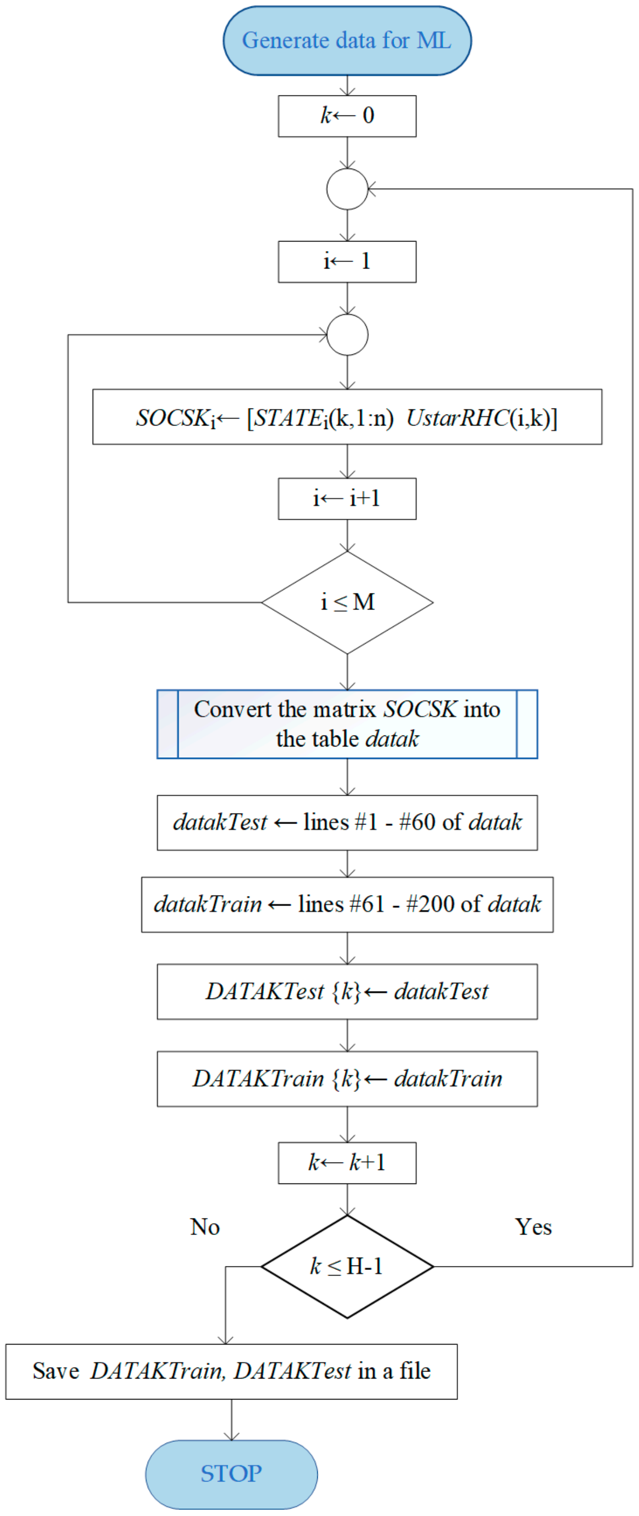

Generate data for ML
$k \leftarrow 0$
$i \leftarrow 1$
$SOCSK_i \leftarrow [STATE_i(k,1:n)\ \ UstarRHC(i,k)]$
$i \leftarrow i+1$
$i \le M$
Convert the matrix *SOCSK* into the table *datak*
datakTest ← lines #1 - #60 of *datak*
datakTrain ← lines #61 - #200 of *datak*
DATAKTest {*k*}← *datakTest*
DATAKTrain {*k*}← *datakTrain*
$k \leftarrow k+1$
No
Yes
$k \le H\text{-}1$
Save *DATAKTrain, DATAKTest* in a file
STOP

Figure 8. Preparing the training and testing datasets for machine learning at the level of each sampling period.

After constructing the matrix *SOCSK*, we shall convert it into a table, which seems more convenient for processing datasets for machine learning in some programming and simulation systems. The result is the table named *datak*, which has variables and properties. After that, this table's lines are split into table *datakTrain* with 140 examples for training (70%) and table *datakTest* with 60 data points for testing (30%). Finally, these tables are stored in cell #*k* of the *DATAKTrain* and *DATAKTest* array, respectively. They will be used by the machine learning algorithm ulteriorly.

3.3. Construction of Machine Learning Models

This section covers step #4 of the design procedure. As stated by Remark 3, the set of functions $\Phi = \{f_k \,|k = 0,\ 1,\ldots,\ H-1\}$ is the machine-learning model for the optimal behaviour of the couple (PM – EA). The data points are couples (Process state – optimal control values) related to moment *k*, which the function f_k can learn.

Of course, another kind of machine-learning model treating globally the learning process (PM – EA's control profile) could be addressed, without splitting the learning at the level of sampling periods. The resulting model would be more complex and difficult to train and integrate into the controller.

In this work, we mainly chose multiple linear regression as a machine learning algorithm because of its simplicity. This characteristic is important, especially when *H* is large. Secondly, the linear regression functions f_k for each sampling period are appropriate for a controller implementation; these functions directly give the control law (see equation (14)). To emphasize this aspect, Table 2 presents the general structure of the ML_controller when using the set of functions Φ.

Table 2. The structure of the controller's algorithm using linear regression functions.

	Controller's algorithm (ML_controller)
1	Get the current value of the state vector, $X(k)$; /* Initialize k and $X(k)$ */
2	$U^*(k) \leftarrow f_k(X(k))$ /* see equation (14) */
4	Send $U^*(k)$ towards the Process.
5	Wait for the next sampling period.

We could determine the set of functions Φ through multiple linear regression considering different models containing an intercept, linear terms for each feature (predictor variable), products of pairs of distinct features (interactions), squared terms, etc. In other words, the resulting functions could be nonlinear as functions of Process states.

In our case study, we will also apply the strategy of *stepwise regression* that adds or removes features starting from a constant model.

