sensors

MDPI

Article

Computationally Efficient Nonlinear Model Predictive Control Using the L_1 Cost-Function

Maciej Ławryńczuk * and Robert Nebeluk

Institute of Control and Computation Engineering, Faculty of Electronics and Information Technology, Warsaw University of Technology, ul. Nowowiejska 15/19, 00-665 Warsaw, Poland; R.Nebeluk@ia.pw.edu.pl

* Correspondence: M.Lawrynczuk@ia.pw.edu.pl

Abstract: Model Predictive Control (MPC) algorithms typically use the classical L_2 cost function, which minimises squared differences of predicted control errors. Such an approach has good numerical properties, but the L_1 norm that measures absolute values of the control errors gives better control quality. If a nonlinear model is used for prediction, the L_1 norm leads to a difficult, nonlinear, possibly non-differentiable cost function. A computationally efficient alternative is discussed in this work. The solution used consists of two concepts: (a) a neural approximator is used in place of the non-differentiable absolute value function; (b) an advanced trajectory linearisation is performed on-line. As a result, an easy-to-solve quadratic optimisation task is obtained in place of the nonlinear one. Advantages of the presented solution are discussed for a simulated neutralisation benchmark. It is shown that the obtained trajectories are very similar, practically the same, as those possible in the reference scheme with nonlinear optimisation. Furthermore, the L_1 norm even gives better performance than the classical L_2 one in terms of the classical control performance indicator that measures squared control errors.

Keywords: process control; model predictive control; L_1 cost function; optimisation

Citation: Ławryńczuk, M.; Nebeluk, R. Computationally Efficient Nonlinear Model Predictive Control Using the L_1 Cost-Function. *Sensors* **2021**, *21*, 5835. https://doi.org/10.3390/s21175835

Academic Editor: Alex Alexandridis

Received: 26 July 2021
Accepted: 25 August 2021
Published: 30 August 2021

1. Introduction

In Model Predictive Control (MPC), a dynamical model of the process is used on-line to repeatedly make predictions of the future values of the controlled variables and to optimise the current and future control policy [1,2]. Due to such a formulation, very good closed-loop control accuracy is obtained and all necessary constraints may be easily imposed on process variables. MPC algorithms are utilised for industrial process control, e.g., chemical reactors [3] and distillation columns [4]. In addition to that, MPC algorithms are also used for fast dynamical systems, e.g., electromagnetic mills [5], electromechanical systems [6], servomotors [7], quadrotors [8], autonomous vehicles [9–11], unmanned aerial vehicles [12] and stochastic systems [13]. Four conditions must be fulfilled to obtain good control quality: good measurements, a precise model, an adequate choice of the MPC cost function and a fast MPC computation scheme carried out on-line.

MPC algorithms depend heavily on precise measurements of process variables that are provided by sensors. In some versions of automatic control systems, e.g., those described in [12,14,15], it is stressed that all necessary variables must be measured because otherwise, a significant loss in control performance is unavoidable. If the measurements are not available, the typical approach is to perform on-line estimation using Kalman or Extended Kalman filters [16]. Furthermore, there are other alternative approaches to solve this problem to some extent. They are usually developed for particular applications. For example, when the measurement of lateral vehicle speed is not available, it can be estimated from other measurable parameters [10]. The study presented in [17] shows another approach of dealing with faulty sensors for linear systems by changing the representation of the process model. The work in [18] presents a real vehicle that uses an

external camera to detect obstacles and lanes on the road as well as external rear-corner radars to detect objects coming from the rear. An interesting example is presented in [19], where an anemometer is used to measure external disturbances such as wind force and direction. In [20], a depth sensor is installed for sea ship depth measurement. Moreover, the heave speed is obtained by calculating the derivative from depth sensor data. Finally, an example outside the vehicle field where fault-tolerant control is handled using MPC is stiction in control valves [21].

Typically, the minimised cost function used in MPC measures the sum of squared control errors predicted over some time horizon (the MPC-L_2 algorithm). Such a formulation is computationally simple and has good numerical properties. The use of the sum of absolute values of the predicted control errors (the MPC-L_1 algorithm) is much less popular. However, it may be easily verified that the second approach leads to better control quality [22–25]; the result is independent of the indicator used [23]. Furthermore, the classical L_2 cost function is not always suitable from the point of stability, as pointed out in [26]. It must be also stressed that in regression (model identification), the classical L_2 approach yields solutions fragile to outliers, while the L_1 norm leads to robustness [27].

It is important to emphasise that the numerical difficulty associated with the MPC-L_1 optimisation task depends on the type of dynamical model used for prediction. Provided that a linear model is used, one obtains a linear optimisation task [2,22], which can be solved very efficiently using the classical simplex method. Unfortunately, it is not possible when a nonlinear model is used. In such a case, a nonlinear optimisation problem with constraints must be solved at each sampling instant on-line, which may be computationally too demanding or completely impossible. Examples of nonlinear MPC-L_1 algorithms in which the Sequential Quadratic Programming algorithm is used are described in [23,24]. A trust-region sequential quadratic programming method is used in [28].

This work describes computationally efficient approaches to the MPC-L_1 algorithms for nonlinear processes. In spite of the fact that a nonlinear model is used for prediction, a computationally simple quadratic optimisation task is solved at each sampling instant on-line, and nonlinear optimisation is not used. The proposed solution is based on two concepts: a neural approximator and advanced on-line linearisation. A practical approach to nonlinear MPC is to perform model or trajectory linearisation on-line. As a result, an easy-to-solve quadratic optimisation problem is obtained. Details of a few such algorithms may be found in [29]. An alternative is to use the fuzzy approach, in which a combination of multiple linear models approximates the nonlinear process, which also results in quadratic optimisation, e.g., [30,31]. Unfortunately, in research carried out so far, only the classical MPC-L_2 cost function has been used. The main difficulty is the fact that the cost function in the rudimentary MPC-L_1 optimisation task is not differentiable. To make it possible, in the presented approach, the classical MPC-L_1 cost function is replaced by its differentiable representation. Such a representation is obtained by means of the neural approximator. Neural networks of the Multi Layer Perceptron (MLP) structure with two layers are used for two reasons. Firstly, they are universal approximators, capable of approximating nonlinear relations with great accuracy. Secondly, the obtained neural approximation is differentiable and poses no numerical problems for the presented computational scheme. The efficiency of the presented approaches is demonstrated for a simulated neutralisation process. In particular, the discussed MPC-L_1 algorithms are compared with the classical MPC-L_2 ones.

It is worth stressing that different neural structures have been used in MPC for different purposes. First of all, they may be used for prediction:

(a) Neural networks are most often used as black-box models of dynamical processes. Various structures are used: the classical MLP networks [29,32,33], Radial Basis Function (RBF) networks [34–36], Long Short-Term Memory (LSTM) [37–39] and Gated Recurrent Unit (GRU) [39] structures. Typically, the input–output neural models are used. The state–space neural models [29,40,41] are used when the state–

space process description is necessary, although such an approach is significantly less popular.

(b) Neural networks are also used in block-oriented cascade dynamical models, which consist of neural static blocks and linear dynamic blocks connected in series. Hammerstein [42–44] and Wiener [45–47] models are the most frequently used cascade models.

(c) Quasi-linear neural models [48,49]. In this approach, the dynamical model has the classical linear form, but its parameters depend on the operating point of the process and their values are determined on-line by neural networks.

(d) Neural step response models [50]. In this approach, time-varying coefficients of the model are computed on-line by a neural network.

(e) Neural multi-models [51,52]. In this approach, separate networks calculate the predictions for the consecutive sampling instants over the prediction horizon. As a result, the neural model is not used recurrently, which significantly simplifies training. Additionally, prediction errors are not propagated.

(f) Hybrid neural models [53]. In this approach, neural networks are used to calculate the parameters of the first-principle models.

(g) Neural networks may be used for modelling and MPC of distributed parameter systems [54–56].

Additionally, neural networks may be utilised to accelerate and simplify on-line calculations in MPC:

(a) Neural inverse static models are used to try to cancel process nonlinearity. In particular, such a method is frequently used when Wiener cascade models are considered [47,57]. As a result, a quadratic optimisation task is obtained in place of a nonlinear one.

(b) A neural approximator may be used to find the initial solution of the MPC optimisation problem, which speeds up calculations [58,59].

(c) Neural networks are able to approximate the MPC control law [60–62]. For training, sufficiently rich data sets are necessary, obtained for different operating points.

(d) Specialised recurrent neural networks may be used to solve the MPC optimisation task on-line [63,64]. As a result, numerical optimisation is not necessary.

The article is organised in the following way. Section 2 recalls the general MPC-L_1 and MPC-L_2 optimisation tasks. The main part of the article, given in Section 3, details the computationally efficient nonlinear MPC schemes in which the L_1 cost function is used. Section 4 thoroughly discusses simulation results for a neutralisation reactor. In particular, MPC-L_1 and MPC-L_2 algorithms are compared. Finally, Section 5 summarises the whole article.

2. Problem Formulation

In this work, MPC of a Single-Input Single-Output processes is considered. The input of the process, i.e., the manipulated variable, is denoted by u. The output of the process, i.e., the controlled variable, is denoted by y. The vector of decision variables calculated on-line at each sampling instant of MPC is defined as a set of N_u increments of the manipulated variable:

$$\triangle \boldsymbol{u}(k) = \begin{bmatrix} \triangle u(k|k) \\ \vdots \\ \triangle u(k+N_\mathrm{u}-1|k) \end{bmatrix} \tag{1}$$

where N_u is named the control horizon. The decision variables (1) are computed from a constrained MPC optimisation task. The general form of the rudimentary MPC optimisation task considered in this work is:

$$\begin{aligned} &\min_{\triangle \boldsymbol{u}(k)} \{J(k)\} \\ &\text{subject to} \\ &u^{\min} \leq u(k+p|k) \leq u^{\max},\ p = 0, \ldots, N_u - 1 \\ &\triangle u^{\min} \leq \triangle u(k+p|k) \leq \triangle u^{\max},\ p = 0, \ldots, N_u - 1 \end{aligned} \tag{2}$$

In this work, two types of constraints are considered: the magnitude constraints imposed on the manipulated variable, defined by $u^{\min}$ and $u^{\max}$, and the constraints imposed on the increments of that variable, defined by $\triangle u^{\min}$ and $\triangle u^{\max}$. These constraints are considered over the control horizon, N_u. Having calculated the decision vector (1) from the MPC optimisation problem (2), the first element of the obtained sequence is applied to the process and the whole computational scheme is repeated at the consecutive sampling instant.

The classical minimised cost function used in MPC (the MPC-L_2 algorithm) has the form:

$$J_2(k) = \sum_{p=1}^{N} \left(y^{sp}(k+p|k) - \hat{y}(k+p|k)\right)^2 + \lambda \sum_{p=0}^{N_u-1} \left(\triangle u(k+p|k)\right)^2 \tag{3}$$

The first part of the cost function measures the sum of squared control errors predicted over the prediction horizon N; the symbols $y^{sp}(k+p|k)$ and $\hat{y}(k+p|k)$ denote the set-point and the predicted value of the controlled variable, respectively, both for the future sampling instant $k+p$ known (it refers to the set-point) or calculated (it refers to the predictions) at the current sampling instant k. The predictions, $\hat{y}(k+p|k)$, are calculated using a dynamical model of the controlled process. The second part of the cost function is used to minimise excessive changes of the manipulated variable; λ is a weighting coefficient. Additionally, it provides good numerical properties.

In the the MPC-L_1 algorithm, the sum of absolute values of the predicted control errors over the prediction horizon are taken into account rather than the sum of squared errors. Hence, the minimised cost function is:

$$J_1(k) = \sum_{p=1}^{N} \left|y^{sp}(k+p|k) - \hat{y}(k+p|k)\right| + \lambda \sum_{p=0}^{N_u-1} \left(\triangle u(k+p|k)\right)^2 \tag{4}$$

To penalise significant changes of the manipulated variable and to obtain good numerical properties, the second part of the cost function is the same as in the MPC-L_2 formulation [23].

