Scientific Research Publishing

Open Journal of Optimization, 2026, 15(2), 23-49
https://www.scirp.org/journal/ojop
ISSN Online: 2325-7091
ISSN Print: 2325-7105

Regression Neural Networks for Emulating Evolutionary Algorithm Predictions in Optimal Control

Iulian Arama[1], Viorel Mînzu[2*]

[1]Informatics Department, "Danubius" University, Galati, Romania
[2]Automation Department, "Dunarea de Jos" University, Galati, Romania
Email: *viorel.minzu@ugal.ro

How to cite this paper: Arama, I. and Mînzu, V. (2026) Regression Neural Networks for Emulating Evolutionary Algorithm Predictions in Optimal Control. *Open Journal of Optimization*, **15**, 23-49.
https://doi.org/10.4236/ojop.2026.152002

Received: March 22, 2026
Accepted: May 4, 2026
Published: May 7, 2026

Abstract

Many controlled processes, such as biochemical ones, are repetitive, similar to batch-organized processes. They generate Optimal Control Problems (OCPs) solved by optimal controllers, which often predict the best control values over a prediction horizon using a process model (PM). The latter can encompass imprecise, incomplete, and uncertain knowledge, nonlinearities, or may represent a distributed-parameter system, among other features. Using metaheuristics like the Evolutionary Algorithm (EA) within control structures is a practical approach for specific OCPs. The main drawback of this solution is its high computational demands, which can exceed the sampling period. Previously, the authors explored a new topic: whether a machine learning (ML) algorithm could "learn" the optimal behavior of the couple (EA and PM). Multilinear regression functions were trained and utilized within the ML controller, leading to a significant reduction in its execution time. This article extends the research and proposes using Regressive Neural Networks (RNNs) instead of regression functions, because they have limited capacity to capture nonlinearities. RNNs are more complex regression models, with a greater capacity to model EA-PM couples. The article meticulously describes how the training data is obtained. Algorithms for constructing the RNNs, developing the RNN predictor, and simulating the closed-loop are also proposed. The results are convincing: a) the control-loop dynamic was practically identical with that using the EA predictor; b) the performance indices were practically the same; c) the controller's execution time decreased remarkably. Using MATLAB for the Park Ramirez Problem, this time was 38 seconds, with the multilinear regression functions, it was 0. 08 seconds, and now with the RNN predictor, it is 0.0549 seconds. This outstanding reduction in prediction time opens the possibility to apply Receding Horizon Control (RHC) not only to slow processes but also to fast ones, which have small time constants.

Keywords

Evolutionary Algorithm, Machine Learning, Optimal Control, Simulation

1. Introduction

Many controlled processes, such as biochemical ones, are repetitive, similar to batch-organized processes. They generate Optimal Control Problems (OCPs) solved by optimal controllers, which often predict the best control values over a prediction horizon using a Process Model (PM).

A control structure that can implement closed-loop optimal solutions for OCPs is Receding Horizon Control (RHC). The essential characteristics of this control structure are:

- It includes a predictor for estimating the future optimal control values required to complete the optimization task.
- It also encompasses a Process Model that the predictor harnesses to generate its predictions.

The predictor implicitly implements a specific method to solve the optimization problem. That is why a metaheuristic algorithm [1]-[3] can be a practical choice, such as the EA, Particle Swarm Optimization, etc. Many articles highlight the ability of metaheuristic algorithms to solve complex problems in the field of dynamic optimization [4]-[9]. A systematic way to use metaheuristics in solving OCPs is presented in [10] [11].

The EA predictor has some characteristics:

- The main strength is that the EA can solve difficult optimization problems with imprecise, incomplete, and uncertain knowledge, nonlinearities, or in a distributed-parameter system [12].
- The main drawback is its computational complexity. The EA is called at each single sampling period to find a partial solution and perform many numerical integrations.
- The execution time is significant; sometimes it can be slightly less than the sampling period. That is why the EA predictor is typically used in control structures for slow processes.
- In this context, there are possibilities to reduce its execution time [13]-[15], but unfortunately, they diminish the solutions' accuracy.

Another way to address the main possible drawback of the EA predictor is to look for Machine Learning-based solutions [16]-[19].

In a previous study [20], the authors explored a new topic: whether a machine learning algorithm could "learn" the optimal behavior of the EA-PM pair. Multilinear regression functions [21]-[23] were trained and utilized within the so-called ML controller, leading to an outstanding reduction in its execution time.

In this paper, we extended the research in line with [20] and proposed replacing the regression functions with Regressive Neural Networks. What would be the

benefit of using RNNs instead of multilinear regression functions? The regression functions proposed in the previous article have limited ability to capture nonlinearities, for example. RNNs are more complex regression models, with a greater capacity to model EA-PM couples. The price to pay will be the model size, but not always.

The novelty of our approach, to the best of our knowledge, is that ML can learn even the EA's behavior within the control structure.

The control structure used in this work is known as Receding Horizon Control (RHC) [10] [24]-[26], a type of Model Predictive Control [27] [28]. This time, its prediction module uses an EA. It predicts, at each sampling moment, the optimal control sequence until the final time of the control horizon, using the process model. Then, the controller sends the first element of this sequence to the process and discards the other elements. After a sampling period, the controller picks up the process's next state and resumes the control action. The EA employs a PM and other blocks for numerical integration and different computations.

We shall consider RHC structures with two types of predictors: (a) one using an EA and (b) one employing Regression Neural Networks. The two structures control the same process to solve the same optimization problem. The controller (b), which will be called the RNN-controller in the sequel, is designed in a specific way that assumes the EA-controller (a) has already been implemented (at least to the simulation level).

The main objective of our work, presented in this article, is to develop the RNN-controller such that it emulates the EA-controller in optimizing the process modeled by the PM.

In article [20], the authors developed a controller called ML-controller that also emulates the EA-controller but uses a set of multilinear regression functions. The results were very good because the process's evolutions were practically identical; moreover, the ML-controller's execution time decreased significantly.

Briefly, the training data for the RNNs is collected from simulations of the closed-loop system equipped with the EA-controller. The optimal behavior of the EA-PM couple is "learned" by the RNNs, which are then used in the new controller.

The main contribution of this work is the complete design process of the RNN-controller, from data collection to closed-loop simulation.

Several algorithms were also proposed, and their corresponding programs were used to yield the results presented here. To write programs and to initially train and analyze the RNNs, we used the MATLAB system.

To exemplify our presentation, we needed a case study, which is a well-known Optimal Control Problem called the Park-Ramirez problem (PRP) (**Appendix A, A.1.4**). This serves as a benchmark problem [29] [30] that can illustrate a final cost OCP. We need to specify the following aspects:

- This research does not specifically aim to solve this problem.
- Solving this benchmark problem enables a comparison between previous work

using regression functions and current work using RNNs.

- The PRP has a simple structure that does not complicate the presentation.

As in [20], replacing thousands of PM's numerical integrations with a single RNN call, during the current sampling period, leads to a significantly shorter execution time than the EA predictor. Section 2 discusses some aspects of using EAs in optimal control systems. It also connects this article to our previous work by referencing appendices that review important theoretical elements and make the presentation self-contained. The concepts of optimal control profile and optimal trajectories are defined within the context of a discrete-time solution of PRP; the latter employs a version of EA. A general overview, Algorithm 1, of the RHC Controller is also included.

The sequence of activities needed to conduct the research is presented at the beginning of section 3, Materials and Methods. Subsection 3.1 is devoted to the collection of data needed for training, harnessing the programs already developed in the previous research. Then the data is split into datasets for each sampling period in Subsection 3.2; the training using these datasets will generate just as many RNNs. Construction sessions using an application of the MATLAB system will generate, as described in Subsection 3.3, RNN models, which are adequate for each sampling period.

The algorithm for constructing the RNNs-Algorithm 2 is described in Subsection 3.4. Algorithm 3 in Subsection 3.5 describes the simulation of the closed-loop over the control horizon using the new controller with an RNN predictor.

The results obtained in this work and the related discussions are the subject of Section 4, including the possible improvements. The overall conclusion is that the results are convincing:

- The dynamics of the control loop were practically the same as those generated by the EA predictor, including the state trajectory and the optimal control profile.
- The performance indices were practically the same.
- The controller's execution time decreased remarkably; it is even smaller than that of the controller using regression functions.

The RNN predictor succeeded in emulating the EA-PM couple. This fact shows that RHC can extend its applicability.

2. Optimal Control Using Evolutionary Algorithms

Since it is essential for our approach, the control structure is recalled in **Figure 1**; at the same time, this also helps us to define the main objective and contribution of our work more clearly. In short, the goal is to define the structure in **Figure 1(b)** capable of emulating the one in **Figure 1(a)**.