3.3.1. Models with linear terms for each state variable

Remark 6. *Our objective is not to find the "best" set of linear regression models but to prove that our approach is working and this new model can replace the EA.*

That is why, for the beginning, we adopt a very simple model such that each regression function f_k is a simple linear model involving only linear terms for each state variable and an intercept. Each model is trained and tested separately, considering the datasets already prepared in the cell arrays *DATAKTrain* and *DATAKTest*.

The construction of these models is presented by the pseudocode in Table 3. The datasets for training and testing, described in section 3.2.3, are now input data for this algorithm. The coefficients of each function f_k, having the form

$$f_k(X(k)) = C^k{}_0 + C^k{}_1 \cdot x_1(k) + C^k{}_2 \cdot x_2(k) + \ldots + C^k{}_n \cdot x_n(k)\,, \tag{15}$$

are stored in an output matrix called COEFF. These models are objects stored in an output cell array called MODEL ($H \times 1$). Line #4 creates the model "mdl" using the function *fitting_to_data* that fits the function (15) to the dataset "datakTrain". At line #6, the function *get_the_coefficients* extracts the six coefficients, which are put in a line of matrix COEFF afterwards.

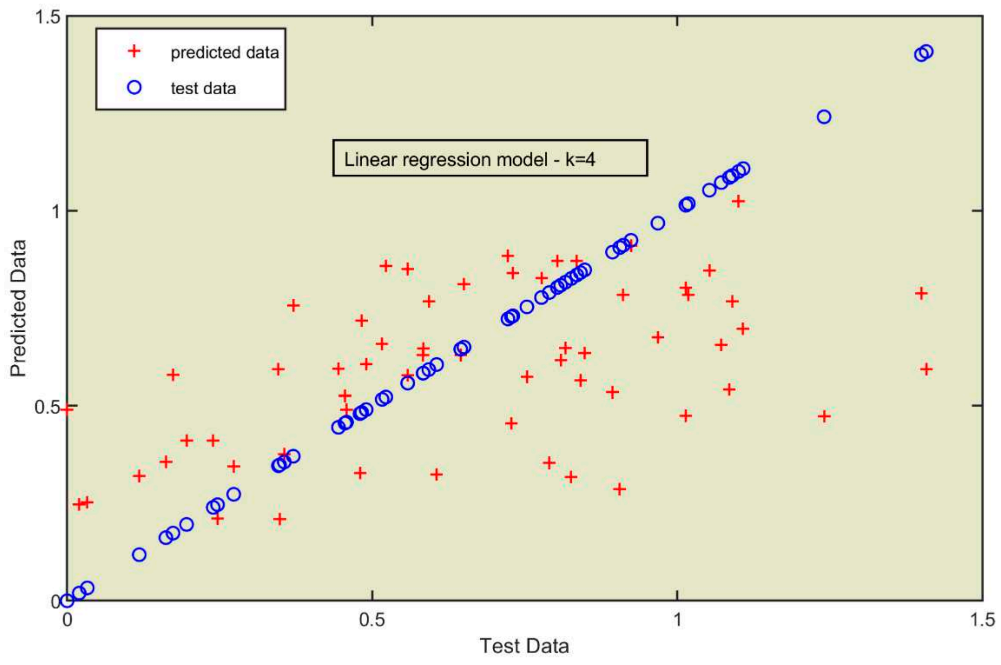
The predicted values corresponding to *datakTest* are stored in the vector "uPred" by the function *fpredict* to be compared with the experienced values. Details concerning the implementation of *fitting_to_data*, *get_the_coefficients*, and *predict* functions are given in Appendix B.

Table 3. The pseudocode of the models' construction.

```
    /* This pseudocode describes the training and testing of the linear models set $\Phi$ */
    Input: cell arrays DATAKTrain, DATAKTest
    Output: matrix COEFF ( $H \times (n+1)$ ), cell array MODEL{ $H \times 1$ } storing objects that are the
    linear models $f_k$
1   for $k = 0 \ldots H-1$.
2           datakTrain ← DATAKTrain{k};
    /* Recover the dataset from the cell array if it was saved in a file */
3           datakTest ← DATAKTest{k};
    /* Recover the dataset from the cell array if it was saved in a file */
4           mdl ← fitting_to_data(datakTrain);
    /* Create the linear regression model that fits datakTrain */
5         #display mdl;
6         coef(:) ← get_the_coefficients(mdl)
7           COEFF(k,:)  ← coef(:);
    /* save the coefficients in the corresponding line of matrix COEFF*/
8           MODEL(k,1) ← mdl;
9           uPred  ← fpredict(mdl, datakTest)
    /* The predicted control values are stored in the vector uPred */
10  # Represent in the same drawing the values of uPred and datakTask's last column for
    comparison.
11  end.
```

For the PRP, e.g., the coefficients are given in Table B2 (Appendix B).

A comparison of the values in "datakTest" to the predicted values "uPred" is given in Figure 9 [29,31]. The blue line is the plot of test values against themselves.



Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 December 2023 doi:10.20944/preprints202312.0876.v1

Figure 9. Predicted data and Test data of the regression model for k=4.

The 60 predicted values are disposed along the blue line at a distance inferior to 0.2 for most of them.

3.3.2. Models constructed through stepwise regression

This section will propose a more elaborate model that allows the possibility to include nonlinear terms as the interactions, that is, the product of predictor variables (ex.: x_1*x_4).

As an example, for the PRP case, we will apply the strategy of stepwise regression that adds or removes features starting from a constant model. This strategy is usually implemented by a function `stepwise`(T) returning a model that fits the dataset in Table T.

model ← `stepwise`(T)

The script `GENERATE_ModelSW.m` constructs de set of models using this function (see Appendix C). This script yields the regression functions given in Table 4.

Table 4. The regression functions following the stepwise strategy.

k	$f_k(X)$	k	$f_k(X)$
0	f_k=-10.449+ 10.606·x_2	8	f_k=4.3452-0.44551 x_5
1	f_k=0.72997 -0.09299·x_4	9	f_k=36.628 -8.1574·x_2 -2.5122 ·x_5
2	f_k=-14.476+3.0317·x_3+1.3432·x_5+4.9049·x_3·x_5	10	f_k=4.2061+713.02·x_2-0.41558·x_5-95.461·x_2·x_5
3	f_k=-1.5603+ 1.4141·x_2	11	f_k=0.87113 -0.60908·x_1
4	f_k=-1.7596+1.552·x_2	12	f_k=1.0149 -0.30622·x_1
5	f_k=-2.351+1.9328·x_2	13	f_k=1.2692 -0.33849·x_1
6	f_k=4.4274 -0.9024·x_5	14	f_k=-0.19244 -0.41509·x_1+ 0.41057·x_2 +0.10083·x_5
7	f_k=4.8791-0.72412·x_5		

A fragment of the listing generated by the script mentioned above is given in Table 5.

