Fast adaptive regression-based model predictive control

Eslam Mostafa · Hussein A. Aly · Ahmed Elliethy

Received: 8 October 2022 / Revised: 22 March 2023 / Accepted: 28 April 2023 / Published online: 25 September 2023

© The Author(s), under exclusive licence to South China University of Technology and Academy of Mathematics and Systems Science, Chinese Academy of Sciences 2023

Abstract
Model predictive control (MPC) is an optimal control method that predicts the future states of the system being controlled and estimates the optimal control inputs that drive the predicted states to the required reference. The computations of the MPC are performed at pre-determined sample instances over a finite time horizon. The number of sample instances and the horizon length determine the performance of the MPC and its computational cost. A long horizon with a large sample count allows the MPC to better estimate the inputs when the states have rapid changes over time, which results in better performance but at the expense of high computational cost. However, this long horizon is not always necessary, especially for slowly-varying states. In this case, a short horizon with less sample count is preferable as the same MPC performance can be obtained but at a fraction of the computational cost. In this paper, we propose an adaptive regression-based MPC that predicts the best minimum horizon length and the sample count from several features extracted from the time-varying changes of the states. The proposed technique builds a synthetic dataset using the system model and utilizes the dataset to train a support vector regressor that performs the prediction. The proposed technique is experimentally compared with several state-of-the-art techniques on both linear and non-linear models. The proposed technique shows a superior reduction in computational time with a reduction of about 35–65% compared with the other techniques without introducing a noticeable loss in performance.

Keywords Regression analysis · MPC · Control · Parametrization · Wavelet · Support vector regression (SVR) · Optimization

1 Introduction
Model predictive control (MPC) [1] is an advanced control method that has been widely used in many applications [2–6]. Given a discrete system with its states defined at specific sampling instances, the MPC utilizes the mathematical model of the system to predict its future states at each sample instant over a finite future horizon time. The predicted states along with a set of given system constraints are used to formulate an optimization problem that is solved online in every control cycle to estimate the optimal control inputs at each sample instant over the horizon. Only the estimated input at the first sample instant is applied to the system, and the MPC repeats the same process for all subsequent control cycles. For a long horizon with a large number of sample instances, the predicted behavior of the system becomes more intimate to the required reference [7], i.e., better control performance. However, for such a long horizon, the MPC easily may not finalize the computations involved in solving the online optimization problem within the control cycle, and this results in a lagging control input to the system in this case.

To avoid the aforementioned lagging control input problem, several studies in the literature focus on speeding up the MPC by reducing its online computations. For example, several techniques try to minimize the number of sample instances within the horizon. In [8–10], the input and state trajectories, that represent their variations over time, are parametrized with some basis functions. The parameterization of the trajectories reduces the degrees of freedom of the online optimization problem by calculating the control inputs only at specific sampling instances in the horizon while evaluating the rest using the parametrized version. In [11–13], non-uniformly spaced sample instances are used such that smaller intervals between sample instances are used with the near future of the horizon while larger ones are used with the
distant future. These techniques use a fixed horizon but keep the number of sample instances relatively small. However, using a fixed horizon is not optimal in all control scenarios. As shown in [14], a short horizon is enough for controlling a vehicle on highways where speed fluctuations are not fast, while a longer horizon is needed when driving in a city due to higher fluctuations in speed and environmental variables. Following this, the technique in [15] adaptively changes the horizon according to the curvature value for the state trajectories. However, the technique uses a heuristic rule to determine the horizon length, which results in a sub-optimal horizon. In [16], a variable horizon MPC is achieved by defining several fixed-horizon optimization problems with different horizon lengths. The overall complexity is relatively reduced by utilizing the time-varying move blocking technique [17] which fixes adjacent in-time decision variables of the optimization problem or its derivatives to be constant over several control cycles, and thus it results in sub-optimal control performance.

In [18], the horizon length is added as an extra degree of freedom to the MPC formulation, and thus it incurs additional computations in the control cycle. The technique in [19] deals with systems with non-linear dynamics and incrementally adjusts the horizon length to its minimum possible value that guarantees stabilization. This incremental adjusting for the horizon is practically not suitable for fast applications.