The objective of this work and the description of our method require many theoretical elements and notations that are used in our presentation. To reduce references to other works and make the presentation self-contained, the annexes of this article provide the necessary information to make it easier to follow the presentation.

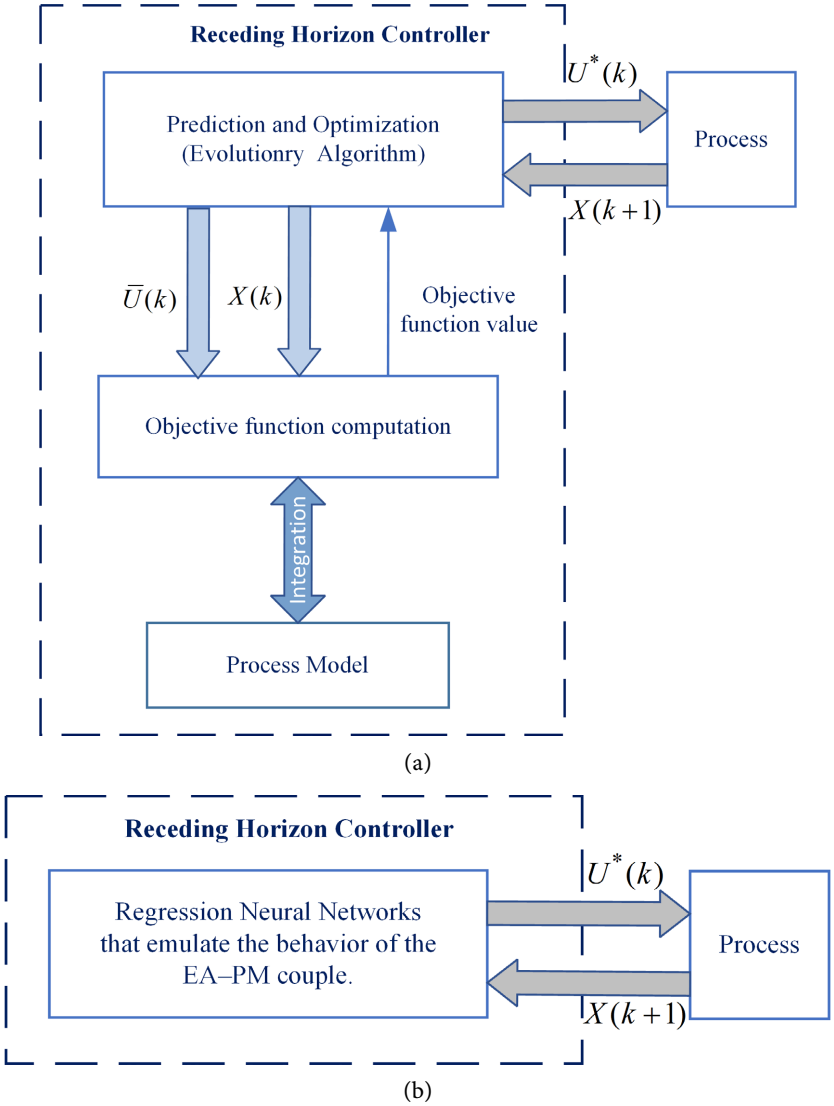


Figure 1. The Receding Horizon Control structure using (a) EA and (b) Regression Neural Networks.

Appendix A provides a short OCPs formulation, which is adequate for our presentation but is not a contribution of this work. This annex also presents the case study, the Park-Ramirez problem (PRP), which is a benchmark problem.

Regarding the PRP, **Appendix B** shows a discrete-time solution using a version of EA; this solution was also presented in [20]. At each sampling period k, the EA calculates the optimal sequence of control outputs, $V(k)$, for the remaining control horizon $[k, H]$. Finally, the first value of the sequence $V(k)$, becomes the controller's best output, denoted $U^*(k)$, sent toward the process:

$$U^*(k) \triangleq V(k)$$

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

learning models as well.

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.

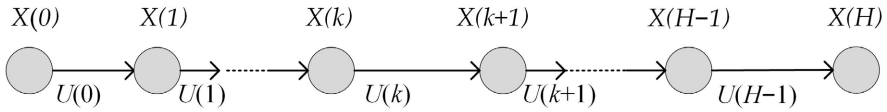


Figure 2. The state trajectory yielded by a control profile.

The controller, based on the EA, constructs the quasi-optimal CP for the initial state $X_0 = X(0)$ and the considered control horizon by concatenating the optimal controls ($U^*(k)$, $k = 0,\cdots,H-1$). It forces the process to go through a sequence of "optimal states" known as the optimal trajectory, $\Gamma(X_0)$:

$$\begin{aligned} \Omega(X_0) &\triangleq \langle U^*(0),U^*(1),\cdots,U^*(H-1)\rangle \\ \Gamma(X_0) &\triangleq \langle X_0,X^*(1),\cdots,X^*(H)\rangle \end{aligned} \tag{1}$$

These two sequences fully define the optimal evolution of the closed loop throughout the control horizon. Ideally, the optimal cost function will reach the value J_0 if the process and its model are identical. In practice, this value will be very close to J_0, making $\Omega(X_0)$ a quasi-optimal solution of our OCP.

Algorithm 1 provides the pseudocode for the RHC Controller, which is structured as shown in Figure 1. Regardless of the prediction type, the optimal control profile is found step-by-step using the following predictions:

$$\begin{aligned} &U^*(0) = predict(0,X_0); U^*(1) = predict(1,X(1)); \cdots; \\ &U^*(H-1) = predict(H-1,X(H-1)) \end{aligned} \tag{2}$$

We considered the predictor to be modeled by a function *predict*(k, X), where k is the index of the sampling period, and X is the current state vector of the process.

Algorithm 1. General implementation of the RHC Controller.

/* This is the function that the controller executes at each sampling period */

begin

1a Get the current value of the state vector, $X(k)$;
/* Update k and $X(k)$ by simuation using PM*/

1b "Read" the state vector of the real process, $X(k)$;
/* Read physically $X(k)$ */

2 $U^*(k) \leftarrow predict(k,X(k))$ /* see equation (2) */

3 Send $U^*(k)$ towards the Process. /Send in simulation or physically */

4 Wait for the next sampling period.

end

We need to make a decision. When the controller is simulated, as in this work, instruction #1a is considered, while instruction #1b only applies to a real-time implemented controller.

The implementation of the *predict* function depends on the choice made. We can use an EA like in **Appendix C**, a set of multilinear regression functions as in [20], or a set of RNNs as shown in this work.

3. Materials and Methods

It is important to emphasize the main goal of our work: to implement the predictor of the RHC structure using RNNs that emulate the optimal behavior of an EA operating within this framework. We expect the new controller to operate the same but more quickly.

The research activities will adopt the same approach as described in [20]. **Table 1** provides the sequence of activities for conducting the research in this work. The activities are numbered from 1 to 6. The blue lines display the results from performing the activities. These results may be structured data, objects as RNNs, functions and programs.

The first two activities are already addressed in the previous work [20], from which we can utilize the algorithms and programs already developed. Therefore, these algorithms and programs, together with the MATLAB system and its applications, can be regarded as the materials used in this work.

Table 1 presents a list of general activities guiding our research, which are further explained in the sequel to specify the methods developed in this work.

Table 1. Sequence of activities for conducting the research.

1. Collect data by simulating M closed-loop operations across the full control horizon [0, H].	
Results:	$state_1\ (H\times n)$, $state_2\ (H\times n)$, …, $state_M\ (H\times n)$; $UstarRHC\ (M\times n)$
2. Extract from the data collected at step 1 the dataset for each sampling period, $k=0, \ldots, H-1$.	
Results:	$SOCSK \in \mathbf{R}^{M\times(n+m)}$; $SOCSK_i \leftarrow \left[\left(X_i^*(k)\right)^{\mathrm{T}}\left(U_i^*(k)\right)^{\mathrm{T}}\right]$; $SOCSK = \begin{bmatrix} x_1(k)^1 & x_2(k)^1 & x_3(k)^1 & x_4(k)^1 & x_5(k)^1 & u(k)^1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_1(k)^i & x_2(k)^i & x_3(k)^i & x_4(k)^i & x_5(k)^i & u(k)^i \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_1(k)^M & x_2(k)^M & x_3(k)^M & x_4(k)^M & x_5(k)^M & u(k)^M \end{bmatrix}$
3. Conduct multiple construction sessions using the Regression Learner application with two main objectives: to determine the dataset size and to identify the best RNN model for each k.	
Results:	The best RNN model for each k, where $k=0, \ldots, H-1$. The size of the dataset used for training the RNN_k, dimdata(k). The training function for every RNN_k.

Continued

4. Develop an algorithm and a program to build the RNN for each k as objects that can be utilized for predictions, using the data and training functions obtained in the previous step.