Table 5. Construction of the regression model by `stepwise` function for k=2.

```
1. Adding x3, FStat = 232.429, pValue = 3.152347e-35
2. Adding x5, FStat = 4.1002, pValue = 0.044229
3. Adding x3:x5, FStat = 4.5157, pValue = 0.034837

Linear regression model:
    u ~ 1 + x3*x5

Estimated Coefficients:
                   Estimate       SE        tStat       pValue   
                   ________    ________    ________    __________

    (Intercept)    -14.476       4.5132    -3.2074      0.0015644
    x3              3.0317      0.87367     3.4701     0.00063985
    x5              1.3432      0.75286     1.7841       0.075955
    x3:x5           4.9049       2.3082      2.125       0.034837


Number of observations: 200, Error degrees of freedom: 196
Root Mean Squared Error: 0.137
R-squared: 0.56,  Adjusted R-Squared: 0.553
F-statistic vs. constant model: 83, p-value = 1.07e-34
```

We can see how the function `stepwise` works and what are the statistic parameters that validate the model.

4. Simulation of the control-loop system equipped with the ML_controller

To evaluate the measure in which the set of functions Φ succeeded in "learning" how the EA acts as an optimal predictor, we shall refer to the simulation results of the control loop. The script `ControlLoop_ML.m` implements step #6 of the design procedure and is described by the pseudocode in Table 6 .

Table 6. The pseudocode of the control loop simulation using linear regression functions.

```
ControlLoop_ML(MODEL, X0)
    /* This pseudocode describes the simulation of the closed loop that uses the proposed
    controller (the linear models set $\Phi$) */
    Input: MODEL ($H\times 1$) storing objects that are the linear models $f_k$ ,
                 X0: the initial Process's state.
    Output: The vector uML (1, H) representing the quasi-optimal CP,
          the matrix State (H+1,n) reprezenting the quasi-optimal trajectory.
1   # Initializations: technological bounds umin, umax;
2   # The matrix State will store $X(k)$, k=0,…,H-1. Initially, it is put to zero.
3   State (1, :) ← X0
4   for k = 0…H-1.
5         mdl ← MODEL{k};   /* mdl is a linear regression model */
6         uML(k) ← feval(mdl, X0(1), X0(2), ..., X0(n));
7         # Limit to umin or umax the value of uML(k).
          X0 ← step_PP_RH(uML(k), X0)
8         /* Determine the new state the Process evolves when the control value uML(k) is
    applied. */;
9         State(k+1,:) ← X0.
10  end.
11  return uML and State
```

In line #6, the function `feval` returns the value uML(*k*) that the model mdl predicts when the current state is X0 (also a local variable). Instead of using `feval`, one can use the product coefficients - state variable.

The function *step_PP_RH*(uML*(k),* X0*)* calculates the next state (at the moment *k*+1) by integration of state equations (7). The codes for all the proposed functions are given in the folder `ART_Math`.

In the PRP case, we used `ControlLoop_ML.m` to simulate the control loop using the models constructed in section 3.3.1. The ML_controller yielded the CP drawn in Figure 10. This one engenders the quasi-optimal state evolution depicted in Figure 11b, which can be compared to the typical evolution, Figure 11a, generated by the RHC endowed with an EA. Figure 11a was produced within the authors' previous work [21].

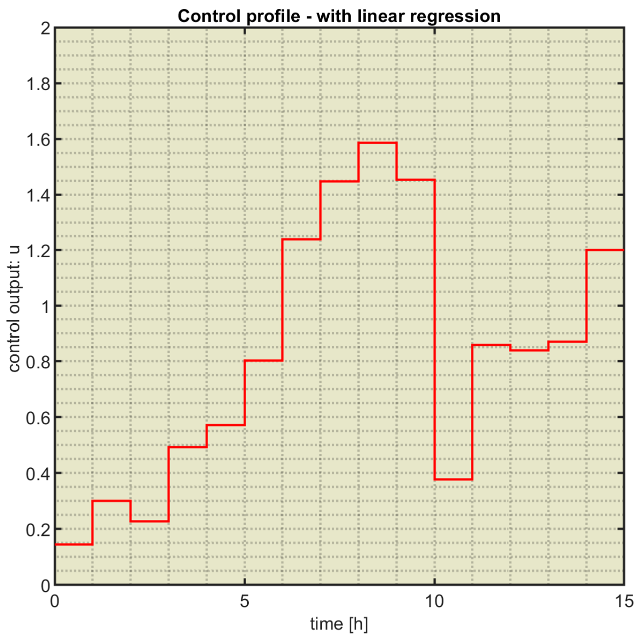


Figure 10. The CP achieved by the ML_controller, linear regression version.