3. Computationally Efficient Nonlinear MPC Using the L_1 Cost-Function

Let us formulate the MPC-L_1 optimisation task. Taking into account the general MPC problem (2) and the MPC-L_1 cost function (4), we obtain:

$$\begin{aligned} &\min_{\triangle \boldsymbol{u}(k)} \left\{ J_1(k) = \sum_{p=1}^{N} \left|y^{sp}(k+p|k) - \hat{y}(k+p|k)\right| + \lambda \sum_{p=0}^{N_u-1} \left(\triangle u(k+p|k)\right)^2 \right\} \\ &\text{subject to} \\ &u^{\min} \leq u(k+p|k) \leq u^{\max},\ p = 0, \ldots, N_u - 1 \\ &\triangle u^{\min} \leq \triangle u(k+p|k) \leq \triangle u^{\max},\ p = 0, \ldots, N_u - 1 \end{aligned} \tag{5}$$

The use of a nonlinear model for prediction leads to two computational difficulties. Firstly, predictions $\hat{y}(k+p|k)$ are nonlinear functions of the calculated decision vector (1), which

means that the cost function is nonlinear. As a result, we obtain a nonlinear optimisation task that must be repeated at each sampling instant. Secondly, the absolute value function is not differentiable, which means that the classical gradient-based optimisation method cannot be used. In spite of the second of the mentioned computational difficulties, in the literature, is it possible to find applications of gradient-based nonlinear optimisation methods to solve the MPC-L_1 optimisation task (5) [23,24,28]. The objective of this work is to derive a much more computationally simple approach to the MPC-L_1 problem in which nonlinear optimisation is not used. To achieve this result, two concepts are used:

(a) The first part of the non-differentiable cost function (4) is replaced by its differentiable representation. For this purpose, a neural network approximation of the absolute value function is used.

(b) The $J_1(k)$ cost function with a neural approximator is differentiable but nonlinear in terms of the computed control moves (1). To simplify the calculation scheme, an advanced trajectory linearisation method is used. As a result, a simple-to-solve quadratic optimisation task is obtained in place of the nonlinear one. Quadratic optimisation problems, for $\lambda > 0$, have only one minimum, which is the global one.

3.1. Neural Approximation of the MPC-L_1 Cost-Function

Only the first part of the cost function $J_1(k)$ defined by Equation (4) is not differentiable. Hence, a differentiable approximator of the first part is only necessary. At first, let us define the predicted control error for the future sampling instant $k+p$ computed at the current instant k:

$$e(k+p|k) = y^{\mathrm{sp}}(k+p|k) - \hat{y}(k+p|k) \tag{6}$$

where $p = 1, \ldots, N$. Hence, the cost function $J_1(k)$ may be rewritten compactly:

$$J_1(k) = \sum_{p=1}^{N} |e(k+p|k)| + \lambda \sum_{p=0}^{N_{\mathrm{u}}-1} (\triangle u(k+p|k))^2 \tag{7}$$

In order to use quadratic optimisation in MPC, it is postulated to use the general form of the differentiable approximation of the cost function (7):

$$J_1(k) = \sum_{p=1}^{N} (\alpha(e(k+p|k)))^2 + \lambda \sum_{p=0}^{N_{\mathrm{u}}-1} (\triangle u(k+p|k))^2 \tag{8}$$

where $\alpha(e(k+p|k))$ describes some nonlinear function of the predicted control error. Comparing the cost functions (7) and (8), it is clear that the following approximation of the absolute value function is required:

$$(\alpha(e(k+p|k)))^2 = |e(k+p|k)| \tag{9}$$

for all $p = 1, \ldots, N$. In order to obtain a differentiable approximation of the absolute value function, the neural network of a Multi Layer Perceptron (MLP) type is used [65]. The network has two layers, the first of which (the hidden one) is nonlinear, and a linear output. It is defined by the following equation:

$$\alpha(e(k+p|k)) = w_0^2 + \sum_{i=1}^{K} w_i^2 \varphi\left[w_{i,0}^1 + w_{i,1}^1 (e(k+p|k))\right] \tag{10}$$

where K denotes the number of hidden neurons and φ is a nonlinear activation function. The weights of the first layer of the network are denoted by $w_{i,0}^1$ and $w_{i,1}^1$ for $i = 1, \ldots, K$. The weights of the second layer are denoted by w_0^2 and w_i^2 for $i = 1, \ldots, K$. Provided that a differentiable activation function φ is used, the function $\alpha(e(k+p|k))$ is also differentiable. It is of course true for the tanh function, which is used in this work.

3.2. Advanced Trajectory Linearisation of the MPC-L_1 Cost-Function

Thanks to the use of neural approximation (10) of the absolute value function, the cost function $J_1(k)$ defined by Equation (8) is differentiable, but still nonlinear. Additionally, the model of the controlled process is nonlinear. As a result, the cost function $J_1(k)$ is nonlinear in terms of the calculated future control increments (1). To simplify the problem, we use the advanced on-line trajectory linearisation approach. Linearisation is not carried out in a simple way, for the current or past operating point of the process, but for some assumed future trajectory of the manipulated variable, defined over the control horizon:

$$\boldsymbol{u}^{\mathrm{traj}}(k) = \begin{bmatrix} u^{\mathrm{traj}}(k|k) \\ \vdots \\ u^{\mathrm{traj}}(k+N_{\mathrm{u}}-1|k) \end{bmatrix} \tag{11}$$

Using the Taylor series expansion formula, the linear approximation of the multivariable function $\alpha(e(k+p|k))$ that has N_{u} arguments, defined by the vector $\boldsymbol{u}^{\mathrm{traj}}(k)$, is:

$$\begin{aligned} \alpha(e(k+p|k)) &= \alpha(e^{\mathrm{traj}}(k+p|k)) \\ &+ \sum_{r=0}^{N_{\mathrm{u}}-1} \frac{\partial \alpha(e^{\mathrm{traj}}(k+p|k))}{\partial u^{\mathrm{traj}}(k+r|k)} (u(k+r|k) - u^{\mathrm{traj}}(k+r|k)) \end{aligned} \tag{12}$$

for $p = 1, \ldots, N$. In order to find the partial derivatives in the right side of Equation (12), the neural approximator defined by Equation (10) is differentiated with respect to the assumed future trajectory of the manipulated variable (11), which gives:

$$\frac{\partial \alpha(e^{\mathrm{traj}}(k+p|k))}{\partial u^{\mathrm{traj}}(k+r|k)} = \sum_{i=1}^{K} w_i^2 \frac{\mathrm{d}\varphi(z_i^{\mathrm{traj}}(k+p|k))}{\mathrm{d}z_i^{\mathrm{traj}}(k+p|k)} \frac{\partial z_i^{\mathrm{traj}}(k+p|k)}{\partial u^{\mathrm{traj}}(k+r|k)} \tag{13}$$

for all $p = 1, \ldots, N$, $r = 0, \ldots, N_{\mathrm{u}} - 1$. The inputs of the hidden nodes of the neural network are:

$$z_i^{\mathrm{traj}}(k+p|k) = w_{i,0}^1 + w_{i,1}^1 (y^{\mathrm{sp}}(k+p|k) - \hat{y}^{\mathrm{traj}}(k+p|k)) \tag{14}$$

where $i = 1, \ldots, K$, $p = 1, \ldots, N$. If the tanh function is used in the hidden layer of the neural network:

$$\frac{\mathrm{d}\varphi(z_i^{\mathrm{traj}}(k+p|k))}{\mathrm{d}z_i^{\mathrm{traj}}(k+p|k)} = 1 - (\varphi(z_i^{\mathrm{traj}}(k+p|k)))^2 \tag{15}$$

Finally, from Equations (13)–(15), we obtain:

$$\frac{\partial \alpha(e^{\mathrm{traj}}(k+p|k))}{\partial u^{\mathrm{traj}}(k+r|k)} = -\sum_{i=1}^{K} w_{i,1}^1 w_i^2 \left(1 - (\varphi(z_i^{\mathrm{traj}}(k+p|k)))^2\right) \frac{\partial \hat{y}^{\mathrm{traj}}(k+p|k)}{\partial u^{\mathrm{traj}}(k+r|k)} \tag{16}$$

for all $p = 1, \ldots, N$, $r = 0, \ldots, N_{\mathrm{u}} - 1$. The partial derivatives in the right side of Equation (16) are derived for a particular model structure used for prediction. Differentiability of the model is required.

Let us stress that in Equation (12), independent linear approximations are obtained for the consecutive sampling instants over the prediction horizon, i.e., for $p = 1, \ldots, N$. In MPC, we need approximations of the absolute value of the predicted error over the whole prediction horizon. In order to simplify derivations, a compact vector matrix notation is used. The predicted trajectory of the function α, i.e., the vector form of Equation (12), is the following:

$$\boldsymbol{\alpha}(k) = \boldsymbol{\alpha}(\boldsymbol{e}^{\mathrm{traj}}(k)) + \frac{\mathrm{d}\boldsymbol{\alpha}(\boldsymbol{e}^{\mathrm{traj}}(k))}{\mathrm{d}\boldsymbol{u}^{\mathrm{traj}}(k)} (\boldsymbol{u}(k) - \boldsymbol{u}^{\mathrm{traj}}(k)) \tag{17}$$

where vectors of length N have the forms:

$$\boldsymbol{\alpha}(k) = \begin{bmatrix} \alpha(e(k+1|k)) \\ \vdots \\ \alpha(e(k+N|k)) \end{bmatrix} \tag{18}$$

and

$$\boldsymbol{\alpha}(\boldsymbol{e}^{\mathrm{traj}}(k)) = \begin{bmatrix} \alpha(e^{\mathrm{traj}}(k+1|k)) \\ \vdots \\ \alpha(e^{\mathrm{traj}}(k+N|k)) \end{bmatrix} \tag{19}$$

the vector of length N_{u}, corresponding to the vector of increments (1), is:

$$\boldsymbol{u}(k) = \begin{bmatrix} u(k|k) \\ \vdots \\ u(k+N_{\mathrm{u}}-1|k) \end{bmatrix} \tag{20}$$

and the $N \times N_{\mathrm{u}}$ matrix of partial derivatives has the structure:

$$\frac{\mathrm{d}\boldsymbol{\alpha}(\boldsymbol{e}^{\mathrm{traj}}(k))}{\mathrm{d}\boldsymbol{u}^{\mathrm{traj}}(k)} = \begin{bmatrix} \frac{\partial \alpha(e^{\mathrm{traj}}(k+1|k)}{\partial u^{\mathrm{traj}}(k|k)} & \cdots & \frac{\partial \alpha(e^{\mathrm{traj}}(k+1|k)}{\partial u^{\mathrm{traj}}(k+N_{\mathrm{u}}-1|k)} \\ \vdots & \ddots & \vdots \\ \frac{\partial \alpha(e^{\mathrm{traj}}(k+N|k)}{\partial u^{\mathrm{traj}}(k|k)} & \cdots & \frac{\partial \alpha(e^{\mathrm{traj}}(k+N|k)}{\partial u^{\mathrm{traj}}(k+N_{\mathrm{u}}-1|k)} \end{bmatrix} \tag{21}$$

The matrix (21), where its entries are defined by Equation (16), is calculated for the specific neural approximator and the nonlinear model used. Because in MPC we calculate not the future values of the manipulated variables (Equation (20)) but the corresponding increments (1), we have to express the vector equation of the linearised trajectory (Equation (17)) in terms of the vector of control increments $\triangle \boldsymbol{u}(k)$. We obtain:

$$\boldsymbol{\alpha}(k) = \frac{\mathrm{d}\boldsymbol{\alpha}(\boldsymbol{e}^{\mathrm{traj}}(k))}{\mathrm{d}\boldsymbol{u}^{\mathrm{traj}}(k)} \boldsymbol{J} \triangle \boldsymbol{u}(k) + \boldsymbol{\alpha}(\boldsymbol{e}^{\mathrm{traj}}(k)) + \frac{\mathrm{d}\boldsymbol{\alpha}(\boldsymbol{e}^{\mathrm{traj}}(k))}{\mathrm{d}\boldsymbol{u}^{\mathrm{traj}}(k)} (\boldsymbol{u}(k-1) - \boldsymbol{u}^{\mathrm{traj}}(k)) \tag{22}$$

where the auxiliary matrix of dimensionality $N_{\mathrm{u}} \times N_{\mathrm{u}}$ is:

$$\boldsymbol{J} = \begin{bmatrix} 1 & 0 & 0 & \ldots & 0 \\ 1 & 1 & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & 1 & \ldots & 1 \end{bmatrix} \tag{23}$$

and the vector of length N_{u} has the structure:

$$\boldsymbol{u}(k-1) = \begin{bmatrix} u(k-1) \\ \vdots \\ u(k-1) \end{bmatrix} \tag{24}$$