Results:
- Cell array MODELNN{H,1}; The cell #k contains a reference to the object RNN_k
- Cell array DATAKTest{H,1}; The cell #k contains the table datakTest used to test the RNN_k
- Cell array DATAKTrain{H,1}; The cell #k contains the table datakTrain used to train the RNN_k

5. Develop an algorithm and the corresponding program that implements the optimal controller and simulates the evolution of the closed loop; it will predict the optimal control values using the RRNs built earlier.

Results: The ControlLoop_RNN algorithm and its corresponding MATLAB function.

6. Conduct multiple simulations for the closed-loop functioning to solve PRP, harnessing the ControlLoop_RNN.

Results:
- Statistics, execution time, and solution quality.
- Comparison with previous solutions, especially the controller developed in [20].

3.1. Step 1—Collection of Data

We need data describing the quasi-optimal behavior of the RHC structure functioning with the EA-PM couple. RNNs will use this to learn this behavior.

The main idea is to simultaneously record the optimal control outputs that the controller provides and the optimal PM's trajectories, which this one is forced to follow.

Remark 1: The RHC controller calculates the optimal control outputs using the PM; it does not depend on the actual process. That's why we are allowed to simulate the control loop's evolution over a control horizon and collect the control and state values for each sampling period.

In other words, to collect more data for training, we might simulate the closed-loop functioning over multiple control horizons and record the trajectories and the CPs.

Simulation of the closed-loop evolution

Appendix C recalls the prediction algorithm based on the EA, and an algorithm that simulates the closed-loop operation over the control horizon; both algorithms are detailed in [20], but are recalled here to make this presentation easier to follow.

Figure A1 in **Appendix C** shows the flowchart of the prediction function called Predictor_EA, which uses an EA for the control horizon [k, H]. Its inputs are k and the current state during this sampling period, denoted as X_0. The function returns a quasi-optimal predicted sequence that maximizes the performance index J_0.

To control the actual process in real-time throughout the entire control horizon [0, H], we can use the algorithm described by Figure A2 in **Appendix C**. The ControlLoop_EA algorithm is the version used for simulations when the real process matches the process model. It calls, for each sampling period, the functions Predictor_EA and RHC_RealProcessStep. The latter function is used to simulate the PM for a single step; this way, the next process state is determined by numer-

ical integration, and the simulation can continue. Finally, the simulation program will store the two sequences, $\Omega(X_0)$ and $\Gamma(X_0)$, in the vector *uRHC* and the matrix *state*, as well as the value of the performance index *J*.

As in our previous work, we have executed *M* times the ControlLoop_EA program to collect the sequences shown in equation (1). The Predictor_EA function has a stochastic character because the EA is so. Consequently, the ControlLoop_EA program inherits the stochastic character. Therefore, the *M* executions give different results, even if the process starts in the same state X_0. Practically, the control profiles and the quasi-optimal trajectories are different at each run but are very close to each other.

3.2. Step 2—Extraction of Datasets Characterizing Each Sampling Period

The *M* matrices, such as $state_1$ ($H \times n$), $state_2$ ($H \times n$), …, $state_M$ ($H \times n$), are stored in a cell array called STATE{1, *M*}, and the vectors of type uRHC are stored in a matrix UstarRHC(*M*, *H*).

Considering *k*, $0 \le k \le H-1$, which is already fixed, we create a matrix $SOCSK \in \mathbf{R}^{M\times(n+m)}$ that collects the States and Optimal Control values concerning step *k* from all the *M* experiences. Line *i*, $1 \le i \le M$, concatenates data from experience *i*:

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

Considering the data defined before, it holds:

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

$STATE_i$ is the ith element of the cell array *STATE*. In the PRP case, we obtained the matrix *SOCSK* presented as a result of activity **2** in Table 1.

3.3. Step 3—Construction Sessions for RNN Models

Using the Regression Learner application in MATLAB, we have conducted multiple sessions called Construction Sessions for each *k*, with two related objectives.

- Determining the dataset size for each $k = 0,\cdots,H-1$. From the available *M* data points collected in step 1, we chose the number of data points that yielded the best accuracy for RNN_k.
- Determining the best RNN model for the dataset chosen at this step *k*. Usually, the optimal model was the "optimizable" RNNs, which provided the best accuracy.

After a thorough analysis of the results, the best RNN was identified, and the training function was saved. The size of the training dataset was also stored in a vector called dimdata($1 \times H$).

As a general rule, for each k, the first 60 values of column *k* from the matrix SOCSK were used for the testing set, and the values from 61 to 60 + dimdata(*k*) were assigned to the training set.

3.4. Step 4: Algorithm to Build the RNN for Each *k*

The pseudocode below mainly describes the Algorithm 2 that generates the RNNs associated with each of the H sampling periods.

Algorithm 2. Models_Construction.

/* This pseudocode outlines the data generation, training, and testing processes of the RNNs for each sampling period.*/
Input: cell array STATE {M × 1} and the matrix UstarRHC (M×H); dimdata = [140,190,290,290,…190,190,290,290];
Output: cell array MODELNN{H × 1}, *DATAKTest,* and *DATAKTrain* storing objects that are RNNs and tables, respectively, for each sampling period

begin

1 **for** $k = 0 \ldots H\text{-}1$

2 **for** i = 0…M

3 $SOCSK_i \leftarrow \left[STATE_i(k,1:n) \;\; UstarRHC(i,k) \right]$

4 **end**

5 #Convert the matrix *SOCSK* into the table *datak*

6 *dataTest*←lines #1 - #60 of *tatak*

7 *dataTrain*←lines #61 - #[dimdata(*k*+1)+60] of *tatak*

8 *DATAKTest*{k}←*dataTest*

9 *DATAKTrain*{k}←*dataTrain*

10 [mdlRNN,*vrmse*] ← trainRNN(k,datakTrain);

11 MODELNN{k+1}←mdlRNN

12 #compute the Root Mean Squared Error for the testing phase (*RMSE*).

13 #display *vrmse* (rmse for the validation phase) and *RMSE*

14 **end**

15 #Save MODELNN, *DATAKTest, DATAKTrain* in workspace file Models&Data.

end

The RNN models are stored in a cell array {H,1} similar to their training and testing data. The dimdata array is determined at step 3, within the Construction Sessions.

The trainRNN function from instruction 10 takes the value of k as an input argument; it selects from the H training functions.

RMSEtest and *vrmse* represent the Root Mean Squared Error values for testing and validation phases. Instruction #12 stores the models in a file for further utilization.

3.5. Step 5-Simulation of the Closed-Loop with RNN Predictor

This section proposes the ControlLoop_RNN algorithm, which simulates Receding Horizon Control with a predictor that employs RNNs; the latter emulates the behavior of the EA-PM couple. This simulation over the control horizon is carried

out by a program unit, which is a function described in Algorithm 3. Calling this function repeatedly allows us to generate statistics.

Algorithm 3. ControlLoop_RNN.

/* This pseudocode simulates the RHC functioning equipped with a controller that predicts using RNNs */
function [*uRNN*, *FinalState*, *Jf*]←ControlLoop_RNN(MODELNN, X00)
Input: cell array MODELNN{*H*×1}; it contains a number of *H* trained RNN models;
Outputs: *uRNN*- the sequence of the optimal control values;
FinalState- the final state at the moment $k = H$;
Jf- the value of the performance index.

begin

1. #Initializations: *H*, umin, umax,
2. #Allocate memory space for two arrays: *uRNN*(1, *H*) and *state*(*H*+1, n); /*n=5*/.
3. X0←X00; /* X0 is the current state of the process during the simulation, X00 is the initial state at the start of the control horizon. */
4. state(1, –)←X0; /*Store the first state*/
5. for *kk* = *1…H;* /* *k* =0,…,*H*-1; *kk*=*k* + 1*/
6. mdlRNN←MODELNN{kk}; /* mdlRNN stores objects of type RNN, which are trained for the current sampling period*/
7. *uRNN*(*kk*)← The value predicted by the mdlRNN for the current state X0.
8. #The value of uRNN(kk) is constrained within established limits: umin and umax.
9. X0←step_PP_RH(*uRNN*(*kk*), X0);
10. state(kk+1, –)←X0;
11. **end**
12. *FinalState*←*state*(*H*+1, –);
13. Jf←*state*(*H*+1, 1) × *state*(*H*+1, 5)*;*
14. #Display *uRNN*, *FinalState*, Jf

return

The vector uRNN(1, H) is gradually filled with the optimal control values calculated by the predictor and sent to the process. Before the return instruction, this vector indicates an optimal control profile. The matrix "state" records the states through which the process passes, ultimately resulting in an optimal trajectory.

Mainly, the simulation is accomplished through an interaction in the loop between the RNN_Controller and the PM.