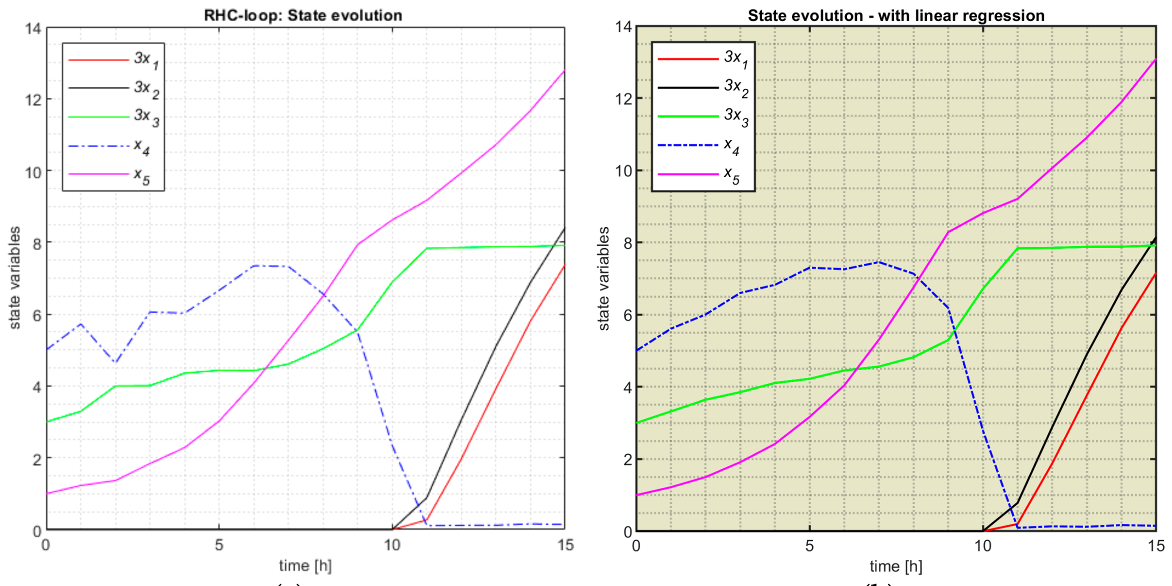


Figure 11. PRP: State evolution comparison **(a)** Prediction with EA using the RHC **(b)**Prediction achieved by machine learning using a set of linear regression functions.

The resemblance between the two process's responses is very high and proves that *the set of functions* f_k *succeeded in emulating the optimal behaviour of the EA*. Moreover, the performance index's value achieved by the CP in Figure 11b is very good $J_0 = 32.0986$ because it equals the maximum value (J_0=32.0936) recorded in the M data points generated by the EA. Hence, the PRP's solution found by the ML model is also quasi-optimal.

The controller's execution time

Because the initial reason for this research is just the EA's execution time decrease, we compared the execution time of the two controllers, the EA and ML controllers. Actually, we compared the simulation times for the two closed loops with the EA and ML models because we already have the programs `ControlLoop_EA.m` and `ControlLoop_ML.m`. The two times are 38 seconds and 0.08

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 December 2023 doi:10.20944/preprints202312.0876.v1

seconds, respectively (see Appendix D). The average execution times of the controller are $38/H$ and $0.08/H$ seconds. H equals 15 in the PRP case.

Remark 7. *The decrease in the controller's execution time is outstanding, all the more so because the ML_controller keeps the evolution accuracy. In anterior research, we have obtained a small decrease but with an acceptable accuracy decrease as well.*

How can we explain this outstanding decrease in the controller's execution time? The key is how the EA predictor works. The EA algorithm generates a population of solutions and then evaluates the cost function by numerical integration of the PM for each solution and along many generations. The PM's numerical integration for thousands of times is the most time-consuming. Nothing of these happens when the ML model works, only a single evaluation of a simple regression function. The EA's computational complexity is huge compared to a few calculations.

The model developed in section 3.3.2 Φ_2 (considering Φ_1 the model constructed in section 3.3.1) leads us to a new ML controller. This time, we used the script `ControlLoop_MLSW.m` (see Appendix C) to simulate the closed-loop system. Figure 12 plots the resulting CP in a blue line. The CP in Figure 10 is also plotted in red to facilitate the comparison.

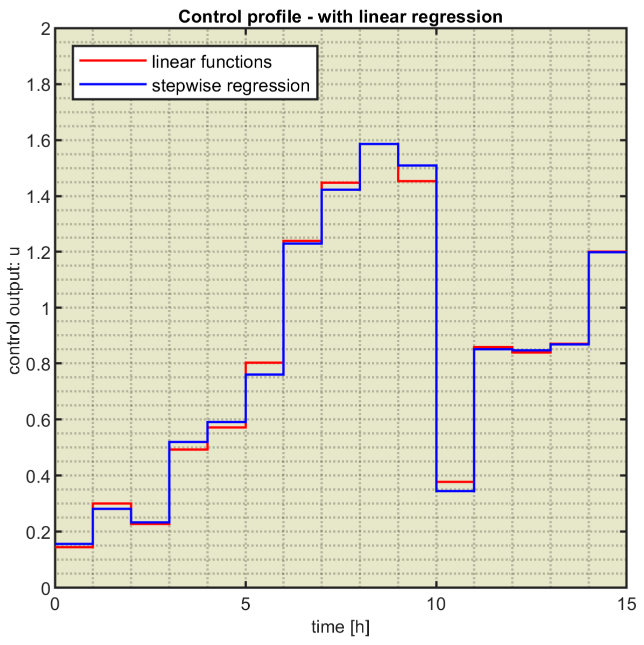


Figure 12. The CP achieved by the ML_controller, stepwise regression version.

The two CPs are practically identical, which will engender the similarity of the state evolutions. Figure 13 confirms that situation and plots the state evolutions involved by the two controllers.

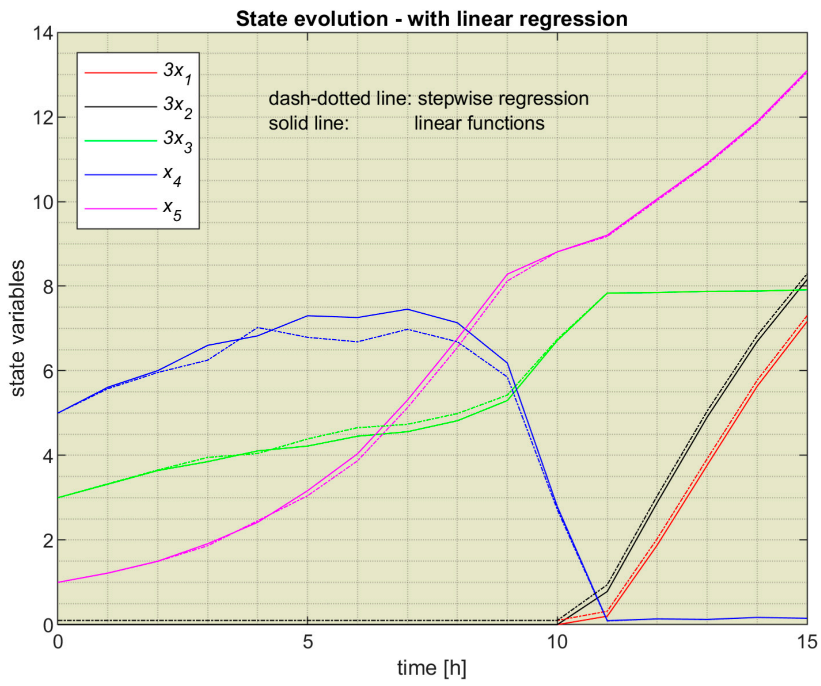


Figure 13. Comparison between the state evolutions involved by the two ML controllers.