3.3. Formulation of the Computationally Simple MPC-L_1 Quadratic Optimisation Task

In order to obtain a quadratic optimisation MPC-L_1 optimisation problem, we take into account the general nonlinear MPC-L_1 optimisation task defined by Equation (5) in

which the first part of the minimised cost function is approximated by Equation (22). As a result, we obtain:

$$\begin{aligned}\min_{\triangle \boldsymbol{u}(k)} \Bigg\{ J_1(k) = \Bigg\| & \frac{\mathrm{d}\boldsymbol{\alpha}(\boldsymbol{e}^{\mathrm{traj}}(k))}{\mathrm{d}\boldsymbol{u}^{\mathrm{traj}}(k)} \boldsymbol{J} \triangle \boldsymbol{u}(k) + \boldsymbol{\alpha}(\boldsymbol{e}^{\mathrm{traj}}(k)) \\ & + \frac{\mathrm{d}\boldsymbol{\alpha}(\boldsymbol{e}^{\mathrm{traj}}(k))}{\mathrm{d}\boldsymbol{u}^{\mathrm{traj}}(k)} (\boldsymbol{u}(k-1) - \boldsymbol{u}^{\mathrm{traj}}(k)) \Bigg\|^2 + \| \triangle \boldsymbol{u}(k) \|^2_{\boldsymbol{\Lambda}} \Bigg\} \\ \text{subject to} & \\ \boldsymbol{u}^{\min} \leq \boldsymbol{J} \triangle \boldsymbol{u}(k) + \boldsymbol{u}(k-1) & \leq \boldsymbol{u}^{\max} \\ \triangle \boldsymbol{u}^{\min} \leq \triangle \boldsymbol{u}(k) & \leq \triangle \boldsymbol{u}^{\max} \end{aligned} \tag{25}$$

The matrix $\boldsymbol{\Lambda} = \mathrm{diag}(\lambda, \ldots, \lambda)$ is of dimensionality $N_{\mathrm{u}} \times N_{\mathrm{u}}$. Thanks to linearisation, the minimised cost function is quadratic in terms of the decision vector, $\triangle \boldsymbol{u}(k)$, and all the constraints are linear with respect to the vector $\triangle \boldsymbol{u}(k)$. The auxiliary vectors of length N_{u} are:

$$\boldsymbol{u}^{\min} = \begin{bmatrix} u^{\min} \\ \vdots \\ u^{\min} \end{bmatrix}, \boldsymbol{u}^{\max} = \begin{bmatrix} u^{\max} \\ \vdots \\ u^{\max} \end{bmatrix}, \triangle \boldsymbol{u}^{\min} = \begin{bmatrix} \triangle u^{\min} \\ \vdots \\ \triangle u^{\min} \end{bmatrix}, \triangle \boldsymbol{u}^{\max} = \begin{bmatrix} \triangle u^{\max} \\ \vdots \\ \triangle u^{\max} \end{bmatrix} \tag{26}$$

The obtained optimisation problem (25) can now be transformed into the standard form, typical of quadratic optimisation tasks:

$$\begin{aligned} & \min_{\boldsymbol{x}(k)} \Big\{ 0.5 \boldsymbol{x}^{\mathrm{T}}(k) \boldsymbol{H}_{\mathrm{QP}}(k) \boldsymbol{x}(k) + \boldsymbol{f}^{\mathrm{T}}_{\mathrm{QP}}(k) \boldsymbol{x}(k) \Big\} \\ & \text{subject to} \\ & \boldsymbol{A}(k) \boldsymbol{x}(k) \leq \boldsymbol{b}(k) \\ & \mathbf{LB} \leq \boldsymbol{x}(k) \leq \mathbf{UB} \end{aligned} \tag{27}$$

where the inequality constraints are defined by the matrix:

$$\boldsymbol{A}(k) = \begin{bmatrix} -\boldsymbol{J} \\ \boldsymbol{J} \end{bmatrix} \tag{28}$$

and the vector:

$$\boldsymbol{b}(k) = \begin{bmatrix} -\boldsymbol{u}^{\min} + \boldsymbol{u}(k-1) \\ \boldsymbol{u}^{\max} - \boldsymbol{u}(k-1) \end{bmatrix} \tag{29}$$

while the box constraints imposed on the decision vector are:

$$\mathbf{LB} = \triangle \boldsymbol{u}^{\min}, \ \mathbf{UB} = \triangle \boldsymbol{u}^{\max} \tag{30}$$

By differentiating the cost function $J_1(k)$ used in the quadratic optimisation problem (25) with respect to the decision variables, $\triangle \boldsymbol{u}(k)$, we obtain:

$$\begin{aligned} \frac{\mathrm{d}J(k)}{\mathrm{d}\triangle \boldsymbol{u}(k)} = & 2 \left(\boldsymbol{J}^{\mathrm{T}} \left(\frac{\mathrm{d}\boldsymbol{\alpha}(\boldsymbol{e}^{\mathrm{traj}}(k))}{\mathrm{d}\boldsymbol{u}^{\mathrm{traj}}(k)} \right)^{\mathrm{T}} \frac{\mathrm{d}\boldsymbol{\alpha}(\boldsymbol{e}^{\mathrm{traj}}(k))}{\mathrm{d}\boldsymbol{u}^{\mathrm{traj}}(k)} \boldsymbol{J} + \boldsymbol{\Lambda} \right) \triangle \boldsymbol{u}(k) \\ & + 2 \boldsymbol{J}^{\mathrm{T}} \left(\frac{\mathrm{d}\boldsymbol{\alpha}(\boldsymbol{e}^{\mathrm{traj}}(k))}{\mathrm{d}\boldsymbol{u}^{\mathrm{traj}}(k)} \right)^{\mathrm{T}} \Bigg(\boldsymbol{\alpha}(\boldsymbol{e}^{\mathrm{traj}}(k)) \\ & + \frac{\mathrm{d}\boldsymbol{\alpha}(\boldsymbol{e}^{\mathrm{traj}}(k))}{\mathrm{d}\boldsymbol{u}^{\mathrm{traj}}(k)} (\boldsymbol{u}(k-1) - \boldsymbol{u}^{\mathrm{traj}}(k)) \Bigg) \end{aligned} \tag{31}$$

Hence, the Hessian matrix necessary in the classical form of the quadratic optimisation task (27) is:

$$\boldsymbol{H}_{\mathrm{QP}}(k) = 2\boldsymbol{J}^{\mathrm{T}}\left(\frac{\mathrm{d}\boldsymbol{\alpha}(\boldsymbol{e}^{\mathrm{traj}}(k))}{\mathrm{d}\boldsymbol{u}^{\mathrm{traj}}(k)}\right)^{\mathrm{T}}\frac{\mathrm{d}\boldsymbol{\alpha}(\boldsymbol{e}^{\mathrm{traj}}(k))}{\mathrm{d}\boldsymbol{u}^{\mathrm{traj}}(k)}\boldsymbol{J} + 2\boldsymbol{\Lambda} \tag{32}$$

while the vector $\boldsymbol{f}_{\mathrm{QP}}$ is:

$$\boldsymbol{f}_{\mathrm{QP}}(k) = 2\boldsymbol{J}^{\mathrm{T}}\left(\frac{\mathrm{d}\boldsymbol{\alpha}(\boldsymbol{e}^{\mathrm{traj}}(k))}{\mathrm{d}\boldsymbol{u}^{\mathrm{traj}}(k)}\right)^{\mathrm{T}}\left(\boldsymbol{\alpha}(\boldsymbol{e}^{\mathrm{traj}}(k)) + \frac{\mathrm{d}\boldsymbol{\alpha}(\boldsymbol{e}^{\mathrm{traj}}(k))}{\mathrm{d}\boldsymbol{u}^{\mathrm{traj}}(k)}(\boldsymbol{u}(k-1) - \boldsymbol{u}^{\mathrm{traj}}(k))\right) \tag{33}$$

Calculation of the matrix of derivatives, as well as the trajectory, are explained for the specific form of the model used in simulations presented in Section 4.

In general, two versions of the presented algorithm are possible. Firstly, in the MPC algorithm with nonlinear prediction and linearisation along the trajectory (MPC-NPLT-L_1), trajectory linearisation may be executed once at each sampling instant for the assumed trajectory $\boldsymbol{u}^{\mathrm{traj}}(k)$. It means that only one quadratic optimisation problem is solved at every sampling instant. Alternatively, in the MPC algorithm with nonlinear prediction and linearisation along the predicted trajectory (MPC-NPLPT-L_1), a few internal iterations are possible at each sampling instant. The first internal iteration is the same as in the MPC-NPLT-L_1 scheme. In the consecutive ones, the optimal solution obtained in the previous internal iteration is used for linearisation.

4. Simulations

4.1. The Neutralisation Reactor

In this work, the neutralisation reactor is used as a benchmark to evaluate and compare all considered MPC methods. The process has one manipulated variable, which is a base (NaOH) stream q_1 (mL/s) and one controlled variable, which is the value of pH of the product. The detailed fundamental model of the process is given in [66]. The process is nonlinear, since its static and dynamic properties depend on the operating point. Hence, it is frequently used as a good benchmark to evaluate model identification algorithms and advanced nonlinear control methods.

4.2. Neutralisation Reactor Modelling for MPC

In this work, a Wiener model [67] of the neutralisation reactor is used in MPC for prediction. Such a block-orientated model is composed of two parts: a linear dynamic block followed by a nonlinear static one. The first part of the Wiener model is described by second-order dynamics:

$$v(k) = b_1 u(k-1) + b_2 u(k-2) - a_1 v(k-1) - a_2 v(k-2) \tag{34}$$

The auxiliary variable between two model blocks is denoted by v. The second part of the model is represented by a differentiable function:

$$y(k) = g(v(k)) \tag{35}$$

Combining Equations (34) and (35), we obtain the output of the Wiener model for the sampling instant k:

$$y(k) = g(v(k)) = g(b_1 u(k-1) + b_2 u(k-2) - a_1 v(k-1) - a_2 v(k-2)) \tag{36}$$

The relations between the process input and output variables, i.e., q_1 and pH, respectively, and the input and output of the model, i.e., u and y, respectively, are given by the following relations:

$$u = q_1 - q_{1,0},\ y = \mathrm{pH} - \mathrm{pH}_0 \tag{37}$$

where in the nominal operating point, $q_{1,0} = 15.5$ and $\mathrm{pH}_0 = 7$. In this study, a sigmoid-like neural network is used to represent the nonlinear static part of the model. Details of model training, validation and selection are given in [68]. An alternative, a Support Vector Machine (SVM) Wiener model, is presented in [69]. The sampling time of the Wiener model is 10 s.

4.3. Calculation of the Predicted Trajectories for the Wiener Model of the Neutralisation Reactor

The trajectory $\boldsymbol{\alpha}(\boldsymbol{e}^{\mathrm{traj}}(k))$, defined by Equation (19), is calculated using the neural approximator from Equation (10). The predicted control errors, $e(k+p|k)$, are computed from Equation (6). The predicted values of the controlled variable, $\hat{y}^{\mathrm{traj}}(k+p|k)$, are calculated from the Wiener model in the following way. From the linear block of the model (Equation (34)), we have:

$$v^{\mathrm{traj}}(k+1|k) = b_1 u^{\mathrm{traj}}(k|k) + b_2 u(k-1) - a_1 v(k) - a_2 v(k-1) \tag{38}$$

$$v^{\mathrm{traj}}(k+2|k) = b_1 u^{\mathrm{traj}}(k+1|k) + b_2 u^{\mathrm{traj}}(k|k) - a_1 v^{\mathrm{traj}}(k+1|k) - a_2 v(k) \tag{39}$$

$$\vdots$$

$$\begin{aligned} v^{\mathrm{traj}}(k+p|k) = {} & b_1 u^{\mathrm{traj}}(k-1+p|k) + b_2 u^{\mathrm{traj}}(k-2+p|k) \\ & - a_1 v^{\mathrm{traj}}(k-1+1|k) - a_2 v^{\mathrm{traj}}(k-2+p|k),\ p = 3,\ldots,N \end{aligned} \tag{40}$$

From the nonlinear block of the model (Equation (36)), the output predictions are:

$$\hat{y}^{\mathrm{traj}}(k+p|k) = g(v^{\mathrm{traj}}(k+p|k)) + d(k) \tag{41}$$

for any sampling time instant $k+p|k$, where $p = 1,\ldots,N$. The disturbance estimate is calculated using Equation (36), which yields:

$$d(k) = y(k) - g(b_1 u(k-1) + b_2 u(k-2) - a_1 v(k-1) - a_2 v(k-2)) \tag{42}$$

4.4. Calculation of the Matrices of Derivatives for the Wiener Model of the Neutralisation Reactor

The elements of the matrix $\frac{\mathrm{d}\boldsymbol{\alpha}(\boldsymbol{e}^{\mathrm{traj}}(k))}{\mathrm{d}\boldsymbol{u}^{\mathrm{traj}}(k)}$ defined by Equation (21) can be found using Equation (16). The derivatives $\frac{\partial \hat{y}^{\mathrm{traj}}(k+p|k)}{\partial u^{\mathrm{traj}}(k+r|k)}$ for all $p = 1,\ldots,N$ and $r = 0,\ldots,N_{\mathrm{u}}-1$ are calculated for the Wiener model used in the following way. By differentiating Equation (41), the following formula is obtained:

$$\frac{\partial \hat{y}^{\mathrm{traj}}(k+p|k)}{\partial u^{\mathrm{traj}}(k+r|k)} = \frac{\mathrm{d}g(v^{\mathrm{traj}}(k+p|k))}{\mathrm{d}v^{\mathrm{traj}}(k+p|k)} \frac{\partial v^{\mathrm{traj}}(k+p|k)}{\partial u^{\mathrm{traj}}(k+r|k)} \tag{43}$$