Remark 2: The RNN_Controller is realized through instructions #6–#8; the mdlRNN loads based on the value of *kk* and predicts the optimal control value according to the current state of the process, X0.

Instruction #9 determines the next state of the process by using the function step_PP_RH. The latter calculates the next state of the process through numerical integration based on the current state and the previous control value.

Instruction #13 calculates the performance index Jf. In our case, Jf corresponds to PRP.

4. Results and Discussion

The first two activities from Table 1, which make up step 1 and step 2 of our work were covered in the previous article [20]. Their programs and results were carried over in this work as available materials. The simulations of the closed-loop evolutions over the entire control horizon [0, H] produced $M = 400$ optimal CPs and their state trajectories. Therefore, for each sampling period, the corresponding RNN could be trained with datasets having a maximum of 400 data points.

4.1. Results of Constructing Regression Neural Networks

In step 3, we conducted multiple Construction Sessions using the Regression Learner application from MATLAB, aiming to develop a Regression Neural Network for each of the H sampling periods ($k = 0, \cdots, H-1$). The total number of RNNs trained exceeded 200. For each of them, we pursued two main objectives:

- to determine the dataset size for the training phase
- to identify the most suitable RNN model.

For each k, we decided by comparison what is the best RNN that can be further used in our work to develop an RHC controller.

Table 2 specifies for each k the size of the training data.

Table 2. The size of the training data for the best RNNs.

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
dimdata	140	190	290	290	290	240	290	190	290	190	190	190	190	290	290

After several Construction Sessions, we have identified the key characteristics of the models—things we need to leverage in our Construction Sessions. Table 3 summarizes these characteristics.

Table 3. Characteristics of the Construction Sessions used to identify and train the RNNs.

Significant training data sizes	140, 190, 240, and 290 data points;
The testing data size	60 unseen data points established at the beginning
The most used RNN model	Optimizable Neural Network
Optimization method	Bayesian optimization
The first quality criterion used to rank the RNNs	Root Mean Square Error in training
The second quality criterion taken into account.	R-Squared

For example, for $k = 10$, the RNN that gives the best accuracy is the "optimizable Neural Network" trained with 190 data points, and tested with 60 unseen data points. The chosen optimizer was Bayesian optimization. The resulting optimizable hyperparameters and the training results are given in Table 4.

Table 4. The optimized hyperparameters and the training results for $k = 10$.

Optimized hyperparameters		Training results	
LayerSizes	1	RMSE	0.0874
Activations	tanh	R -Squared	0.84
Lambda	0.00167	MSE	0.007655
Iteration Limit	200	MAE	0.069985
Standardize	false	MAPE	104.2%
		Training time	44.09 sec
		Model size (Compact):	5 kB

The accuracy performance of each RNN, across the sampling periods, is given in Table 5, considering the 60 data points of the testing set.

Table 5. The accuracy metrics across sampling periods.

k	Median test error	Mean Absolute Error	RMSE
0	0.0543	0.0783	0.1035
1	0.1529	0.1784	0.1025
2	0.1609	0.1656	0.1198
3	0.2562	0.3079	0.1835
4	0.4181	0.3996	0.3039
5	0.5753	0.5396	0.3280
6	1.1669	1.0580	0.3821
7	1.2750	1.2664	0.3455
8	1.4489	1.4223	0.3705
9	1.3357	1.2762	0.5578
10	0.2001	0.2326	0.0936
11	0.6639	0.6435	0.0828
12	0.6733	0.6550	0.0731
13	0.6884	0.6743	0.0556
14	1.0103	1.0036	0.0061

For each k, we have also obtained a training function that was called in step 4 to create the RNN object. This action was accomplished by running the program Models_Construction, which implements Algorithm 2.

4.2. Optimal Control Profiles and State Trajectories-Comparison with Previous Results

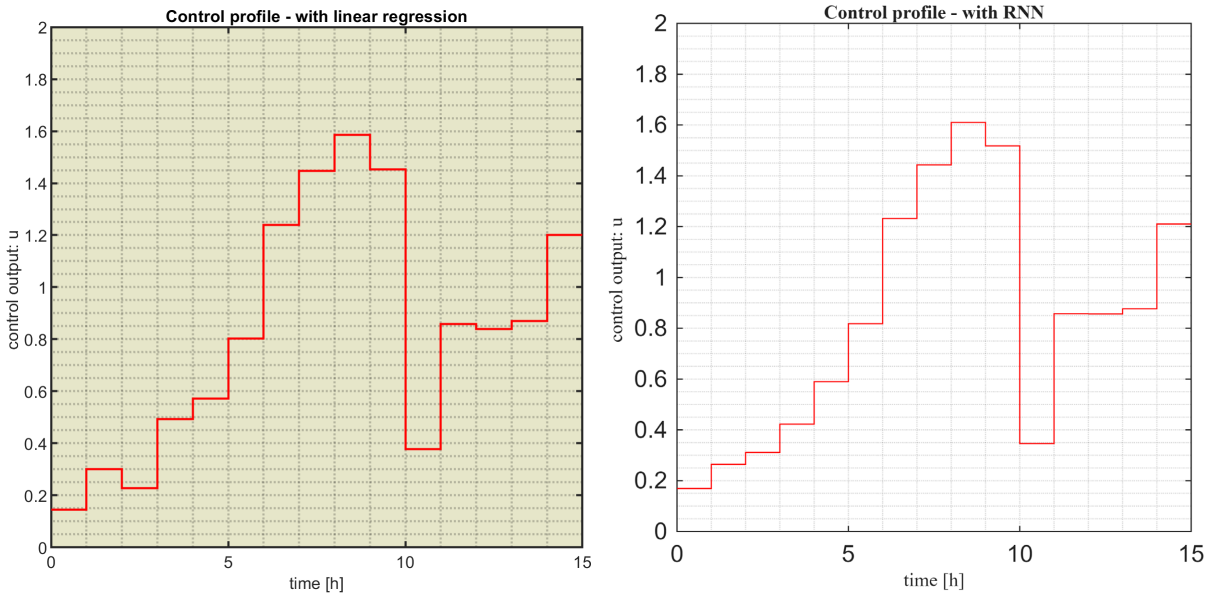
To see the new RNN controller in action, we have run the ControlLoop_RNN simulation program multiple times. Of interest were the shape of the control profiles and of the state trajectories, as well as the performance index achieved in the final state.

We made the comparison among the three evolutions of the closed-loop sys-

tem, using the three controllers:

- EA_controller: RHC with predictions made by EA.
- ML_controller: the control laws are actually the multilinear regression functions developed in the previous article [20]; each function is associated with a specific sampling period of the control horizon.
- RNN_controller: the RNNs predict the optimal control values; each RNN is associated with a specific sampling period of the control horizon.

The comparison among the CPs developed using machine learning is given in **Figure 3**. **Figure 3(a)** is taken from our article [20].



(a) (b)

Figure 3. The CP achieved by (a) the ML_controller, linear regression version; (b) the RNN_controller.

The CPs are very similar, which explains why the two performance indices are almost equal. The three state evolutions associated with the three controllers are shown in **Figure 4**. **Figure 4(a)** and **Figure 4(b)** are recalled here for comparison from our article [20].

The resemblance between the three processes' responses is very high and proves that

- The set of functions f_k successfully replicated the optimal behavior of the pair EA PM couple, as previously shown in [20].
- The set of RNNs also succeeded in emulating the optimal behavior of the EA-PM couple, which was our goal to demonstrate.

Furthermore, the performance index value ($J_{01} = 32.0986$) achieved by the CP in **Figure 4(b)** is very good because it matches the maximum value ($J_0 = 32.0936$) recorded in the M data points generated by the EA. Therefore, the PRP's solution identified by the ML model is also nearly optimal. The performance index's value achieved by the CP in **Figure 4(c)** is also very good $J_{02} = 32.106$, and very close to J_0.