This remarkable similarity is because the models Φ_1 and Φ_2 emulate the same couple (EA-PM), and both do it very well. Consequently, the performance index is practically the same ($J_0 = 32.0986$). Although the results are identical, the sets of regression functions are different. The functions set Φ_2 seems more appropriate for the controller's implementation because of their simpler formulas.

In real-time, the difference between the PM's state and the state of the process is the biggest problem. We can consider that the Process is affected by a noise representing this difference. When the noise is important, the control loop could loose totally its functionality. Our desideratum is that the controller keeps acceptable performances for a significative noise range. Inside this range, the controller must reject this difference as much as possible, involve a quasi-optimal process evolution, and produce a performance index near J_0 (see Appendix E).

5. Discussion

This paper answers positively to the issue raised in section 1: whether an ML algorithm could "learn" the optimal behaviour of the predictor based on an EA. The proof was made in the context of OCPs, giving rise to some findings:

1. The issue statement oneself: to find an ML model experiencing the datasets generated by the couple (EA -PM), trying to capture its optimal behaviour. An EA is a product of computational intelligence. This link between two "intelligent" entities is interesting; further developments can be derived from this "equivalence". The same issue can be considered when instead of EA is another metaheuristic.
2. The dataset's construction as a dynamic trace of the EA predictor by aggregating the trajectories and CPs. The number M is a procedure's parameter established according to the process complexity. The dynamic trace must include couples (state, optimal control value) spread throughout the evolution space. In this way, the ML model could generalize well.
3. The dataset extraction for each sampling period, which is a premise to find an ML model for each *k*.
4. The outstanding decrease in the ML_controller's execution time.
5. The design procedure for the ML_controller and all associated algorithms (simulation and models' construction algorithms).

Finding #3 is related to the fact that we are looking for an ML model comprising a set of regression functions Φ. We underline the motivation: its simplicity, especially when the control horizon has many sampling periods. If need be, obtaining the model during the sampling period is conceivable. On the other hand, a regression function for each sampling period means that the control law is directly implemented. The controller implementation is now straightforward.

The degree to which the ML_controller accurately reproduces the couple (EA, PM)'s behaviour outclasses another achievement of the controller: its execution time. Because the initial reason for this research is just the EA's execution time decrease, this subject deserves more attention. The simulation time for the closed loop using the EA predictor for the PRP case is 38 seconds, while the same simulation using the ML_controller is 0.08 seconds. This outstanding achievement is due to three factors:

- the ML model is split at the level of each sampling period; a single regression function f_k is the current model;
- a regression function has a very simple expression, which is, in fact, just the control law.
- the PM's numerical integration is totally avoided.

According to the authors' experience addressing this subject, the execution time decrease to a small extent is done by paying the price of predictions' accuracy decrease as well. In the case of the ML_controller, remarkably, that does not happen; the accuracy is kept.

As we mentioned before, predictors based on metaheuristics are usually used for slow process control. Owing to its small execution time, the ML_controller extends largely the set of processes, which can be controlled using EA or another metaheuristic. We must implement the EA predictor in the first phase of the controller's design, which will produce the datasets for the ML algorithm's training. Finally, the ML predictor will be used as a faster equivalent of the EA predictor in the control structure.

Special attention was addressed to the implementation aspects related to the PRP case. All algorithms used in this work are implemented, the associated scripts are attached as supplementary materials, and all the necessary details are given in the Appendixes. Readers can find support to apprehend and eventually reproduce parts of this work.

Finally, the ML models and the controller must fulfil their task in real conditions, namely when the control system works with a real Process. Although the design procedure does not depend on the real Process, an interested reader can ask how to forecast the closed-loop behaviour when the real Process and the PM differ. At the end of section #4 and in Appendix D, some elements can help to simulate this situation.

To simplify the presentation, we applied the proposed methods only in the PRP case. The methods were also applied in other case studies not presented in this paper, with the same favourable conclusion; the proposed ML models succeeded in apprehending the optimal behaviour of the couple (EA, PM) and yielded efficient controllers.

Obviously, another ML model treating globally the learning process is conceivable, without splitting the learning at the level of sampling periods. The resulting model would be more complex and difficult to train and integrate into the controller, but it could be useful in other applications. In a further research project we will make investigations in this direction.

Author Contributions: Conceptualization, V.M.; methodology, I.A. and V.M.; software, V.M.; validation, V.M. and I.A.; formal analysis, V.M.; investigation, I.A.; resources, I.A.; data curation, I.A.; writing—original draft preparation, V.M.; writing—review and editing, V.M. and I.A.; visualization, I.A.; supervision, I.A.; project administration, I.A.; funding acquisition, I.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: No new data were created.

Acknowledgments: This work benefited from the administrative support of the Doctoral School of "Dunarea de Jos" University of Galati, Romania.