The first derivative in Equation (43) depends on the type of nonlinear static function. The second is calculated recursively for $k+1|k,\ldots,k+N|k$. For the first sampling time instant, $k+1$, calculated at the current time instant, k; by differentiating the formula (38), the following formula is acquired:

$$\frac{\partial v^{\mathrm{traj}}(k+1|k)}{\partial u^{\mathrm{traj}}(k+r|k)} = \begin{cases} b_1 & \text{for } r = 0 \\ 0 & \text{for } r > 0 \end{cases} \tag{44}$$

Prediction $\hat{y}(k+1|k)$ does not depend on control signals $u(k+1|k), u(k+2|k), \ldots$. With that in mind, the following formula is obtained:

$$\frac{\partial \hat{y}^{\mathrm{traj}}(k+1|k)}{\partial u^{\mathrm{traj}}(k+r|k)} = 0 \text{ for all } r > 0 \tag{45}$$

For the next time instant, $k+2$, by differentiating Equation (39), the following is acquired:

$$\frac{\partial v^{\text{traj}}(k+2|k)}{\partial u^{\text{traj}}(k+r|k)} = b_1 \frac{\partial u^{\text{traj}}(k+1|k)}{\partial u^{\text{traj}}(k+r|k)} + b_2 \frac{\partial u^{\text{traj}}(k|k)}{\partial u^{\text{traj}}(k+r|k)} - a_1 \frac{\partial v^{\text{traj}}(k+1|k)}{\partial u(k+r|k)} \tag{46}$$

where the derivatives $\frac{\partial u^{\text{traj}}(k+p|k)}{\partial u^{\text{traj}}(k+r|k)}$ can only take the values 0 or 1:

$$\frac{\partial u^{\text{traj}}(k+p|k)}{\partial u^{\text{traj}}(k+r|k)} = \begin{cases} 1 & \text{for } p = r \text{ or } (p > r \text{ and } r = N_{\text{u}} - 1) \\ 0 & \text{otherwise} \end{cases} \tag{47}$$

For the time instant $k+p|k$, where $p = 3, \ldots, N$, by differentiating Equation (40), we obtain:

$$\begin{aligned} \frac{\partial v^{\text{traj}}(k+p|k)}{\partial u^{\text{traj}}(k+r|k)} = b_1 \frac{\partial u^{\text{traj}}(k-1+p|k)}{\partial u^{\text{traj}}(k+r|k)} + b_2 \frac{\partial u^{\text{traj}}(k-2+p|k)}{\partial u^{\text{traj}}(k+r|k)} \\ - a_1 \frac{\partial v^{\text{traj}}(k-1+p|k)}{\partial u^{\text{traj}}(k+r|k)} - a_2 \frac{\partial v^{\text{traj}}(k-2+p|k)}{\partial u^{\text{traj}}(k+r|k)} \end{aligned} \tag{48}$$

Analogously, using Equation (45), a general regularity can be observed:

$$\frac{\partial \hat{y}^{\text{traj}}(k+p|k)}{\partial u^{\text{traj}}(k+r|k)} = 0 \text{ for } r \geq p \tag{49}$$

4.5. Organisation of Calculations

1. For the Wiener model, the disturbance estimate is calculated from Equation (42).
2. The trajectory of the manipulated variable, $\boldsymbol{u}^{\text{traj}}(k)$, that defines the linearisation point (Equation (11)), is formed. Three possible choices are discussed in the next section.
3. For the Wiener model, the derivatives $\frac{\partial \hat{y}^{\text{traj}}(k+p|k)}{\partial u^{\text{traj}}(k+r|k)}$, for all $p = 1, \ldots, N$ and $r = 0, \ldots, N_{\text{u}} - 1$, are calculated using Equations (43)–(49).
4. The matrix $\frac{\mathrm{d}\boldsymbol{\alpha}(\boldsymbol{e}^{\text{traj}}(k))}{\mathrm{d}\boldsymbol{u}^{\text{traj}}(k)}$ defined by Equation (21) is calculated using Equation (16).
5. The quadratic optimisation task (25) is solved.
6. In the case of the MPC-NPLT-L_1 algorithm, the first element of the obtained decision vector, $\triangle \boldsymbol{u}^{\text{opt}}(k)$, is applied to the process, i.e., $u(k) = \triangle u^{\text{opt}}(k|k) + u(k-1)$.
7. In the case of the MPC-NPLPT-L_1 algorithm, steps 2–5 are repeated a few times (in this work, maximally five times). The trajectory used for linearisation is defined as $\boldsymbol{u}^{\text{traj}}(k) = \boldsymbol{J} \triangle \boldsymbol{u}^{\text{opt}}(k) + \boldsymbol{u}(k-1)$, where the matrix $\boldsymbol{J}$ and the vector $\boldsymbol{u}(k-1)$ are defined by Equations (23) and (24), respectively, and $\triangle \boldsymbol{u}^{\text{opt}}(k)$ denotes the optimal solution calculated in the previous internal iteration (for the current sampling instant k). When the internal iterations are terminated, the first element of the decision vector computed in the last internal iteration is applied to the process.

4.6. Comparison of MPC-L_1 and MPC-L_2 Algorithms for the Neutralisation Reactor

At first, let us verify the usefulness of the MLP neural network to serve as an approximator of the absolute value function of the predicted control error (Equation (10)). The neural network with $K = 10$ hidden nodes of the tanh type is used. Figure 1 compares the non-differentiable absolute value (abs) function and its differentiable neural approximation. The range of the control error, e, is adequate for further use of the neural approximator in MPC for the considered neutralisation reactor. For the chosen neural network structure and the number of hidden nodes, the obtained approximation accuracy is very good.

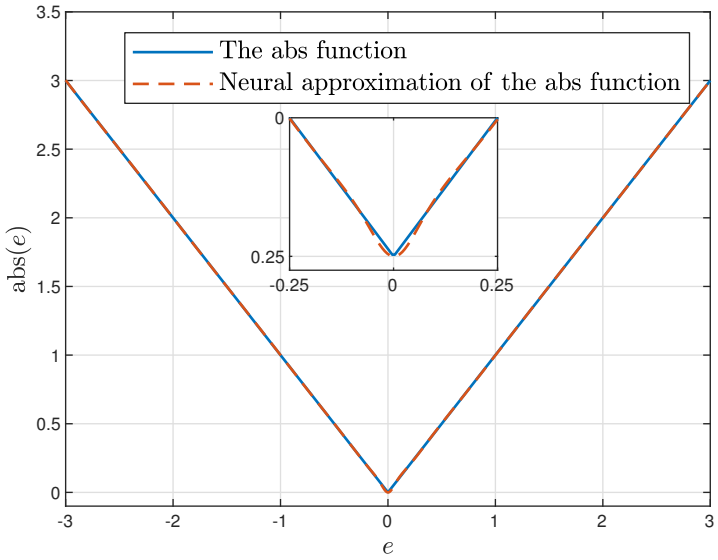

Figure 1. The absolute value function vs. its neural approximation.

In the following part of the article, the following MPC algorithms are considered:

- MPC-NO-L_1: the MPC algorithm with nonlinear optimisation with the L_1 norm used in the first part of the minimised cost function defined by Equation (4). The resulting nonlinear optimisation task is given by Equation (5). Two versions of the MPC-NO-L_1 are considered: the non-differentiable absolute value function or its differentiable neural approximation may be used.
- MPC-NPLT1-L_1: the discussed MPC algorithm with nonlinear prediction and linearisation along the trajectory. The neural network is used to approximate the non-differentiable absolute value function. Moreover, a linear approximation of the nonlinear trajectory of the predicted control errors over the prediction horizon is used in the cost function. The resulting quadratic optimisation task is given by Equation (25). The trajectory used for linearisation, i.e., $\boldsymbol{u}^{\mathrm{traj}}(k)$ (Equation (11)) is constant; all its elements are equal to the value of the manipulated variable calculated at the previous sampling instant, i.e., $u(k-1)$, and applied to the process.
- MPC-NPLT2-L_1: the trajectory used for linearisation is defined by the last $N_{\mathrm{u}}-1$ elements of the optimal solution $\triangle\boldsymbol{u}(k)$ computed at the previous sampling instant. Only the first element of this sequence is actually used for control.
- MPC-NPLT3-L_1: the trajectory used for linearisation is constant, all its elements are equal to the value of the process input corresponding to the current output set-point. For this purpose, the inverse static model of the process is used: $u^{\mathrm{sp}}(k) = \tilde{g}(y^{\mathrm{sp}}(k))$. In this work, a neural network of the MLP type with two layers serves as the inverse model (the first nonlinear layer contains 10 hidden nodes of the tanh type).
- MPC-NPLPT-L_1: the discussed MPC algorithm with nonlinear prediction and linearisation along the predicted trajectory. In this case, trajectory linearisation and quadratic optimisation are repeated maximally five times at each sampling instant. The trajectory used for linearisation is taken from the previous internal iteration of the algorithm. In the first internal iteration, for linearisation, the trajectory obtained from the inverse static model for the current set-point is used, exactly as it is done in the MPC-NPLT3-L_1 scheme.

Similarly, the following MPC algorithms with the L_2 cost function (3) are considered:

- MPC-NO-L_2: the MPC algorithm with nonlinear optimisation with the L_2 norm used in two parts of the minimised cost function. The resulting nonlinear optimisation task is given by Equation (2).

- MPC-NPLT1-L_2, MPC-NPLT2-L_2 and MPC-NPLT3-L_2: the MPC algorithm with nonlinear prediction and linearisation along the trajectory [29]. Trajectory linearisation and quadratic optimisation are performed once at each sampling instant.
- MPC-NPLPT-L_2: the MPC algorithm with Nonlinear Prediction and Linearisation along the Predicted Trajectory [29]. Trajectory linearisation and quadratic optimisation are repeated maximally five times at each sampling instant.

In all simulations presented next, the same parameters of MPC are used: $N = 10$, $N_u = 3$, $\lambda = 1$. The constraints are imposed on the magnitude of the manipulated variable: $q_1^{min} = 0$, $q_1^{max} = 30$. All simulations are performed in MATLAB. The `fmincon` and `quadprog` functions are used for nonlinear and quadratic optimisation, respectively; default parameters are used in both cases.

Firstly, let us evaluate the efficiency of two versions of the MPC-NO-L_1 control scheme. In both of them, nonlinear optimisation is used at each sampling instant on-line, but the objective of the comparison is to only check the efficiency of the neural approximator of the absolute value function. In the first case, the non-differentiable absolute value function is used in the first part of the minimised cost function, while in the second case, a neural approximator is considered. Figure 2 depicts the obtained results for a few changes of the set-point. In general, the non-differentiable absolute value function results in some numerical problems for the optimisation routine. The influence of such problems on the resulting trajectories of the process is best visible for the second step of the set-point. For better comparison, two bottom panels of Figure 2 show an enlarged fragment of the obtained results. For the non-differentiable absolute value function, the sign of the real control error changes a few times, while the neural approximator smooths the trajectory of the process output.