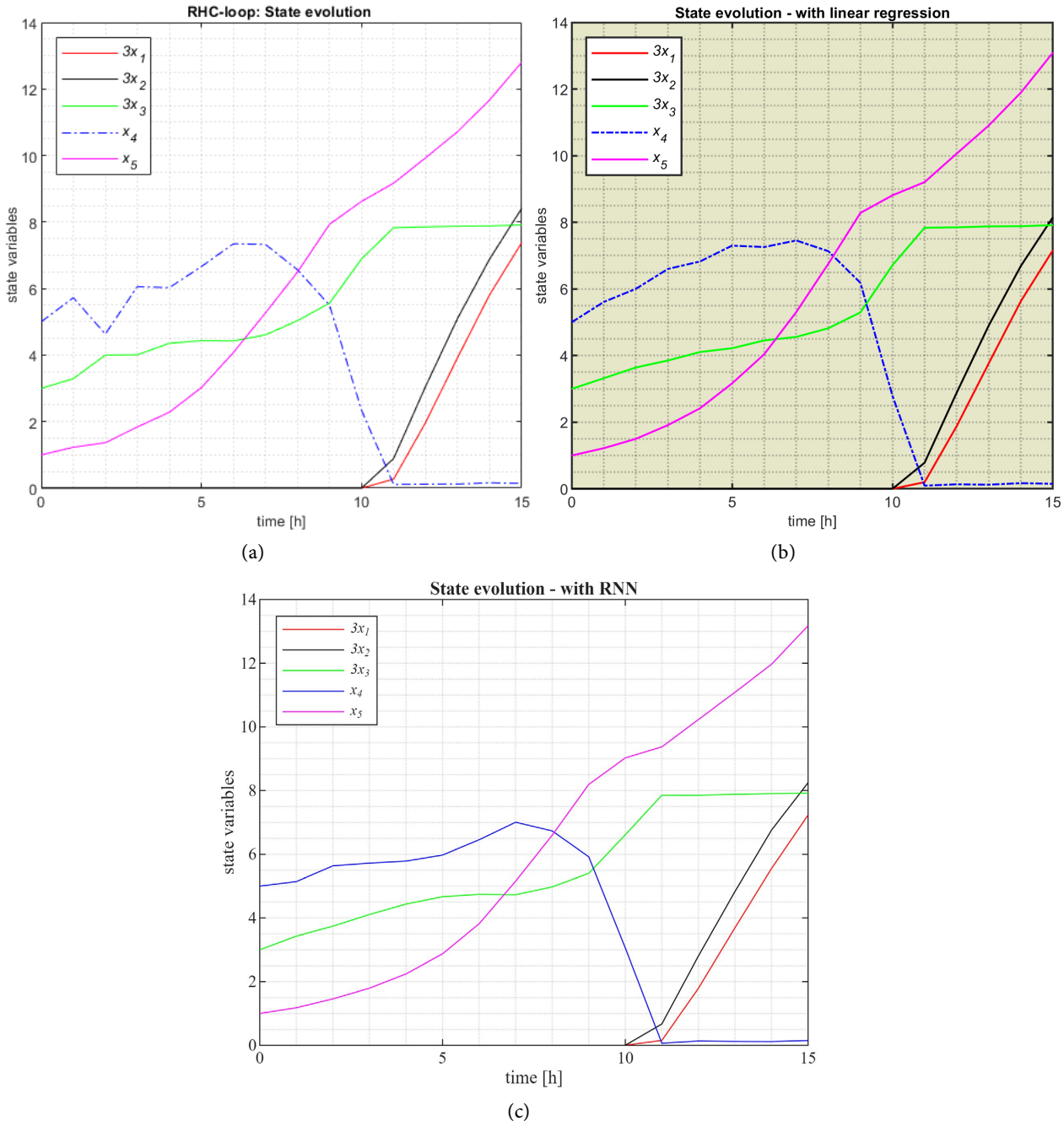


Figure 4. State evolution comparison: (a) Prediction with EA using RHC, (b) Machine learning prediction with a set of linear regression functions, (c) Machine learning prediction with a set of RNNs.

To evaluate the execution time of the closed-loop evolution over the control horizon, we ran the ControlLoop_RNN simulation program 50 times—due to its stochastic nature—under the same conditions as in our previous works. **Table 6** compares these results with those obtained using the other two controllers.

The typical value is the execution time of a specific simulation out of the 50, which is closest to the average. We also note that, from this perspective, the predictor with RNN is faster even than the predictor with regression functions.

Table 6. Duration of the controller action during the evolution of the closed loop.

Execution time - RNN controller	
Minimum value	0.0342 sec
Average value	0.0548 sec
Maximum value	0.1143 sec
Typical value	**0.0549** sec
Standard Deviation	0.0150
Execution time - controller with multilinear regression functions	0.08 sec
Execution time - EA controller	38.0 sec

4.3. Aspects of Designing and Validating the RNN Controller

There are a few aspects that we need to emphasize in the sequel.

4.3.1. The Design Procedure of the RNN Controller Is Carried out Offline

In this subsection, our perspective is that the design procedure develops an RNN controller that will be used in real-time.

1) An important aspect of the design procedure is that the data used for training is collected offline through simulation of the closed-loop over multiple control horizons (see Remark 1). We want to emphasize that this is not due to the fact that our work, presented in this paper, is a simulated study.

Remark 3: Even if the RNN controller will physically control a real process, the data will be collected through simulation. The latter is achieved by a program like ControlLoop_EA.

2) When it is used within the RHC structure, the EA predictor makes its predictions using only the PM.

Remark 4: The RNNs involved in this work must "learn" the behavior of the EA–PM couple. The actual process is not involved in collecting data.

The Construction Sessions, the selection of the RNNs, and the preliminary simulations are also carried out offline.

4.3.2. The Predictions Rely on the States of the Actual Process

The RNN controller, in its control action, will predict the control output according to the actual process's state. What is the process involved in? Either the PM or the real process, like in Algorithm 1, which provides for both possibilities (instructions #1a or #1b).

Remark 5. The state vectors presented to the RNNs to make predictions are not the same as those used in training. In particular, the states "read" from the current process have nothing to do with the training data because the PM is at least a little different from the real process. The process model can never perfectly model the real process.

In this paper, we used the RNN controller in simulations where the process model is identical to the PM, a situation called the simulated regime. Even in a simulated regime, the state vectors can be different from those of collected data, due to numerical perturbations and inherent numerical integrations (to calculate

the next state).

In what measure can the misfit between the PM and the actual process's characterization impact the quasi-optimality of the control loop?

4.3.3. Simulation of the Misfit between the PM and the Actual Process at the Level of State Variables

In our simulations, the difference between the PM and the actual process's characterization will be perceived only at the level of state variables. We used a very simple method to characterize this mismatch: the state variables are modified as if they are affected by noise. We consider the actual process's characterization to be symbolically represented at the level of state variables by this expression:

actual process's characterization = PM + an additive perturbation.

More specifically, each state variable $x_i(k)$ was perturbed by the value $z_i(k)$ to produce the perturbed state variable $xp_i(k)$

$$xp_i(k) \leftarrow x_i(k) + z_i(k);\ i = 1, \cdots, 5;\ k = 0, \cdots, H-1;$$

The value $z_i(k)$ is a kind of additive noise uniformly distributed in the range $[-L, L]$, where it holds:

$$L = p \cdot \left| x_i(k) \right|.$$

The value of p gives the amplitude of the perturbation, for example, $p = 0.5\%$. The resulting range of the noise will be 1% of the absolute value of the state variable at hand. We modified the simulation program ControlLoop_RNN to perturb the PM and analyzed the effects on the control action.

Figure 5 illustrates the two optimal CPs when the controller commands the PM or the actual process.

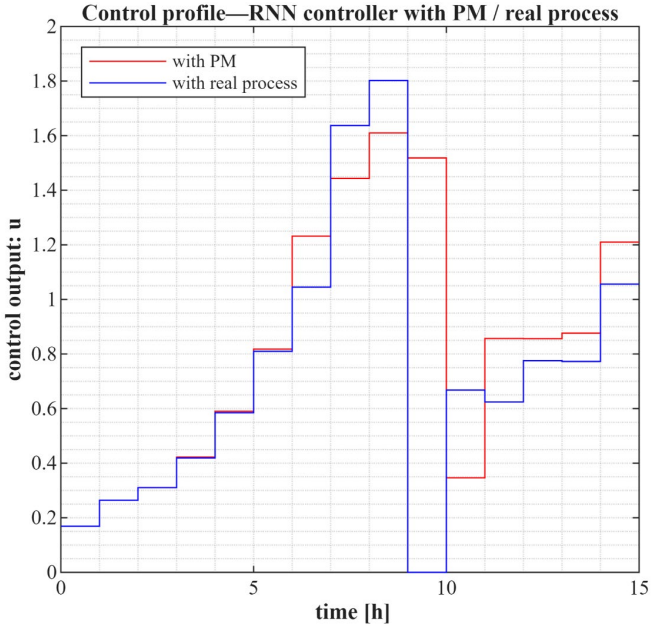


Figure 5. Comparison: CP when the controller works with the PM versus the actual process.

For the state trajectories, the comparison between the two situations is presented in Figure 6.