Another line of research is the so-called explicit MPC [20–22], which pre-computes the optimal control inputs offline as a function of the current state and reference states. Thus, the online optimization reduces to a simple search within the pre-computed values. The major drawback here is the searching time for the optimal solution which quickly increases when the number of states, horizon length, or the number of control inputs increases. Thus the explicit MPC can only be applied to small problems. This drawback is partially mitigated in [23] which uses a partial enumeration technique that offline computes and stores in a table the optimal control inputs for only frequently occurring constraint sets. The table is searched online for the best control and updated to incorporate new constraint sets as the control progresses in time.

Recently, machine learning techniques are employed for horizon prediction. In [24, 25], a novel technique is proposed that uses reinforcement learning (RL) [26] to predict the optimal horizon length using a policy function of the current state. The policy of the RL is modeled as a neural network (NN) [27] which is trained using the data collected during the operation of the system being controlled. The training of the NN is performed online within each control cycle, which adds more computations within the cycle. Another technique in [28] trains a NN offline that is used to predict the optimal horizon at run-time. However, these techniques perform the prediction solely based on the instantaneous state of the system, without any consideration for the future values of the states. Moreover, the techniques do not employ any feature engineering for training the NN, and thus it easily over-fits the training data. Additionally, the technique in [24] predicts the horizon length only while the technique in [28] uses the move-blocking strategy [17] that fixes the ratio between the number of sample instances and the time span of the predicted horizon. Therefore, the predictions performed by these techniques are not optimal in general.

In this paper, our goal is to accurately estimate both the best minimum horizon length and the number of samples without introducing a noticeable loss in the performance of the MPC. To this goal, we first propose a mathematical formulation that relates the horizon length and the sample count with the performance of the MPC, then we propose an efficient solution for it. Specifically, we propose an adaptive regression-based MPC (ARMPC) that predicts the best minimum horizon length and the sample count according to the current and future variations exhibited in the reference states of the model. To train the regression, we build a dataset by extracting several features that capture the variations of the reference trajectories of the model over future time with the associated best horizon and sample count in each situation. In run time, we extract the same set of features which presented to the regressor to predict the best minimum horizon length and the sample count.

Compared with previous techniques, our proposed ARMPC has several advantages. First, it estimates both the best horizon length and the best number of samples on the horizon, and this allows the proposed technique to provide more reduction in computations. Second, our technique does not rely on raw values of the reference states but employs feature engineering to extract several distinctive features from the reference trajectories, and thus it avoids over-fitting in the learning phase. Finally, these features are extracted not from the instantaneous values of the states but over a future span of the horizon, which allows a more accurate estimate for the horizon and the sample count. These advantages are reflected in our experimental results where we compared the proposed ARMPC with three different state-of-the-art techniques on both linear and non-linear models. The proposed ARMPC shows a superior reduction in computational time with a reduction of about 35–65% compared with the other techniques without introducing a noticeable loss in performance. The source code of the proposed technique is available online.

The remainder of this paper is organized as follows. Section 2 presents a background for the MPC and briefly outlines its parametrization. Section 3 presents motivation examples and formulates our problem statement. Section 4 introduces the proposed adaptive regression-based MPC. Section 5 describes the experimental results that evaluate our proposed technique.

1 https://github.com/ahmed-elliethy/fast-regression-mpc.
Section 6 summarizes our conclusion and presents our future work.

2 MPC background

A continuous linear time-invariant system can be described in state space form as \([10]\)

\[ \dot{x}(t) = Ax(t) + Bu(t), \]  

where \( t \in \mathbb{R} \) is a continuous time instant, \( x(t) \in \mathbb{R}^{m \times 1} \) and \( u(t) \in \mathbb{R}^{n \times 1} \) represent a vector of states and a vector of system inputs at time \( t \), respectively, \( A \in \mathbb{R}^{m \times m} \) is the state matrix, and \( B \in \mathbb{R}^{m \times n} \) is the input matrix. The system can be discretized using any of the discretization methods (such as the Euler method) by

\[ x_k = \frac{x_{k+1} - x_k}{t_s}, \]

where \( k \in \mathbb{Z} \) is a discrete-time instant, \( t_s \) is the sampling time, and \( x_k = [x_k^1, \ldots, x_k^m]^T \) represents the vector of states at \( k \). With the discretization, (1) can be written as

\[ x_{k+1} = A_d x_k + B_d u_k, \]  

where \( u_k = [u_k^1, \ldots, u_k^n]^T \) represents the vector of system inputs at \( k \), \( A_d = I + t_s A \), \( B_d = t_s B \), and \( I \) is the identity matrix.