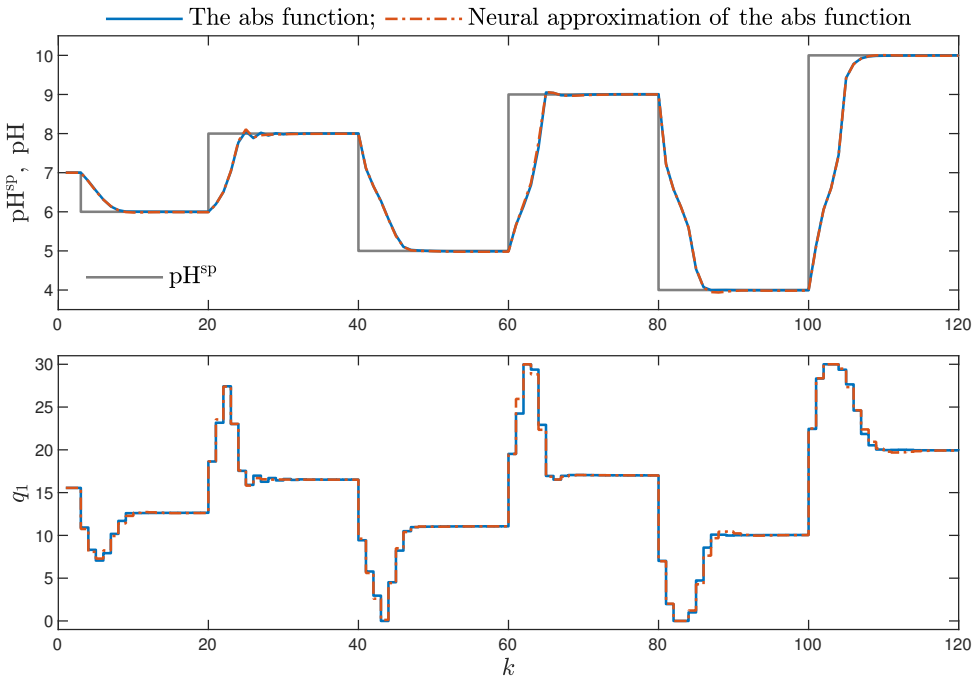

Figure 2. *Cont.*

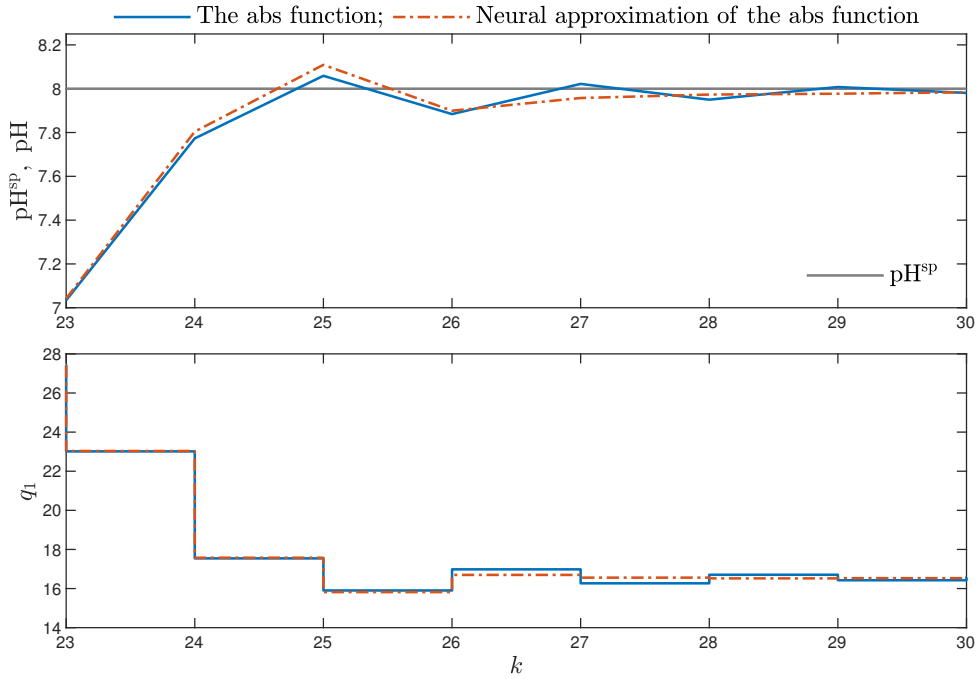

Figure 2. Simulation results: the MPC-NO-L_1 algorithm using the non-differentiable absolute value function vs. the MPC-NO-L_1 algorithm using the neural approximation of the absolute value function; two top panels show the results for the whole simulation horizon, two bottom panels show an enlarged fragment for the sampling instants $23 \leq k \leq 30$.

Figure 3 compares the simulation results of three computationally efficient MPC algorithms with one repetition of trajectory linearisation and quadratic optimisation at each sampling instant, but the trajectory used for linearisation, i.e., $\boldsymbol{u}^{\text{traj}}(k)$, is chosen in different ways. In all algorithms, the L_1 norm cost function is used. The first two initialisation methods, used in the MPC-NPLT1-L_1 and MPC-NPLT2-L_1 algorithms, lead to some problems for the second set-point step. The explanation is the following: when the set-point changes abruptly, the linear approximation of the predicted trajectory performed using the past operating point (MPC-NPLT1-L_1) or the part of the optimal trajectory for the past operating point (MPC-NPLT2-L_1) differs significantly from the true nonlinear trajectory, performed in the MPC-NO-L_1 algorithm without any linearisation (Figure 2). Of course, increasing the value of the coefficient λ would solve the problem, but it leads to slower control. The best results are obtained in the MPC-NPLT3-L_1 algorithm, in which the trajectory used for linearisation is constant, and all its elements are equal to the value of the process input corresponding to the current output set-point. Such an input value is calculated using an inverse neural static model of the process $u^{\text{sp}}(k) = \tilde{g}(y^{\text{sp}}(k))$.

Let us remind ourselves that our objective is to obtain an MPC algorithm that uses the L_1 cost function but it should be computationally efficient, i.e., quadratic optimisation should be used in place of nonlinear optimisation. On the other hand, the "ideal" MPC-NO-L_1 algorithm is treated as the reference. In different words, we develop a method that is computationally much simpler, but we hope to obtain control accuracy similar to that possible in the reference MPC-NO-L_1 algorithm. Figure 4 compares the best computationally efficient MPC algorithms with one repetition of trajectory linearisation and quadratic optimisation at each sampling instant, i.e., the MPC-NPLT3-L_1 scheme, with the reference MPC-NO-L_1 scheme. We can see that the MPC-NPLT3-L_1 algorithm discussed in this work gives very good trajectories, very similar to those obtained in the MPC-NO-L_1 control approach. However, there are some small discrepancies between the obtained trajectories.

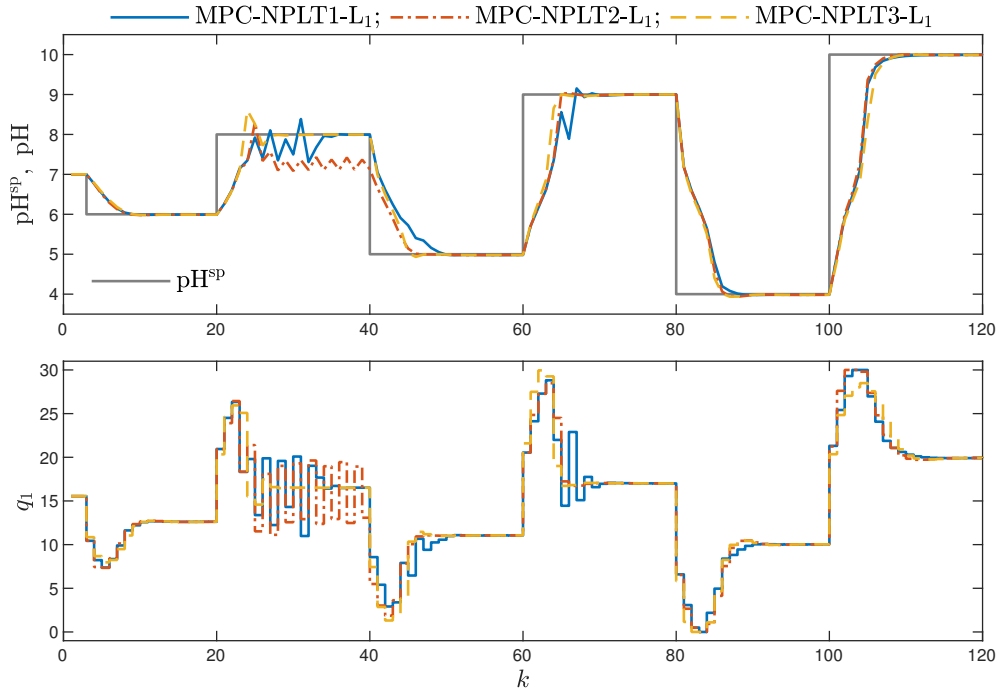

Figure 3. Simulation results: the MPC-NPLT1-L_1, MPC-NPLT2-L_1 and MPC-NPLT3-L_1 algorithms.

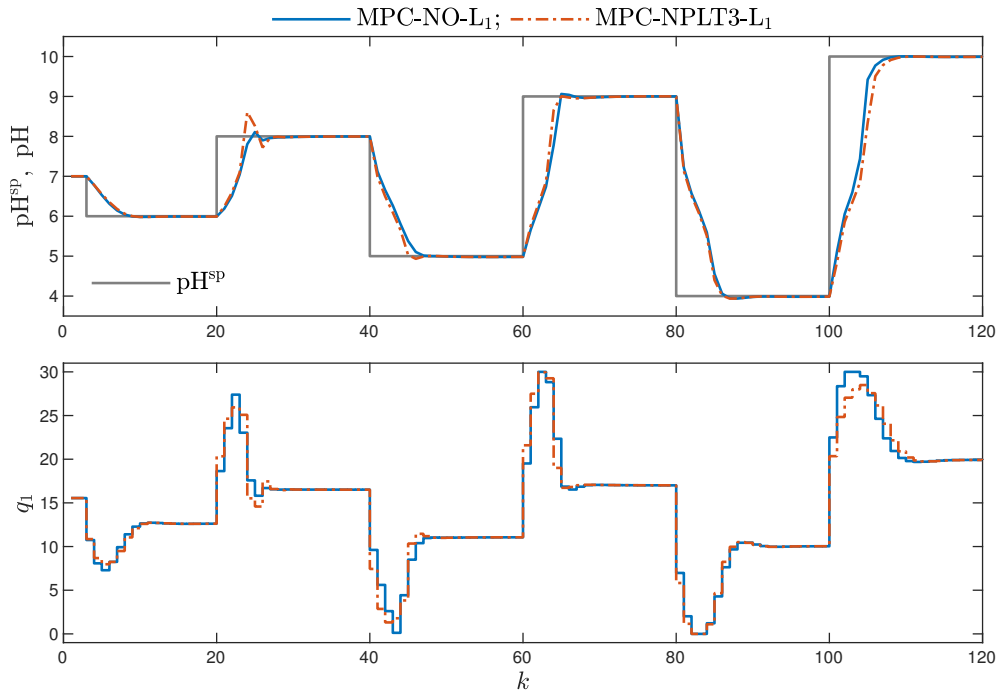

Figure 4. Simulation results: the MPC-NO-L_1 and MPC-NPLT3-L_1 algorithms.

Now, let us consider the MPC-NPLPT-L_1 algorithm, in which a few repetitions of trajectory linearisation and quadratic optimisation are possible at each sampling instant. The maximal allowed number of such repetitions is five. The stopping criteria (continuation and termination of the internal iterations) are defined by an additional parameter δ [29]. The internal iterations are continued if the sum of squared differences between the prediction of the model output and the required set-point is greater than δ. The internal iterations are terminated if the sum of squared differences between the optimal solution obtained in two consecutive internal iterations is lower than δ. In general, the lower the

parameter δ, the better control quality we expect to obtain, but the number of repetitions of the internal iterations increases. Figure 5 shows the obtained trajectories of the MPC-NPLPT-L_1 algorithm for three values of the parameter δ. For the considered neutralisation process, the obtained differences are small, but the best results are obtained for $\delta = 10^{-1}$, as it gives the best trajectory for the second set-point step. The considered neutralisation process is nonlinear. The process gain and its dynamics heavily depend on the current operating points. Hence, for different operating points, different overshoot and settling times are observed. Figure 6 depicts the number of internal iterations in the consecutive sampling instants of the MPC-NPLPT-L_1 algorithm for three values of the parameter δ. More than one internal iteration is necessary when the set-point changes; when the process is close to the required operating point, one internal iteration is sufficient. It is clear that the lower the parameter δ, the more internal iterations are necessary. This is true provided that a perfect model is used in MPC. In reality, the model is an approximation of the process. In our case, the unavoidable process–model mismatch results in the lowest overshoot for $\delta = 10^{-1}$.

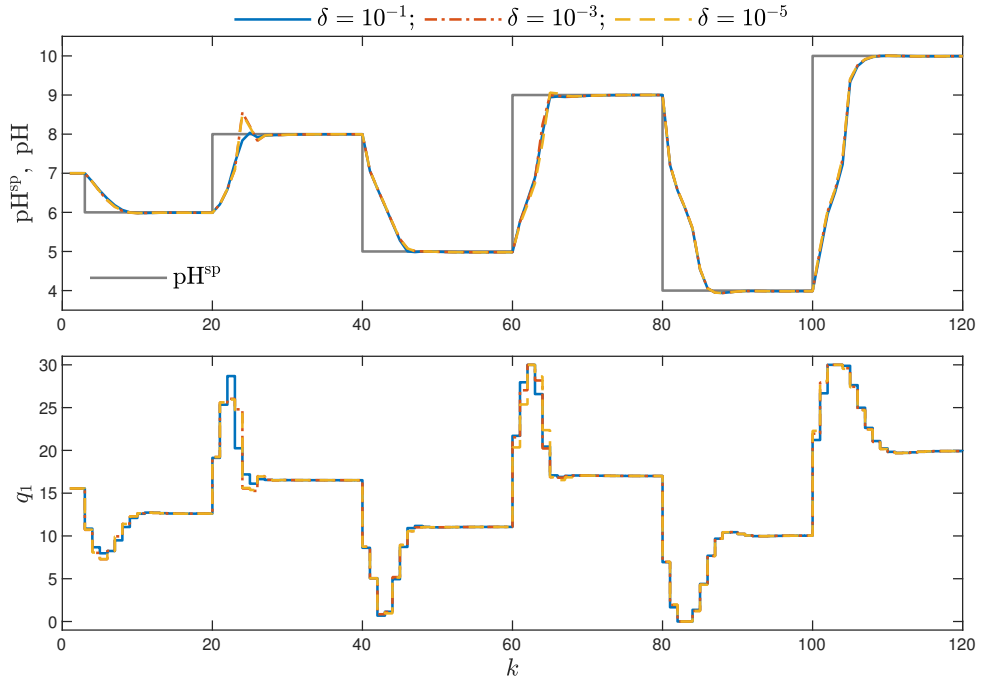

Figure 5. Simulation results: the MPC-NPLPT-L_1 algorithm for three values of the parameter δ.