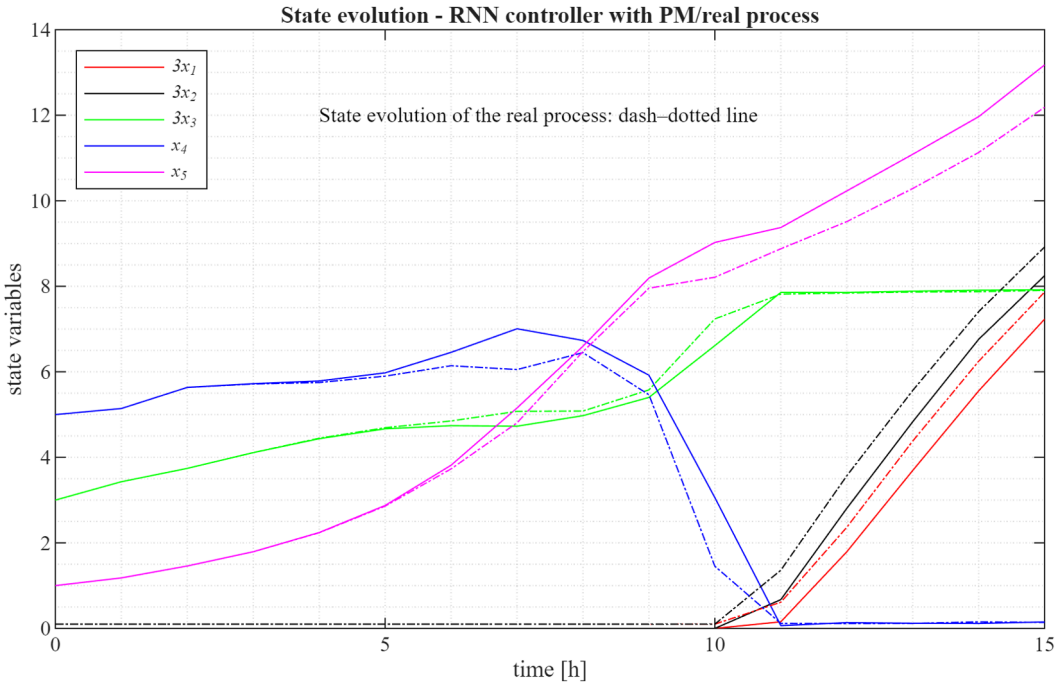


Figure 6. Comparison: CP when the RNN controller works with the PM versus the actual process.

The performance indexes are $J_1 = 31.7701$ and $J_2 = 31.5394$, respectively. We observe that the control loop has remained effective even with this deviation from the PM, and the CPs and state trajectories are slightly different. Both evolutions keep the quasi-optimal character.

Certainly, the deviation from the loop's behavior when operating with PM increases as the noise amplitude z grows larger. Beyond a certain value of L, the control loop will have a degraded behavior far away from a quasi-optimal character.

4.4. Possible Improvements of the RNNs' Generation

For each sampling period (#k), the RNN_k is stored in a workspace and used. This situation is justified by the fact that each sampling period has its own training data, and the RNN_k is relevant when using this data.

For $H = 15$, this situation is acceptable, even though the number of RNNs analyzed during the Construction Session is substantial. But how do we manage when the number of sampling periods is much larger (e.g., 100)?

There are two possible solutions:

- A single RNN model that will be used for all sampling periods. The training to obtain RNN_k *is achieved during the current sampling period.* Then it is used to predict according to the state vector.
- A single RNN for all sampling periods. This RNN will be trained offline with

the global collected data. Variable $\#k$ could be added to the predictor variables. The validation and testing sets will be selected in a more complex way. The unique RNN will be used to generate predictions during each sampling period. In this regard, further work remains.

4.5. The Relationship between the Controllers Using EA or RNN

If we need to modify the PM, even slightly, we have to redo the entire design process, including the simulation of the M control horizons, to generate the new training data. This happens even though the EA did not suffer any change. From the beginning, it was mentioned that the RNNs emulate the EA-PM couple.

Therefore, this paper does not suggest entirely abandoning the EA predictor.

Remark 6. If we choose to solve an optimization problem with an EA, the starting point would be the EA predictor. The faster RNN predictor will be developed once the EA-controller is designed and simulated to generate the training data. The RNN-controller is only a fast implementation variant for the EA-controller.

5. Conclusions

The work presented in this paper showed that a set of Regression Neural Networks can also be used to emulate ("learn") the optimal behavior of the couple EA-PM. Consequently, they can implement the predictor of the optimal controller. In comparison with the EA control loop, the simulated new control loop proved to have very good features:

- Similar state trajectory and optimal control profile.
- Quasi-equal performance index, which means keeping the solution's optimality.
- The controller's execution time is even shorter than that of the controller using regression functions.

Besides the qualitative features mentioned above, there are also positive and non-negligible characteristics of the presented approach for constructing the RNN-controller:

- The training data are obtained by simulations of the EA control loop working only with the PM.
- Data from the actual process is not needed because only the EA-PM couple is "learned".
- The entire design process of the RNN controller is performed online.
- If the control loop works in real-time, the state vectors provided to RNNs for predictions are those "read" from the actual process, not from the PM.
- Compared to linear regression functions, the main advantage of using RNNs is their greater *capacity* to capture the behavior of the EA-PM couple, especially when the PM has essential nonlinearities.

RNNs have a greater capacity to model complex systems than multilinear regressions. We have solved other OCPs whose process has essential nonlinearities.

The price to pay for this greater capacity to model complex systems is additional training effort, especially when each sampling period has its own RNN. The main drawback of the proposed approach is the number of sampling periods in the control horizon, which can be much larger than in our case study. This situation involves a huge number of RNNs that must be analyzed and trained. Therefore, we can now consider the proposed approach as one that shows the feasibility of implementation with RNNs. Managing a large value for H can be done through *possible improvements of the current version, within future research*. There are two possibilities to consider:

- A single RNN model will be utilized across all sampling periods, with the training to obtain RNNk conducted during the current sampling period.
- A single RNN will be used for all sampling periods. This model will be trained offline using the globally collected data. The unique RNN will generate predictions during each sampling period.

Remark 6 highlights that the RNN-controller is merely a more efficient way to implement the EA-controller. Therefore, the use of EAs in optimal control will remain relevant.

Conflicts of Interest

The authors declare no conflicts of interest.

References

[1] Siarry, P. (2016) Metaheuristics. Springer.

[2] Talbi, E. (2009) Metaheuristics. Wiley. https://doi.org/10.1002/9780470496916

[3] Kruse, R., Borgelt, C., Braune, C., Mostaghim, S. and Steinbrecher, M. (2016) Computational Intelligence—A Methodological Introduction. 2nd Edition, Springer.

[4] Faber, R., Jockenhövel, T. and Tsatsaronis, G. (2005) Dynamic Optimization with Simulated Annealing. *Computers & Chemical Engineering*, **29**, 273-290. https://doi.org/10.1016/j.compchemeng.2004.08.020

[5] Onwubolu, G. and Babu, B.V. (2004) New Optimization Techniques in Engineering. Springer.

[6] Valadi, J. and Siarry, P. (2014) Applications of Metaheuristics in Process Engineering, Springer International Publishing, 1-39. https://doi.org/10.1007/978-3-319-06508-3

[7] Minzu, V., Riahi, S. and Rusu, E. (2021) Optimal Control of an Ultraviolet Water Disinfection System. *Applied Sciences*, **11**, Article 2638. https://doi.org/10.3390/app11062638

[8] Abraham, A., Jain, L. and Goldberg, R. (2005) Evolutionary Multiobjective Optimization—Theoretical Advances and Applications. Springer.

[9] Mînzu, V., Rusu, E. and Arama, I. (2023) Execution Time Decrease for Controllers Based on Adaptive Particle Swarm Optimization. *Inventions*, **8**, Article 9. https://doi.org/10.3390/inventions8010009

[10] Minzu, V. and Serbencu, A. (2020) Systematic Procedure for Optimal Controller Implementation Using Metaheuristic Algorithms. *Intelligent Automation & Soft Computing*, **26**, 663-677. https://doi.org/10.32604/iasc.2020.010101

[11] Minzu, V. (2020) Optimal Control Implementation with Terminal Penalty Using Metaheuristic Algorithms. *Automation*, **1**, 48-65. https://doi.org/10.3390/automation1010004

[12] Minzu, V., Ifrim, G. and Arama, I. (2021) Control of Microalgae Growth in Artificially Lighted Photobioreactors Using Metaheuristic-Based Predictions. *Sensors*, **21**, 8065. https://doi.org/10.3390/s21238065

[13] Minzu, V., Riahi, S. and Rusu, E. (2021) Implementation Aspects Regarding Closed-Loop Control Systems Using Evolutionary Algorithms. *Inventions*, **6**, Article 53. https://doi.org/10.3390/inventions6030053

[14] Mînzu, V. and Arama, I. (2022) Optimal Control Systems Using Evolutionary Algorithm-Control Input Range Estimation. *Automation*, **3**, 95-115. https://doi.org/10.3390/automation3010005

[15] Mînzu, V., Georgescu, L. and Rusu, E. (2022) Predictions Based on Evolutionary Algorithms Using Predefined Control Profiles. *Electronics*, **11**, Article 1682. https://doi.org/10.3390/electronics11111682

[16] Goodfellow, I., Bengio, Y. and Courville, A. (2016) Machine Learning Basics. In: Dietterich, T., Bishop, C., Heckerman, D., Jordan, M., and Kearns, M., Eds., *Deep Learning*, The MIT Press, 95-161.