doi:10.20944/preprints202312.0876.v1

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

The prediction function using an EA

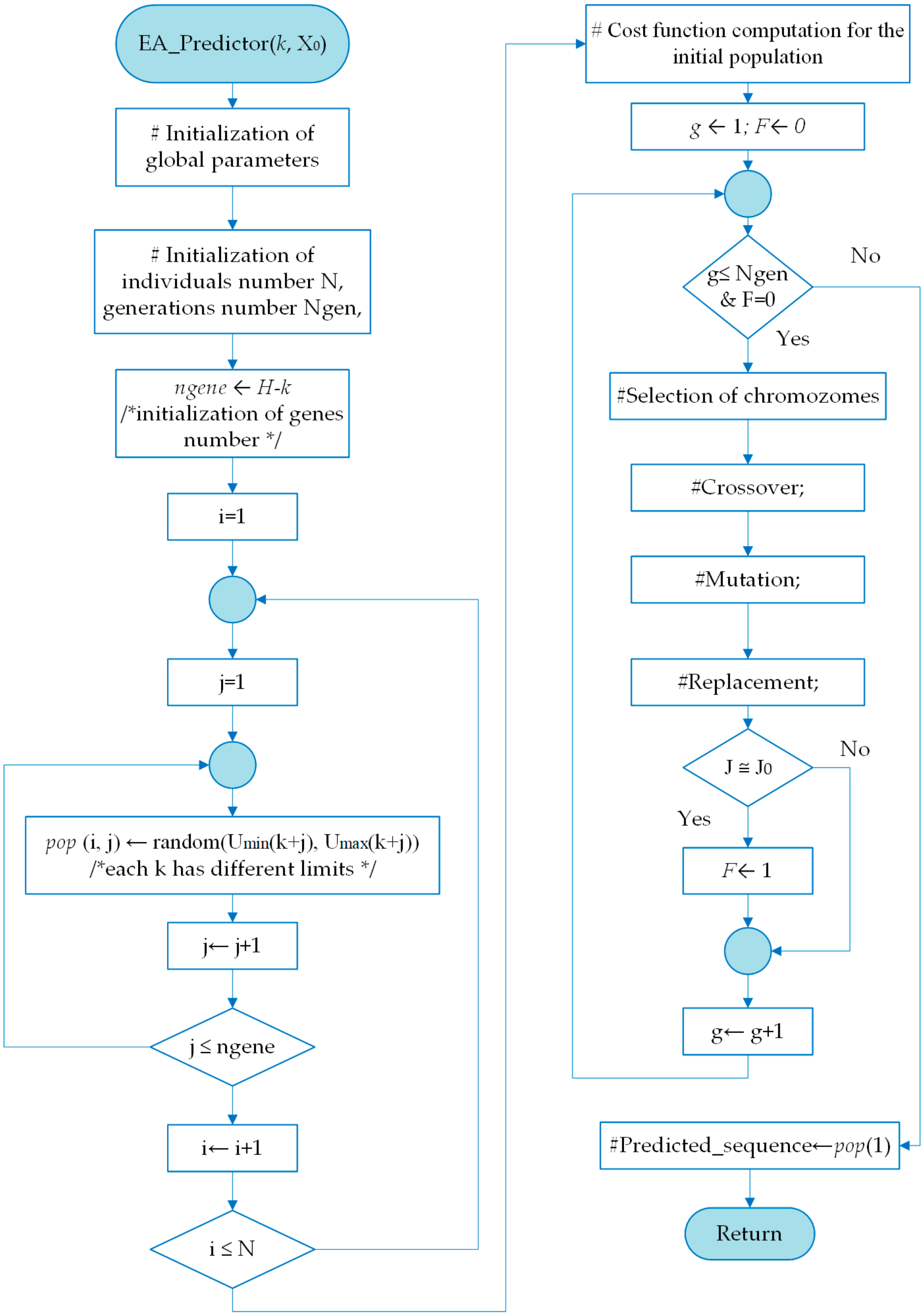

EA_Predictor(k, X0)
# Initialization of global parameters
# Initialization of individuals number N, generations number Ngen,
ngene ← H-k /*initialization of genes number */
i=1
j=1
pop (i, j) ⟵ random(Umin(k+j), Umax(k+j)) /*each k has different limits */
j⟵ j+1
j ≤ ngene
i⟵ i+1
i ≤ N
# Cost function computation for the initial population
g ← 1; F← 0
g≤ Ngen & F=0
No
Yes
#Selection of chromozomes
#Crossover;
#Mutation;
#Replacement;
J ≅ J0
No
Yes
F← 1
g⟵ g+1
#Predicted_sequence⟵pop(1)
Return

Figure A1. The flowchart of the prediction function using an EA (`RHC_Predictor_EA.m`).

The cost function values of the offspring are calculated inside the EA's operators. Variable F is used to stop the iterative process toward the optimal solution. When the performance index is close to J_0, the EA stops, considering that a very good solution is found, and the best solution becomes the predicted control sequence.

Appendix B

The models' construction script

Our implementation is based on the MATLAB system, in which "`fitlm`", "`model.Coefficients`", and "`predict`" correspond to *fitting_to_data*, *get_the_coefficients*, and *fpredict* (the functions proposed in section 3.3.1), respectively. The algorithms in Figure 8 and Table 2 are joined and implemented by the script `Model_Construction.m`. The coefficients found by the script `Model_Construction.m` are listed in the table below.

Table B1. The coefficients of the linear regression *fk*.

k	C0	C1	C2	C3	C4	C5
0	-38.079	0	0	14.128	6.0244	-6.0264
1	5.6291	-1.3608e+07	2.4367e+06	-3.7392	-0.47639	1.139
2	-38.33	3.0713e+05	-2.2759e+05	21.679	1.5663	1.8422
3	-28.268	-40144	4487.7	14.638	1.1171	1.2135
4	-29.163	1.6973e+05	-1.6184e+05	14.311	1.2206	0.63983
5	-88.374	95097	-1.0768e+05	40.145	4.0721	0.78088
6	-25.715	3.5858e+05	-3.7572e+05	13.047	1.5373	-1.0067
7	-62.516	-4.657e+05	4.4453e+05	29.336	4.0186	-2.0782
8	45.254	-1.387e+07	1.3915e+07	-13.445	-0.83017	-2.415
9	504.78	-2.211e+06	2.2891e+06	-192.32	-23.957	-1.8747
10	-236.04	-82.588	11.879	95.084	12.898	-1.3842
11	-147.92	0.58548	0.29874	57.825	7.5075	-0.34759
12	93.46	-0.37731	-0.76193	-35.138	-4.7889	0.094402
13	34.849	-1.0111	0.66035	-13.353	-0.79003	0.1234
14	4.0848	-0.30413	0.29959	-1.6444	-0.43335	0.11348

Appendix C

Implementation of the stepwise strategy

The stepwise strategy is implemented through the function "`stepwiselm`", which adds or removes terms to the current model. The set of available terms is addressed, and the one having an F-test for adding it with a p-value of 0.05 or less is added to the model. If no terms can be added, the function evaluates the terms in the model and removes that one having an F-test for removing it with a p-value of 0.10 or greater. This process stops when no terms can be added or removed.