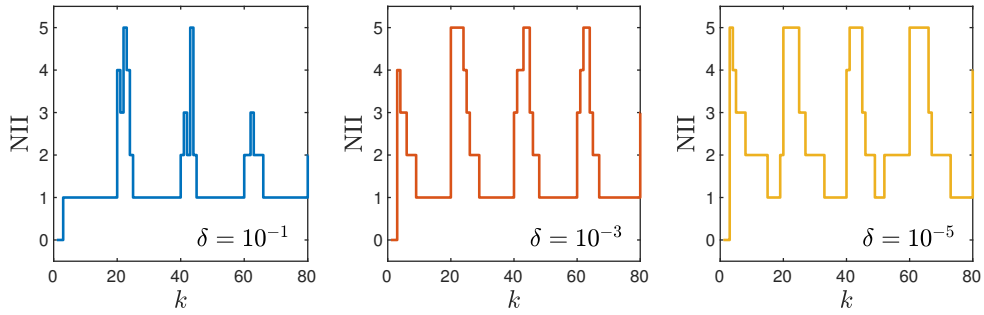

Figure 6. Simulation results: the number of internal iterations (NII) in the consecutive sampling instants of the MPC-NPLPT-L_1 algorithm for three values of the parameter δ.

Remembering our objective the most important issue is to compare the trajectories of the reference MPC-NO-L_1 algorithm and the computationally efficient MPC-NPLPT-L_1 scheme in the latter one, $\delta = 10^{-1}$. Figure 7 shows the obtained trajectories. It is very important to note that the recommended MPC-NPLPT-L_1 algorithm with on-line trajectory

linearisation and quadratic optimisation gives very similar, practically the same, trajectories as the algorithm with nonlinear optimisation. It is necessary to recall Figure 4, which compares the performance of the MPC-NPLT3-L_1 algorithm in which linearisation and optimisation are performed only once at each sampling instant. Comparing Figures 4 and 7, it is clear that multiple linearisation and optimisation possible at each sampling instant in the MPC-NPLPT-L_1 approach is really beneficial for the considered process.

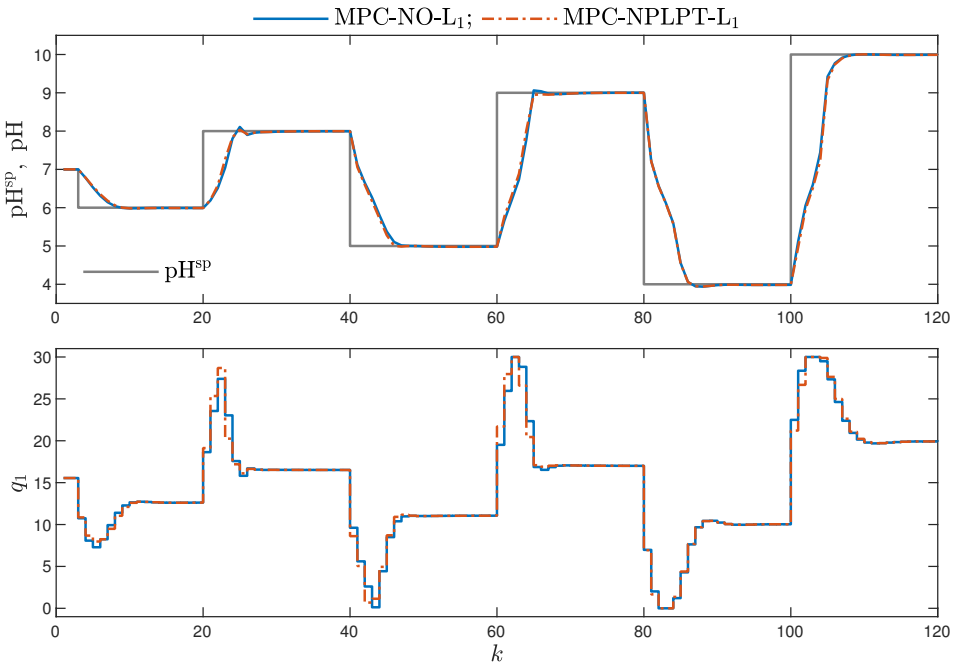

Figure 7. Simulation results: the MPC-NO-L_1 algorithm vs. MPC-NPLPT-L_1 algorithm.

In Figure 8, the recommended MPC-NPLPT-L_1 algorithm, which uses the L_1 norm, is compared with its counterpart, MPC-NPLPT-L_2, in which the classical L_2 norm is used. The use of the L_1 norm leads to a lower overshoot and shorter setting time.

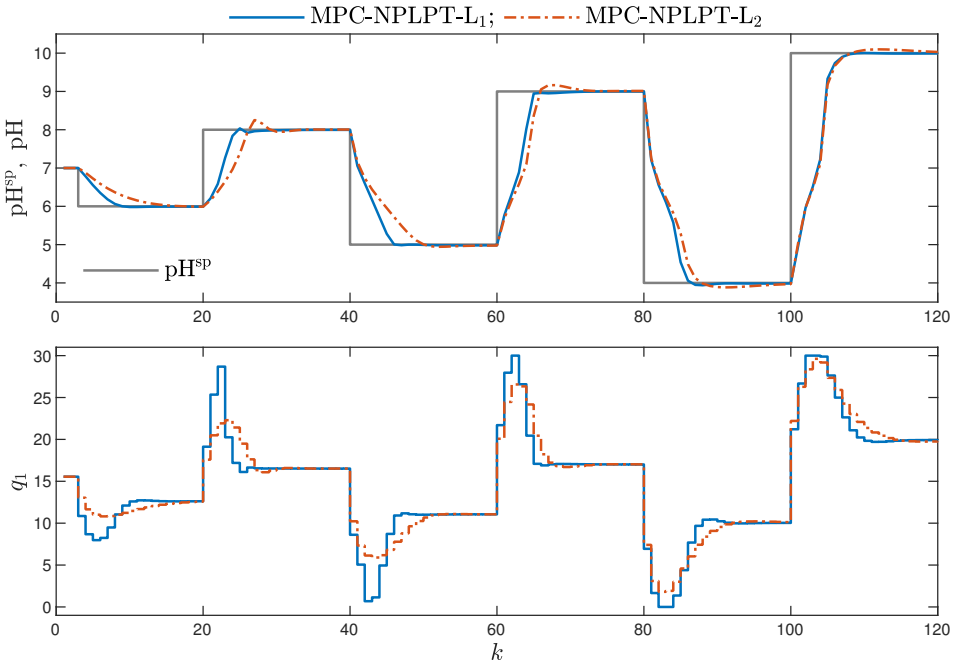

Figure 8. Simulation results: the MPC-NPLPT-L_1 algorithm vs. MPC-NPLPT-L_2 algorithm.

Having visually compared the obtained trajectories of the discussed MPC algorithms, it is interesting to analyse their performance and differences using some numerical control quality indicators [70]. We consider the following indices:

- The sum of absolute values of control errors for the whole simulation horizon defined as:

$$E_1 = \sum_{k=0}^{120} |y^{\mathrm{sp}}(k) - y(k)| \tag{50}$$

  where $y(k)$ denotes the real value of the process output obtained in simulation.
- The sum of absolute values of differences between the output of the process when it is controlled by the "ideal" MPC-NO-L_1 algorithm ($y^{\text{MPC-NO-L}_1}(k)$) and the output of the process when it is controlled by a compared MPC scheme ($y(k)$). These differences are considered for the whole simulation horizon:

$$E_1^{\text{MPC-NO-L}_1} = \sum_{k=0}^{120} |y^{\text{MPC-NO-L}_1}(k) - y(k)| \tag{51}$$

- The sum of squared control errors for the whole simulation horizon defined as:

$$E_2 = \sum_{k=0}^{120} (y^{\mathrm{sp}}(k) - y(k))^2 \tag{52}$$

- The sum of squared differences between the output of the process when it is controlled by the "ideal" MPC-NO-L_1 algorithm and the output of the process when it is controlled by a compared MPC scheme. These differences are considered for the whole simulation horizon:

$$E_2^{\text{MPC-NO-L}_1} = \sum_{k=0}^{120} (y^{\text{MPC-NO-L}_1}(k) - y(k))^2 \tag{53}$$

It is important to stress the fact that the performance criterion E_1 is not directly minimised in MPC algorithms that use the L_1 cost function J_1 (Equation (4)). This is because the actually minimised cost function J_1 takes into account the absolute value of control errors predicted over the prediction horizon. The horizon is shifted at each sampling instant. Moreover, it takes into account the second part of the cost function in which the sum of squared moves of the manipulated variable of the control horizon is penalised. Similarly, the performance criterion E_2 is not directly minimised in MPC algorithms that use the L_2 cost function J_2 (Equation (3)).

Let us state our expectations and objectives. When the norm L_1 is used in MPC, we expect to obtain the lowest value of the indices E_1 and $E_1^{\text{MPC-NO-L}_1}$. The more similar the obtained trajectories are to those obtained in the MPC-NO-L_1 scheme, the closer the index $E_1^{\text{MPC-NO-L}_1}$ is to 0.

Table 1 presents numerical values of the control quality indices (50)–(53). The following algorithms are considered: MPC-NPLT1, MPC-NPLT2 and MPC-NPLT3 algorithms, three versions of the MPC-NPLPT scheme with different stopping criterion defined by the parameter δ and the MPC-NO approach. The results are divided into two parts: in the first one, the norm L_1 is used in MPC; in the second one, the norm L_2 is considered; the cost function type minimised in MPC is denoted in colour. In addition to the control quality indices, Table 1 also specifies the relative calculation time of all algorithms; the results are given in percentages in such a way that the calculation time for the most demanding algorithm, i.e., MPC-NO-L_1, is treated as 100%. Considering the obtained numerical results, we are able to formulate the following observations concerned with control quality:

1. Comparing the MPC algorithms with the norm L_1, in which one trajectory linearisation and quadratic optimisation are executed at each sampling instant, the best results

are obtained in the MPC-NPLT3-L_1 scheme, in which the trajectory linearisation is performed using an inverse static model of the process. That algorithm gives the lowest values of the performance indices E_1 and $E_1^{\text{MPC-NO-L}_1}$. It confirms the comparison given in Figure 3.

2. Better results are possible when a few repetitions of trajectory linearisation and quadratic optimisation are possible at each sampling instant in the MPC-NPLPT-L_1 scheme. The obtained value of the indices E_1 and $E_1^{\text{MPC-NO-L}_1}$ are lower. It confirms the comparison given in Figure 7. Moreover, the lower the parameter δ, the more similar the obtained trajectory is to that possible in the reference MPC-NO-L_1 scheme.
3. Bearing in mind our expectations and objectives, the classical MPC algorithms that use the L_2 norm give a worse performance. This confirms the comparison given in Figure 8. For the corresponding algorithms, the values of both E_1 and $E_1^{\text{MPC-NO-L}_1}$ performance indices are better (i.e., lower) when the norm L_1 is used; the norm L_2 gives worse results. This effect is best visible when we consider the $E_1^{\text{MPC-NO-L}_1}$ performance index. For example, comparing the MPC-NPLPT-L_1 and MPC-NPLPT-L_2 algorithms with $\delta = 10^{-5}$, that index is in the first case approximately 11 times lower.
4. It is very interesting that the use of the L_1 norm in place of the classical L_2 one leads to not only better (lower) values of the indices E_1 and $E_1^{\text{MPC-NO-L}_1}$, which is natural, but also makes it possible to reduce the indices E_2 and $E_2^{\text{MPC-NO-L}_1}$. For all pairs of algorithms (with L_1 and L_2 norms), the E_2 index is slightly lower when the L_1 norm is used. This difference is even more clear when we consider the $E_2^{\text{MPC-NO-L}_1}$ index. It confirms the comparison given in Figure 8.