[17] Zou, S., Chu, C., Shen, N. and Ren, J. (2023) Healthcare Cost Prediction Based on Hybrid Machine Learning Algorithms. *Mathematics*, **11**, Article 4778. https://doi.org/10.3390/math11234778

[18] Cuadrado, D., Valls, A. and Riaño, D. (2023) Predicting Intensive Care Unit Patients' Discharge Date with a Hybrid Machine Learning Model That Combines Length of Stay and Days to Discharge. *Mathematics*, **11**, Article 4773. https://doi.org/10.3390/math11234773

[19] Albahli, S., Irtaza, A., Nazir, T., Mehmood, A., Alkhalifah, A. and Albattah, W. (2022) A Machine Learning Method for Prediction of Stock Market Using Real-Time Twitter Data. *Electronics*, **11**, Article 3414. https://doi.org/10.3390/electronics11203414

[20] Mînzu, V. and Arama, I. (2024) A Machine Learning Algorithm That Experiences the Evolutionary Algorithm's Predictions—An Application to Optimal Control. *Mathematics*, **12**, Article 187. https://doi.org/10.3390/math12020187

[21] Newbold, P., Carlson, W.L. and Thorne, B. (2007) Multiple Regression. In: Pfaltzgraff, M. and Bradley, A., Eds., *Statistics for Business and Economics*, 6th Edition, Pearson Education, Inc., 454-537.

[22] Shi, H., Zhang, X., Gao, Y., Wang, S. and Ning, Y. (2023) Robust Total Least Squares Estimation Method for Uncertain Linear Regression Model. *Mathematics*, **11**, Article 4354. https://doi.org/10.3390/math11204354

[23] Goodfellow, I., Bengio, Y. and Courville, A. (2016) Example: Linear Regression. In: Dietterich, T., Bishop, C., Heckerman, D., Jordan, M., and Kearns, M., Eds., *Deep Learning*, The MIT Press, 104-113.

[24] Hu, X. and Chen, W. (2005) Genetic Algorithm Based on Receding Horizon Control for Arrival Sequencing and Scheduling. *Engineering Applications of Artificial Intelligence*, **18**, 633-642. https://doi.org/10.1016/j.engappai.2004.11.012

[25] Hu, X.B. and Chen, W.H. (2005) Genetic Algorithm Based on Receding Horizon Control for Real-Time Implementations in Dynamic Environments. *Proceedings of the* 16*th Triennial World Congress*, Prague, 4-8 July 2005, 156-161.

[26] Mayne, D.Q. and Michalska, H. (1990) Receding Horizon Control of Nonlinear Systems. *IEEE Transactions on Automatic Control*, **35**, 814-824.

https://doi.org/10.1109/9.57020

[27] Goggos, V. and King, R.E. (1996) Evolutionary Predictive Control (EPC) *Computers & Chemical Engineering*, **20**, S817-S822. https://doi.org/10.1016/0098-1354(96)00144-5

[28] Chiang, P. and Willems, P. (2015) Combine Evolutionary Optimization with Model Predictive Control in Real-Time Flood Control of a River System. *Water Resources Management*, **29**, 2527-2542. https://doi.org/10.1007/s11269-015-0955-5

[29] Banga, J.R., Balsa-Canto, E., Moles, C.G. and Alonso, A.A. (2005) Dynamic Optimization of Bioprocesses: Efficient and Robust Numerical Strategies. *Journal of Biotechnology*, **117**, 407-419. https://doi.org/10.1016/j.jbiotec.2005.02.013

[30] Balsa-Canto, E., Banga, J.R., Alonso, A.A. and Vassiliadis, V.S. (2001) Dynamic Optimization of Chemical and Biochemical Processes Using Restricted Second-Order Information. *Computers & Chemical Engineering*, **25**, 539-546. https://doi.org/10.1016/s0098-1354(01)00633-0

Appendix A

Formulation of Optimal Control Problems

This section reviews the formulation of OCPs used to state the Park–Ramirez problem, the case study discussed in this paper. It can be used for many practical OCPs.

A.1. Optimal Control Problems with Final Cost

Since the structure of an OCP is well-known, we focus only on the key elements relevant to the problem used as an example in this section. Rigorous mathematical details will be omitted to keep the presentation simple.

A.1.1. Process Model

Our approach involves a controller with a process model made up of algebraic and ordinary differential equations.

$X(t)=\left[x_1(t)\cdots x_n(t)\right]^{\mathrm{T}}$ —a vector with n state variables;

$U(t)=\left[u_1(t)\cdots u_m(t)\right]^{\mathrm{T}}$ —a vector with m control variables.

The set of equations (A1) below exemplifies a process model.

A.1.2. Constraints

Although there are many constraint types, we mention only those used in this paper.

Control horizon: $t\in\left[t_0,t_{final}\right]\quad t_0=0$

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

Bound constraints: The values $u^j_{\min},u^j_{\max}$ are the limits of the variable $u_j(t)$.

If T represents the sampling period of the control system, then the control horizon can be divided into H sampling periods:

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

A.1.3. Cost Function

The problem is to identify the control function, $U(\cdot)$, that optimizes (maximizes or minimizes) a specific cost (objective) function J, whose general form is shown below.

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

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

Remark A1. *When the final cost is present, whether or not there is an integral term, the prediction horizon must end at the final time, which involves the greatest computational complexity.*

Given Remark A1, we focus solely on the final cost, as it aligns with our OCP.

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

The solution to the problem is the function $U(\cdot)$, which generates the optimal

value of the cost function known as the performance index.

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

A.1.4. An Example of OCP with a Final Cost

The Park–Ramirez problem (PRP) is a benchmark problem [26] [27] that can illustrate a final cost OCP. The nonlinear model represents a fed-batch reactor that produces secreted protein. PRP has been explored in many studies to analyze integration methods.

(PM):

$$\begin{aligned}
\dot{x}_1 &= g_1 \cdot \left(x_2 - x_1\right) - \frac{u}{x_5} \cdot x_1 \\
\dot{x}_2 &= g_2 \cdot x_3 - \frac{u}{x_5} \cdot x_2 \\
\dot{x}_3 &= g_3 \cdot x_3 - \frac{u}{x_5} \cdot x_3 \\
\dot{x}_4 &= -7.3 \cdot g_3 \cdot x_3 + \frac{u}{x_5} \cdot \left(20 - x_4\right) \\
\dot{x}_5 &= u \\
g_1 &= \frac{4.75 \cdot g_3}{0.12 + g_3} \\
g_2 &= \frac{x_4}{0.1 + x_4} \cdot \mathrm{e}^{-5.0 \cdot x_4} \\
g_3 &= \frac{21.87 \cdot x_4}{\left(x_4 + 0.4\right)\left(x_4 + 62.5\right)}
\end{aligned} \tag{A1}$$

The state variables have the following significance: $x_1(t)$—concentration of secreted protein, $x_2(t)$—concentration of total protein, $x_3(t)$—density of culture cell, $x_4(t)$—concentration of substrate, and $x_5(t)$—holdup volume.

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

Constraints:

Control horizon: $t \in \left[t_0, t_{final}\right]$; $t_0 = 0$, $t_{final} = 15$ h.

Initial state: $X(0) = X_0 = \left[0,\ 0,\ 1,\ 5,\ 1\right]^{\mathrm{T}} \in \mathbf{R}^5$.

Bound constraints: $u(t) \in \Omega \triangleq \left[0,\ 2\right]; 0 < t < t_{final}$

Performance index:

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

An open-loop solution cannot be used in real time because the real process and PM have different dynamics, even with small differences; this can lead to unpredictable efficiency. Our goal is to develop a controlled, optimal process (a closed-loop solution) starting from a given X_0, with the final cost being J_0.

Appendix B

A Discrete-Time Solution of an OCP using EA

To use an EA, we add the following constraint to our OCP:

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

Therefore, the control variables are modeled as step functions. For simplicity, the time instant kT will be denoted as k in the following. For example, within the sampling period $[kT, (k+1)T)$, the control vector is

$$U(kT) \equiv U(k) \triangleq [u_1(k), \cdots, u_m(k)]^{\mathrm{T}}.$$

We will call a "control profile" (CP) a sequence $U(0), U(1), \cdots, U(H-1)$ that spans the control horizon. It will produce the transfer diagram shown in Figure 2.