The models from section 3.3.2 are generated using this function for each k in the script `GENERATE_ModelSW.m`. The latter has the same structure as `Model_Construction.m`, except the function `stepwiselm` is used instead of `fitlm`.

To simulate the closed loop, we used the script `ControlLoop_MLSW.m`, which is very similar to `ControlLoop_ML.m` but uses a different `mat-file`. Finally, it calls the script DRAW_ML to plot the CP and the state evolution.

Appendix D

Execution time

Because the ControlLoop_ML is conceived as a function, the closed loop simulation is made by the script `SIM1.m`, which calls it after loading the ML model. The simulation processor is Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz.

Finally, the comparison is made between the execution time of two programs, `ControlLoop_EA.m` and `SIM1.m`. The results are 38 seconds and 0.8 seconds, respectively.

Appendix E

The ML_controller's capacity to reject noises affecting the Process state

The general problem of noise modelling depends on the Process specificity and overpass the aim of this work. Our immediate goal is to provide a simple technique to test the existence of a noise range where the controller keeps acceptable performances.

In our simulation, we considered the noise is added to the PM's state variables. For our example, the noise I n_i, equivalent to all influences, is added to the PM's vector state:

$$x_i(k) \leftarrow x_i(k) + n_i\ ,\ i=1,\ldots,5$$

We adopted the hypothesis the noise is a random variable RT in the interval [-L_i, L_i], where

$$L_i = p \cdot |x_i(k)|,\ 0<p<1$$

Hence, the noise depends on the state variable's absolute value to avoid annulling its influence. We have chosen a constant value of 4%, which means an 8% interval for placing the noise. The controller keeps the regression models from Table 5.

These elements have been inserted into the control loop simulation script (`ControlLoop_ML_noise.m`), which has been carried out many times. Table 6 presents only three CPs and the associated performance index to evaluate whether the control loop has a margin of robustness to noise.

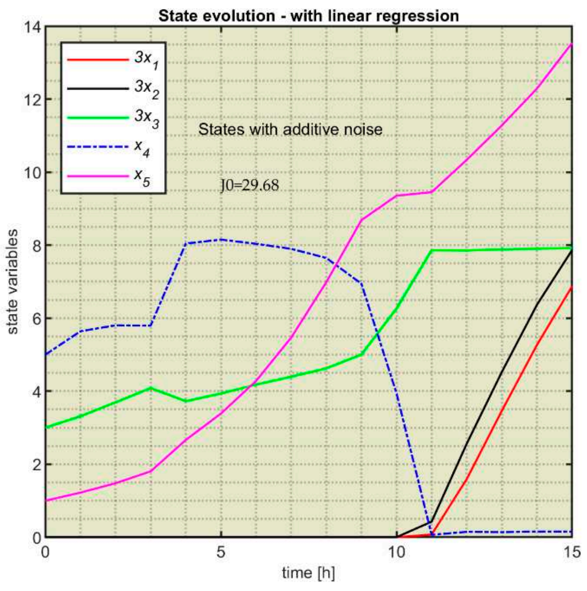
Table E1. Three simulations of the control loop with noise.

Control Profile	*J*
uML1=[0.1570 0.2926 0.0000 0.8799 0.5285 0.8301 1.1906 1.4888 1.6225 2.0000 0.0894 0.8711 0.9350 0.9830 1.2488]	29.6884
uML2=[0.1570 0.2842 0.3744 0.3296 0.6528 0.7238 1.2591 1.4624 1.5555 2.0000 0.1898 0.8700 0.9106 0.9412 1.1815]	28.4537
uML3=[0.1570 0.2831 0.0873 0.7395 0.5383 0.7105 1.2914 1.4879 1.6041 1.9905 0.0962 0.8693 0.8938 0.9029 1.1856]	31.8093

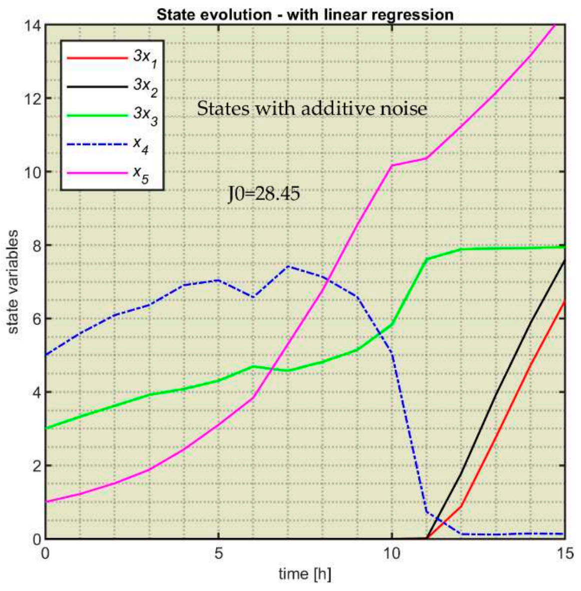
The state evolution for the two first simulations is presented in Figure E1.

After simulations, some conclusions can be drawn:

- The controller succeeded in keeping the stability, but the state evolution changed the look compared to Figure 11b.
- The value J is generally smaller than J_0 and has different values for different simulations because the noise has random values.
- Technological aspects must be considered when deciding whether or not the controller works acceptably.



(a)



(b)

Figure E1. The state evolution for two simulations with additive noise (**a**) CP=uML1 (**b**) CP=uML2.