Table 1. Simulation results: the values of the control quality criteria and the scaled calculation time, the cost function type corresponding to that minimised in MPC is denoted in color; in all cases $\lambda = 1$.

Algorithm	E_1	$E_1^{\text{MPC-NO-L}_1}$	E_2	$E_2^{\text{MPC-NO-L}_1}$	Calculation Time
MPC-NPLT1-L_1	7.8818×10^1	1.0431×10^1	2.2752×10^2	4.4144	34.0%
MPC-NPLT2-L_1	7.8897×10^1	1.5057×10^1	2.1985×10^2	9.3569	33.7%
MPC-NPLT3-L_1	7.1167×10^1	7.4706	2.2197×10^2	3.6258	33.6%
MPC-NPLPT-L_1, $\delta = 10^{-1}$	6.9590×10^1	2.7467	2.1626×10^2	3.3734×10^{-1}	40.3%
MPC-NPLPT-L_1, $\delta = 10^{-3}$	6.9768×10^1	2.6469	2.1435×10^2	8.7502×10^{-1}	49.3%
MPC-NPLPT-L_1, $\delta = 10^{-5}$	7.0350×10^1	1.5524	2.1573×10^2	5.4643×10^{-1}	60.8%
MPC-NO-L_1	7.0371×10^1	–	2.1631×10^2	–	100.0%
MPC-NPLT1-L_2	8.6977×10^1	1.8184×10^1	2.3746×10^2	8.9869	21.0%
MPC-NPLT2-L_2	8.3845×10^1	1.6043×10^1	2.2758×10^2	5.8777	20.8%
MPC-NPLT3-L_2	8.3647×10^1	1.5071×10^1	2.3459×10^2	6.2455	21.0%
MPC-NPLPT-L_2, $\delta = 10^{-1}$	8.3784×10^1	1.4916×10^1	2.3559×10^2	5.5591	23.6%
MPC-NPLPT-L_2, $\delta = 10^{-3}$	8.4162×10^1	1.6693×10^1	2.2885×10^2	6.6640	30.9%
MPC-NPLPT-L_2, $\delta = 10^{-5}$	8.4795×10^1	1.7317×10^1	2.2976×10^2	7.2166	35.6%
MPC-NO-L_2	8.5089×10^1	1.7708×10^1	2.3033×10^2	7.6448	73.3%

Furthermore, it is also possible to put forward the following observations concerned with computation time:

1. In general, all MPC algorithms with the norm L_1 are more computationally demanding than their counterparts that use the norm L_2. This is because, in the first case, in all calculations, i.e., in prediction, linearisation and optimisation, the neural approximator determines the absolute values of the control errors over the prediction horizon whereas, in the second case, no approximator is used, the predictions and control errors are used directly in all calculations.
2. All MPC algorithms with linearisation and quadratic optimisation are less computationally demanding than the reference "ideal" MPC-NO algorithm.

3. The more complicated the trajectory linearisation, the longer the calculation time. The lowest calculation time is observed in the MPC-NPLT1, MPC-NPLT2 and MPC-NPLT3 algorithms, with one repetition of linearisation and quadratic optimisation at each sampling instant. The calculation time becomes longer in the MPC-NPLPT scheme, with a few repetitions of linearisation and optimisation at each instant; the lower the parameter δ, the longer the calculation time.

Finally, let us compare the results of the MPC-NPLPT-L_1 and MPC-NPLPT-L_2 algorithms when the penalty coefficient is increased. In our study, the value $\lambda = 5$ is considered. Figure 9 presents the obtained trajectories. Two top panels show the results for the whole simulation scenario, whereas the bottom ones show enlarged fragments for three set-point changes. It is interesting to see an additional advantage of the recommended L_1 norm because, in such a case, the overshoot is much lower and the setting time is much shorter when compared with the trajectories possible when the classical L_2 norm is used. In the considered comparison, for $\lambda = 5$, the benefits of using the L_1 norm are even more clear than in the case of the default value of the penalty coefficient, i.e., $\lambda = 1$, as presented in Figure 8.

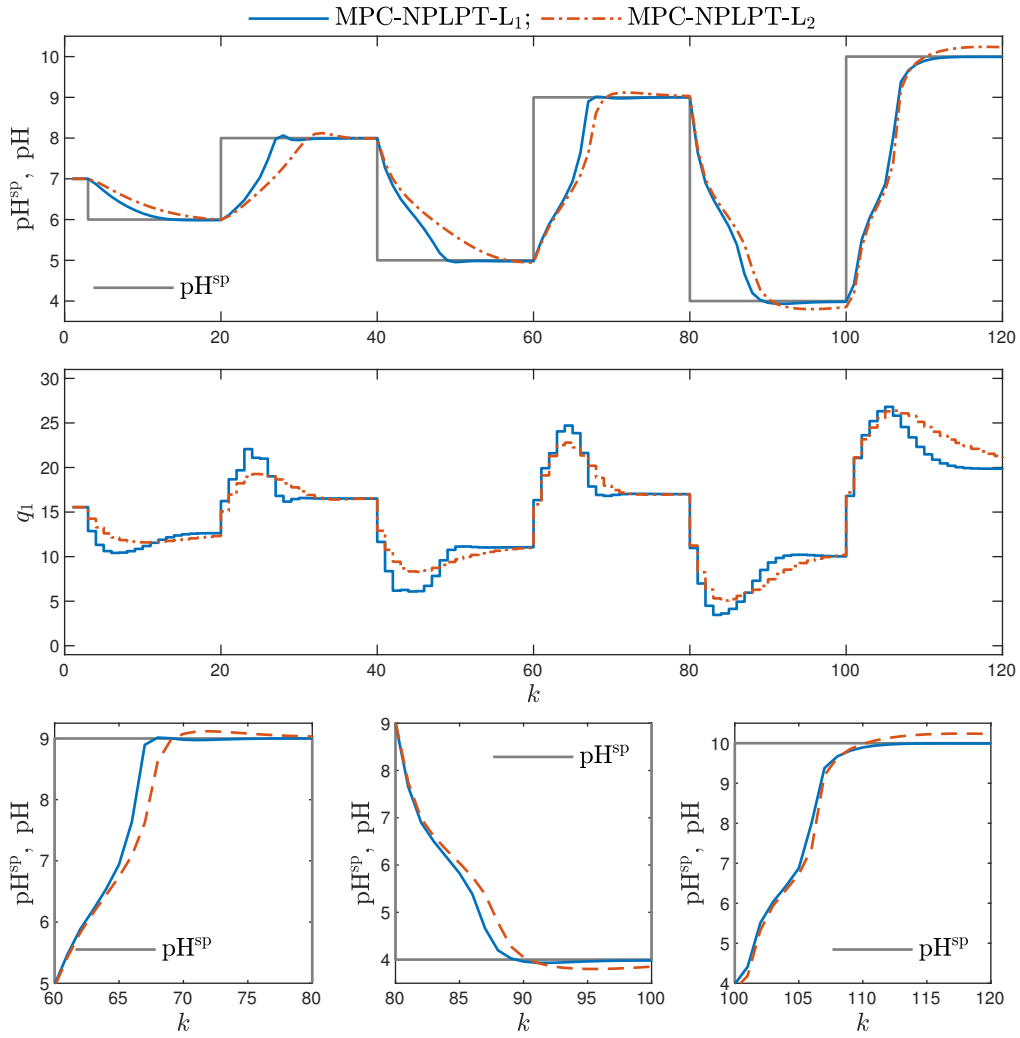

Figure 9. Simulation results: the MPC-NPLPT-L_1 algorithm vs. MPC-NPLPT-L_2 algorithm, in both cases $\lambda = 5$; two top panels show the results for the whole simulation horizon, three bottom panels show enlarged fragments for the sampling instants $60 \leq k \leq 80$, $80 \leq k \leq 100$ and $100 \leq k \leq 120$.

5. Conclusions

The presented MPC-L_1 algorithms have three essential advantages. Firstly, thanks to using a neural differentiable approximator of the non-differentiable absolute value function and on-line advanced trajectory linearisation, computationally simple quadratic optimisation is used in place of demanding nonlinear optimisation. Secondly, the obtained trajectories are very similar, practically the same, as those possible in the reference scheme with nonlinear optimisation. Thirdly, it is shown that the use of the recommended L_1 norm gives better results defined using different control quality criteria, such as the sum of absolute values or squared control errors, overshoot and setting time. Furthermore, it must be stressed that the L_1 norm even gives better results than the classical L_2 one in terms of the classical control performance indicator that measures squared control errors. The discussed MPC is very universal, it may be used in industrial process control applications and in fast embedded systems. The only condition is that the model used for prediction and the neural approximator must be differentiable.

As future works, the following issues are worth investigating: taking into account not only constraints imposed on the manipulated variable, but also on the predicted controlled one, the development of nonlinear MPC-L_1 algorithms for multivariable processes with multiple inputs and multiple outputs, considering state–space process description in place of the input–output one and considering alternative types of the cost function, not only the discussed L_1 norm.

Author Contributions: Conceptualisation, M.Ł. and R.N.; methodology, M.Ł. and R.N.; software, M.Ł. and R.N.; validation, M.Ł. and R.N.; formal analysis, M.Ł. and R.N.; investigation, M.Ł. and R.N.; writing–original draft preparation, M.Ł. and R.N.; writing–review and editing, M.Ł. and R.N.; visualisation, M.Ł.; supervision, M.Ł. Both authors have read and agreed to the published version of the manuscript.

Funding: This research was financed by Warsaw University of Technology in the framework of the project for the scientific discipline automatic control, electronics and electrical engineering.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

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

References

1. Tatjewski, P. *Advanced Control of Industrial Processes, Structures and Algorithms*; Springer: London, UK, 2007.
2. Maciejowski, J. *Predictive Control with Constraints*; Prentice Hall: Harlow, UK, 2002.
3. Nebeluk, R.; Marusak, P.M. Efficient MPC algorithms with variable trajectories of parameters weighting predicted control errors. *Arch. Control Sci.* **2020**, *30*, 325–363.
4. Huyck, B.; De Brabanter, J.; De Moor, B.; Van Impe, J.F.; Logist, F. Online model predictive control of industrial processes using low level control hardware: A pilot-scale distillation column case study. *Control Eng. Pract.* **2014**, *28*, 34–48. [CrossRef]
5. Ogonowski, S.; Bismor, D.; Ogonowski, Z. Control of complex dynamic nonlinear loading process for electromagnetic mill. *Arch. Control Sci.* **2020**, *30*, 471–500.
6. Zarzycki, K.; Ławryńczuk, M. Fast real-time model predictive control for a ball-on-plate process. *Sensors* **2021**, *21*, 3959. [CrossRef]
7. Horla, D. Experimental Results on Actuator/Sensor Failures in Adaptive GPC Position Control. *Actuators* **2021**, *10*, 43. [CrossRef]
8. Eskandarpour, A.; Sharf, I. A constrained error-based MPC for path following of quadrotor with stability analysis. *Nonlinear Dyn.* **2020**, *98*, 899–918. [CrossRef]
9. Ducajú, S.; Salt Llobregat, J.J.; Cuenca, Á.; Tomizuka, M. Autonomous Ground Vehicle Lane-Keeping LPV Model-Based Control: Dual-Rate State Estimation and Comparison of Different Real-Time Control Strategies. *Sensors* **2021**, *21*, 1531.
10. Liang, Y.; Yin, Z.; Nie, L. Shared Steering Control for Lane Keeping and Obstacle Avoidance Based on Multi-Objective MPC. *Sensors* **2021**, *21*, 4671. [CrossRef] [PubMed]
11. Patria, D.; Rossi, C.; Fernandez, R.A.S.; Dominguez, S. Nonlinear control strategies for an autonomous wing-in-ground-effect vehicle. *Sensors* **2021**, *21*, 4193. [CrossRef] [PubMed]
12. Bassolillo, S.R.; D'Amato, E.; Notaro, I.; Blasi, L.; Mattei, M. Decentralized Mesh-Based Model Predictive Control for Swarms of UAVs. *Sensors* **2020**, *20*, 4324. [CrossRef]