The EA produces 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 with the following structure:

$$\bar{U}(k) = \langle U(k), \cdots, U(H-1) \rangle$$

The vector $X(k)$ is the current state of the process. It is also the initial state for candidate prediction with H-k elements. This fact supports the name "Receding Horizon Control". The EA also calculates the corresponding state sequence (with $H-k+1$ elements):

$$\bar{X}(k) = \langle X(k), \cdots, X(H) \rangle.$$

When EA converges, it returns the optimal prediction sequence:

$$\bar{V}(k) \triangleq \arg\max_{\bar{U}(k)} J(\bar{U}(k), X(k)) = \langle V(k), \cdots, V(H-1) \rangle.$$

The first value of the sequence, $V(k)$, becomes the controller's optimal output, denoted $U^*(k)$, which is sent to the process. The remaining values in $V(k)$ are discarded. The controller will treat the next sampling period $[k+1, k+2)$.

Appendix C

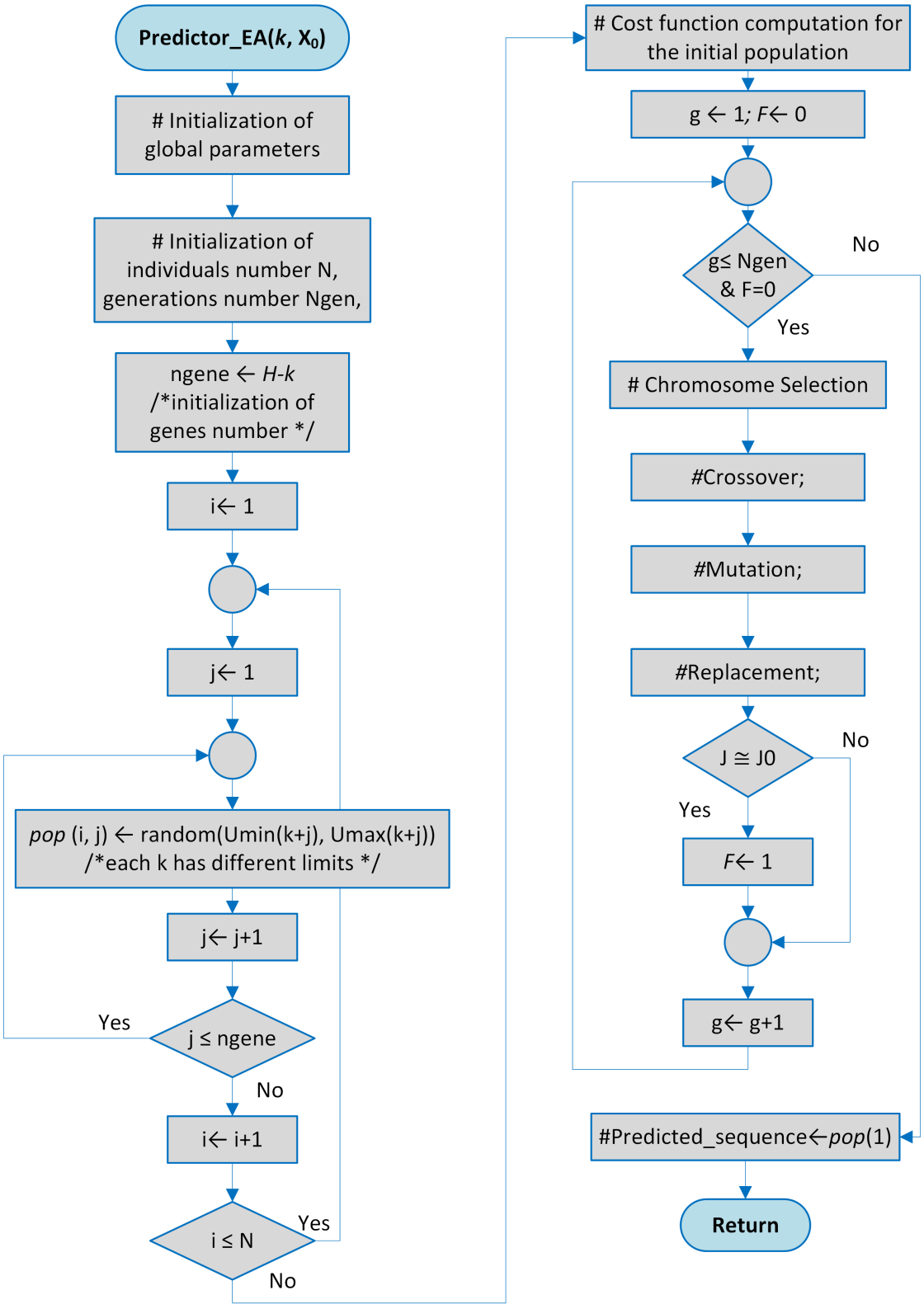


Figure A1. The flowchart of the prediction function using an EA for the control horizon [*k*, *H*].

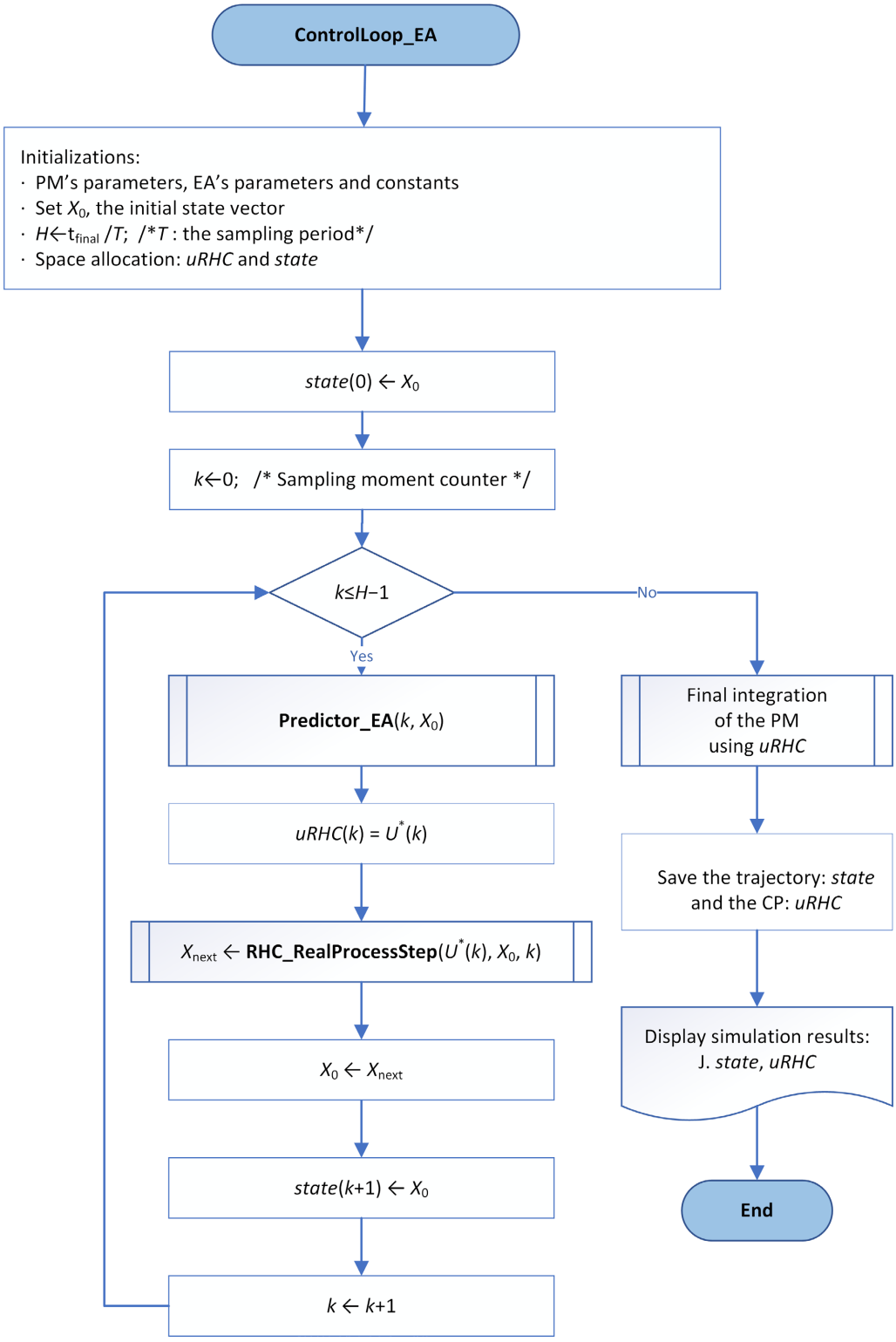


Figure A2. An algorithm using EA that provides an optimal state trajectory and a CP for the control horizon [0, H].

If we consider that the PM matches the real process, this algorithm can be used to simulate the closed-loop functioning over an entire control horizon.