References

1. Siarry, P. Metaheuristics; Springer: Berlin/Heidelberg, Germany, 2016; ISBN 978-3-319-45403-0
2. Talbi, E.G. Metaheuristics—From Design to Implementation; Wiley: Hoboken, NJ, USA, 2009; ISBN 978-0-470-27858-1.
3. Kruse, R.; Borgelt, C.; Braune, C.; Mostaghim, S.; Steinbrecher, M. Computational Intelligence—A Methodological Introduction, 2nd ed.; Springer: Berlin/Heidelberg, Germany, 2016; https://doi.org/10.1007/978-1-7296-3.
4. Faber, R.; Jockenhövelb, T.; Tsatsaronis, G. Dynamic optimization with simulated annealing. Comput. Chem. Eng. 2005, 29, 273–290.
5. Onwubolu, G.; Babu, B.V. New Optimization Techniques in Engineering; Springer: Berlin/Heidelberg, Germany, 2004.
6. Valadi, J.; Siarry, P. Applications of Metaheuristics in Process Engineering; Springer International Publishing: Berlin/Heidelberg, Germany, 2014; pp. 1–39. https://doi.org/10.1007/978-3-319-06508-3
7. Minzu, V.; Riahi, S.; Rusu, E. Optimal control of an ultraviolet water disinfection system. Appl. Sci. 2021, 11, 2638, https://doi.org/10.3390/app11062638
8. Minzu, V.; Ifrim, G.; Arama, I. Control of Microalgae Growth in Artificially Lighted Photobioreactors Using Metaheuristic-Based Predictions. Sensors 2021, 21, 8065, https://doi.org/10.3390/s21238065
9. Abraham, A.; Jain, L.; Goldberg, R. Evolutionary Multiobjective Optimization—Theoretical Advances and Applications; Springer: Berlin/Heidelberg, Germany, 2005; ISBN 1-85233-787-7.
10. Hu, X.B.; Chen, W.H. Genetic algorithm based on receding horizon control for arrival sequencing and scheduling. Eng. Appl. Artif. Intell. 2005, 18, 633–642.
11. Hu, X.B.; Chen, W.H. Genetic algorithm based on receding horizon control for real-time implementations in dynamic environments. Proceedings of the 16th Triennial World Congress, Prague, Czech Republic, 4–8 July 2005; Elsevier IFAC Publications: Amsterdam, The Netherlands, 2005.
12. Mayne, D.Q.; Michalska, H. Receding Horizon Control of Nonlinear Systems. IEEE Trans. Autom. Control. 1990, 35, 814–824.
13. Mînzu, V.; Rusu, E.; Arama, I. Execution Time Decrease for Controllers Based on Adaptive Particle Swarm Optimization. Inventions 2023, 8, 9. https://doi.org/10.3390/inventions8010009
14. Goggos, V.; King, R.; Evolutionary predictive control. Comput. Chem. Eng. 1996, 20 (Suppl. 2), S817–S822.
15. Chiang, P.-K.; Willems, P. Combine Evolutionary Optimization with Model Predictive Control in Real-time Flood Control of a River System. Water Resour. Manag. 2015, 29, 2527–2542.
16. Minzu, V.; Serbencu, A. Systematic procedure for optimal controller implementation using metaheuristic algorithms. Intell. Autom. Soft Comput. 2020, 26, 663–677, https://doi.org/10.32604/iasc.2020.010101
17. Minzu, V. Quasi-optimal character of metaheuristic-based algorithms used in closed-loop — Evaluation through simulation series. Proceedings of ISEEE, Galati, Romania, 18 – 20 October 2019.

18. Minzu, V. Optimal Control Implementation with Terminal Penalty Using Metaheuristic Algorithms. Automation 2020, 1, 48–65, https://doi.org/10.3390/automation1010004
19. Minzu, V.; Riahi, S.; Rusu, E. Implementation aspects regarding closed-loop control systems using evolutionary algorithms. Inventions 2021, 6, 53, https://doi.org/10.3390/inventions6030053
20. Minzu, V.; Arama, I.; Optimal Control Systems Using Evolutionary Algorithm-Control Input Range Estimation, Automation 2022, 3(1), 95–115, https://doi.org/10.3390/automation3010005
21. Minzu, V.; Georgescu, L.; Rusu, E. Predictions Based on Evolutionary Algorithms Using Predefined Control Profiles. Electronics 2022, 11, 1682. https://doi.org/10.3390/electronics11111682.
22. Goodfellow, I.; Bengio, Y.; Courville, A., Machine Learning Basics. *In Deep Learning*; The MIT Press, USA, 2016; pp. 95–161; ISBN 978-0262035613
23. Zou, S.; Chu, C.; Shen, N.; Ren, J. Healthcare Cost Prediction Based on Hybrid Machine Learning Algorithms. Mathematics 2023, 11, 4778. https://doi.org/10.3390/math11234778
24. Cuadrado, D.; Valls, A.; Riaño, D. Predicting Intensive Care Unit Patients' Discharge Date with a Hybrid Machine Learning Model That Combines Length of Stay and Days to Discharge. Mathematics 2023, 11, 4773. https://doi.org/10.3390/math11234773
25. Albahli, S.; Irtaza, A.; Nazir, T.; Mehmood, A.; Alkhalifah, A.; Albattah, W. A Machine Learning Method for Prediction of Stock Market Using Real-Time Twitter Data. Electronics 2022, 11, 3414. https://doi.org/10.3390/electronics11203414
26. Wilson, C.; Marchetti, F.; Di Carlo, M.; Riccardi, A.; Minisci, E. Classifying Intelligence in Machines: A Taxonomy of Intelligent Control. Robotics 2020, 9, 64. https://doi.org/10.3390/robotics9030064
27. Banga, J.R.; Balsa-Canto, E.; Moles, C.G.; Alonso, A. Dynamic optimization of bioprocesses: Efficient and robust numerical strategies. J. Biotechnol. 2005, 117, 407–419.
28. Balsa-Canto, E.; Banga, J.R.; Aloso, A.V. Vassiliadis. Dynamic optimization of chemical and biochemical processes using restricted second-order information 2001. Comput. Chem. Eng. 2001, 25, 539–546.
29. Newbold, P.; Carlson, W.L.; Thorne, B. Multiple Regression. *In Statistics for Business and Economics*, 6th ed.; Pfaltzgraff, M; Bradley, A., Eds.; Publisher: Pearson Education, Inc., New Jersey,07458, USA, 2007, pp. 454–537
30. Shi, H.; Zhang, X.; Gao, Y.; Wang, S.; Ning, Y. Robust Total Least Squares Estimation Method for Uncertain Linear Regression Model. Mathematics 2023, 11, 4354. https://doi.org/10.3390/math11204354
31. Goodfellow, I.; Bengio, Y.; Courville, A., Example: Linear Regression. *In Deep Learning*; The MIT Press, USA, 2016; pp. 104–113; ISBN 978-0262035613.