13. Bania, P. An information based approach to stochastic control problems. *Int. J. Appl. Math. Comput. Sci.* **2020**, *30*, 47–59.
14. Ding, Z.; Sun, C.; Zhou, M.; Liu, Z.; Wu, C. Intersection Vehicle Turning Control for Fully Autonomous Driving Scenarios. *Sensors* **2021**, *21*, 3995. [CrossRef]
15. Xiong, L.; Fu, Z.; Zeng, D.; Leng, B. An Optimized Trajectory Planner and Motion Controller Framework for Autonomous Driving in Unstructured Environments. *Sensors* **2021**, *21*, 4409. [CrossRef]
16. Simon, D. *Optimal State Estimation: Kalman, H_∞, and Nonlinear Approaches*; John Wiley & Sons: Hoboken, NJ, USA, 2006.
17. Bououden, S.; Boulkaibet, I.; Chadli, M.; Abboudi, A. A Robust Fault-Tolerant Predictive Control for Discrete-Time Linear Systems Subject to Sensor and Actuator Faults. *Sensors* **2021**, *21*, 2307. [CrossRef]
18. Karimshoushtari, M.; Novara, C.; Tango, F. How Imitation Learning and Human Factors Can Be Combined in a Model Predictive Control Algorithm for Adaptive Motion Planning and Control. *Sensors* **2021**, *21*, 4012. [CrossRef] [PubMed]
19. Miller, A.; Rybczak, M.; Rak, A. Towards the Autonomy: Control Systems for the Ship in Confined and Open Waters. *Sensors* **2021**, *21*, 2286. [CrossRef] [PubMed]
20. Yao, F.; Yang, C.; Liu, X.; Zhang, M. Experimental Evaluation on Depth Control Using Improved Model Predictive Control for Autonomous Underwater Vehicle (AUVs). *Sensors* **2018**, *18*, 2321. [CrossRef]
21. Bacci di Capaci, R.; Vaccari, M.; Pannocchia, G. Model predictive control design for multivariable processes in the presence of valve stiction. *J. Process Control* **2018**, *71*, 25–34. [CrossRef]
22. Dötlinger, A.; Kennel, R.M. Near time-optimal model predictive control using an L_1-norm based cost functional. In Proceedings of the 2012 IEEE Congress on Evolutionary Computation, Pittsburgh, PA, USA, 14–18 September 2014; pp. 3504–3511.
23. Domański, P.; Ławryńczuk, M. Impact of MPC embedded performance index on control quality. *IEEE Access* **2021**, *9*, 24787–24795. [CrossRef]
24. Fehér, M.; Straka, O.; Šmídl, V. Model predictive control of electric drive system with L_1-norm. *Eur. J. Control* **2020**, *56*, 242–253. [CrossRef]
25. Karamanakos, P.; Geyer, T.; Kennel, R. On the choice of norm in finite control set model predictive control. *IEEE Trans. Power Electron.* **2018**, *33*, 7105–7117. [CrossRef]
26. Müller, M.; Worthmann, K. Quadratic costs do not always work in MPC. *Automatica* **2017**, *82*, 269–277. [CrossRef]
27. Brunton, S.L.; Kutz, J.N. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*; Cambridge University Press: Cambridge, UK, 2017.
28. Boiroux, D.; Jørgensen, J.B. Sequential ℓ_1 quadratic programming for nonlinear model predictive control. *IFAC-PapersOnLine* **2019**, *52*, 474–479. [CrossRef]
29. Ławryńczuk, M. Computationally Efficient Model Predictive Control Algorithms: A Neural Network Approach. In *Studies in Systems, Decision and Control*; Springer: Cham, Switzerland, 2014; Volume 3.
30. Wu, X.; Shen, J.; Li, Y.; Wang, M.; Lawal, A. Flexible operation of post-combustion solvent-based carbon capture for coal-fired power plants using multi-model predictive control: A simulation study. *Fuel* **2018**, *220*, 931–941. [CrossRef]
31. Marusak, P.M. A numerically efficient fuzzy MPC algorithm with fast generation of the control signal. *Int. J. Appl. Math. Comput. Sci.* **2021**, *31*, 59–71.
32. Kittisupakorn, P.; Thitiyasook, P.; Hussain, M.A.; Daosud, W. Neural network based model predictive control for a steel pickling process. *J. Process Control* **2009**, *19*, 579–590. [CrossRef]
33. Reynolds, J.; Rezgui, Y.; Kwan, A.; Piriou, S. A zone-level, building energy optimisation combining an artificial neural network, a genetic algorithm, and model predictive control. *Energy* **2018**, *151*, 729–739. [CrossRef]
34. Ardabili, S.F.; Mahmoudi, A.; Gundoshmian, T.M. Modeling and simulation controlling system of HVAC using fuzzy and predictive (radial basis function, RBF) controllers. *J. Build. Eng.* **2016**, *6*, 301–308. [CrossRef]
35. Han, H.G.; Qiao, J.F.; Chen, Q.L. Model predictive control of dissolved oxygen concentration based on a self-organizing RBF neural network. *Control Eng. Pract.* **2012**, *20*, 465–476. [CrossRef]
36. Huang, C.; Li, L.; Wang, X. Extended Model Predictive Control Based on Multi-Structure RBF Networks: Design and Application to Clutch Control. *IFAC-PapersOnLine* **2018**, *51*, 653–658. [CrossRef]
37. Jeon, B.K.; Kim, E.J. LSTM-based model predictive control for optimal temperature set-point planning. *Sustainability* **2021**, *13*, 894. [CrossRef]
38. Karimanzira, D.; Rauschenbach, T. Deep learning based model predictive control for a reverse osmosis desalination plant. *J. Appl. Math. Phys.* **2020**, *8*, 2713–2731. [CrossRef]
39. Zarzycki, K.; Ławryńczuk, M. LSTM and GRU neural networks as models of dynamical processes used in predictive control: A comparison of models developed for two chemical reactors. *Sensors* **2021**, *21*, 5625. [CrossRef]
40. Sabzevari, S.; Heydari, R.; Mohiti, M.; Savaghebi, M.; Rodriguez, J. Model-free neural network-based predictive control for robust operation of power converters. *Energies* **2021**, *14*, 2325. [CrossRef]
41. Zamarreno, J.; Vega, P.; Garcia, L.D.; Francisco, M. State-space neural network for modelling, prediction and control. *Control Eng. Pract.* **2000**, *8*, 1063–1075. [CrossRef]
42. Cervantes-Bobadilla, M.; Escobar-Jimenez, R.F.; Gomez-Aguilar, J.F.; Garcia-Morales, J.; Olivares-Peregrino, V.H. Experimental study on the performance of controllers for the hydrogen gas production demanded by an internal combustion engine. *Energies* **2018**, *11*, 2157. [CrossRef]

43. Huo, H.B.; Zhu, X.J.; Hu, W.Q.; Tu, H.Y.; Li, J.; Yang, J. Nonlinear model predictive control of SOFC based on a Hammerstein model. *J. Power Sources* **2008**, *185*, 338–344. [CrossRef]
44. Ławryńczuk, M. Suboptimal nonlinear predictive control based on multivariable neural Hammerstein models. *Appl. Intell.* **2016**, *32*, 173–192. [CrossRef]
45. Arefi, M.A.; Montazeri, A.; Poshtan, J.; Jahed-Motlagh, M.R. Wiener-neural identification and predictive control of a more realistic plug-flow tubular reactor. *Chem. Eng. J.* **2008**, *138*, 274–282. [CrossRef]
46. Li, S.; Li, Y. Model predictive control of an intensified continuous reactor using a neural network Wiener model. *Neurocomputing* **2016**, *185*, 93–104. [CrossRef]
47. Ławryńczuk, M. Nonlinear Predictive Control Using Wiener Models: Computationally Efficient Approaches for Polynomial and Neural Structures. In *Studies in Systems, Decision and Control*; Springer: Cham, Switzerland, 2022; Volume 389.
48. Peng, H.; Wu, J.; noussa, G.; Deng, Q.; Nakano, K. Nonlinear system modeling and predictive control using the RBF nets-based quasi-linear ARX model. *Control Eng. Pract.* **2009**, *17*, 59–66. [CrossRef]
49. Peng, H.; Ozaki, T.; Toyoda, Y.; Shioya, H.; Nakano, K.; Haggan-Ozaki, V.; Mori, M. RBF-ARX model-based nonlinear system modeling and predictive control with application to a NOx decomposition process. *Control Eng. Pract.* **2004**, *12*, 191–203. [CrossRef]
50. Ławryńczuk, M. Neural Dynamic Matrix Control algorithm with disturbance compensation. In Proceedings of the 23th International Conference on Industrial, Engineering & Other Applications of Applied Intelligent Systems (IEA-AIE 2010), Cordoba, Spain, 1–4 June 2010; García Pedrajas, N., Herrera, F., Fyfe, C., Benítez, J.M., Ali, M., Eds.; Lecture Notes in Artificial Intelligence; Springer: Berlin, Germany, 2010; Volume 6098, pp. 52–61.
51. Doncevic, D.T.; Schweidtmann, A.M.; Vaupel, Y.; Schäfer, P.; Caspari, A.; Mitsos, A. Deterministic global nonlinear model predictive control with recurrent neural networks embedded. *IFAC-PapersOnLine* **2020**, *53*, 5273–5278. [CrossRef]
52. Ławryńczuk, M.; Tatjewski, P. Nonlinear predictive control based on neural multi-models. *Int. J. Appl. Math. Comput. Sci.* **2010**, *20*, 7–21. [CrossRef]
53. Hosen, M.A.; Hussain, M.A.; Mjalli, F.S. Control of polystyrene batch reactors using neural network based model predictive control (NNMPC): An experimental investigation. *Control Eng. Pract.* **2011**, *19*, 454–467. [CrossRef]
54. Aggelogiannaki, E.; Sarimveis, H. Nonlinear model predictive control for distributed parameter systems using data driven artificial neural network models. *Comput. Chem. Eng.* **2008**, *32*, 1225–1237. [CrossRef]
55. Aggelogiannaki, E.; Sarimveis, H.; Koubogiannis, D. Model predictive temperature control in long ducts by means of a neural network approximation tool. *Appl. Therm. Eng.* **2007**, *27*, 2363–2369. [CrossRef]
56. Xie, W.; Bonis, I.; Theodoropoulos, C. Data-driven model reduction-based nonlinear MPC for large-scale distributed parameter systems. *J. Process Control* **2015**, *35*, 50–58. [CrossRef]
57. Tang, Y.; Han, Z.; Liu, F.; Guan, X. Identification and control of nonlinear system based on Laguerre-ELM Wiener model. *Commun. Nonlinear Sci. Numer. Simul.* **2016**, *38*, 192–205. [CrossRef]
58. Stogiannos, M.; Alexandridis, A.; Sarimveis, H. Model predictive control for systems with fast dynamics using inverse neural models. *ISA Trans.* **2018**, *72*, 161–177. [CrossRef] [PubMed]
59. Vaupel, Y.; Hamacher, N.C.; Caspari, A.; Mhamdi, A.; Kevrekidis, I.G.; Mitsos, A. Accelerating nonlinear model predictive control through machine learning. *J. Process Control* **2020**, *92*, 261–270. [CrossRef]
60. Bonzanini, A.D.; Paulson, J.A.; Makrygiorgos, G.; Mesbah, A. Fast approximate learning-based multistage nonlinear model predictive control using Gaussian processes and deep neural networks. *Comput. Chem. Eng.* **2021**, *145*, 107174. [CrossRef]
61. Ławryńczuk, M. Explicit nonlinear predictive control algorithms with neural approximation. *Neurocomputing* **2014**, *129*, 570–584. [CrossRef]
62. Maddalena, E.T.; da S. Moraes, C.G.; Waltrich, G.; Jones, C.N. A neural network architecture to learn explicit MPC controllers from data. *IFAC-PapersOnLine* **2020**, *53*, 11362–11367. [CrossRef]
63. Pan, Y.; Wang, J. Two neural network approaches to model predictive control. In Proceedings of the American Control Conference (ACC2008); Seattle, WA, USA, 11–13 June 2008 pp. 1685–1690.
64. Xu, J.; Li, C.; He, X.; Huang, T. Recurrent neural network for solving model predictive control problem in application of four-tank benchmark. *Neurocomputing* **2016**, *190*, 172–178. [CrossRef]
65. Haykin, S. *Neural Networks and Learning Machines*; Pearson Education: Upper Saddle River, NJ, USA, 2009.
66. Gómez, J.C.; Jutan, A.; Baeyens, E. Wiener model identification and predictive control of a pH neutralisation process. *IEE Proc. Control Theory Appl.* **2004**, *151*, 329–338. [CrossRef]
67. Janczak, A.; Korbicz, J. Two-stage instrumental variables identification of polynomial Wiener systems with invertible nonlinearities. *Int. J. Appl. Math. Comput. Sci.* **2019**, *29*, 571–580. [CrossRef]
68. Ławryńczuk, M. Practical nonlinear predictive control algorithms for neural Wiener models. *J. Process Control* **2013**, *23*, 696–714. [CrossRef]
69. Ławryńczuk, M. Modelling and predictive control of a neutralisation reactor using sparse Support Vector Machine Wiener models. *Neurocomputing* **2016**, *205*, 311–328. [CrossRef]
70. Domański, P. Control Performance Assessment: Theoretical Analyses and Industrial Practice. In *Studies in Systems, Decision and Control*; Springer: Cham, Switzerland, 2020; Volume 245.