The goal of MPC is to estimate the optimal vector of system inputs for a fixed horizon length of \( T \) time steps in control cycles. More clearly, in the \( c \)th control cycle, the MPC estimates the optimal vector of system inputs from the time instant \( c \) to \( c + T - 1 \). Then only the first control input (at the time instant \( c \)) is applied to the system and new system states are predicted. In the next cycle, the MPC estimates the system inputs using the newly predicted states, then only the first input is applied again to the system. This loop will be repeated to find the optimum control input in every control cycle.

To do so, in the \( c \)th control cycle, the MPC first expresses the system states as a function of only the inputs and the initial state \( x_c \) in the cycle as

\[
\begin{bmatrix}
    x_{c+1} \\
    x_{c+2} \\
    \vdots \\
    x_{c+T}
\end{bmatrix}
= \begin{bmatrix}
    A_d x_c + B_d u_c \\
    A_d x_{c+1} + B_d u_{c+1} \\
    \vdots \\
    A_d x_{c+T-1} + B_d u_{T-1}
\end{bmatrix}
= S_c z_c + n_c,
\]

where

\[
S_c = \begin{bmatrix}
    B_d & 0 & 0 & \cdots & 0 \\
    A_d B_d & B_d & 0 & \cdots & 0 \\
    \vdots & \vdots & \vdots & \ddots & \vdots \\
    A_d^{T-1} B_d & A_d^{T-2} B_d & A_d^{T-3} B_d & \cdots & B_d
\end{bmatrix},
\]

\[ n_c = \begin{bmatrix}
    A_d x_c \\
    A_d^2 x_c \\
    \vdots \\
    A_d^T x_c
\end{bmatrix}, \]

and \( z_c \in \mathbb{R}^{nT \times 1} = [u_c^T, u_{c+1}^T, \ldots, u_{T-1}^T]^T \). Then, the MPC estimates the optimal vector of system inputs \( z_c^* \) that minimizes a cost function \( J \) in every control cycle over \( T \) time steps as \([10]\)

\[ z_c^* = \arg \min_{z_c} J(z_c, T), \]  

where

\[
J(z_c, T) = \sum_{k=c}^{c+T-1} \left( [r_k - x_k]^T Q [r_k - x_k] + [v_k - u_k]^T R [v_k - u_k] \right),
\]

s.t.

\[
x_{k+1} = A_d x_k + B_d u_k, \quad \forall k = c, \ldots, c + T - 1, \\
x_{\text{min}} \leq x_k \leq x_{\text{max}}, \quad \forall k = c, \ldots, c + T - 1, \\
u_{\text{min}} \leq u_k \leq u_{\text{max}}, \quad \forall k = c, \ldots, c + T - 1, \\
\Delta u_{\text{min}} \leq \Delta u_k \leq \Delta u_{\text{max}}, \quad \forall k = c, \ldots, c + T - 1.
\]  

In the above cost function, \( r_k = [r_k^1, \ldots, r_k^n]^T \) and \( v_k = [v_k^1, \ldots, v_k^n]^T \) are the reference states and reference inputs at \( k \), respectively. \( x_{\text{min}} \) and \( x_{\text{max}} \) are the minimum and maximum bounds of the states, respectively. Similarly, \( u_{\text{min}} \) and \( u_{\text{max}} \) are the minimum and maximum bounds of the inputs, respectively. \( \Delta u_{\text{min}} \) and \( \Delta u_{\text{max}} \) are the minimum and the maximum bounds of the inputs rate of change. The matrices \( Q \) and \( R \) are the weight matrices for states and inputs, respectively.

Within the sampling time \( t_s \), the MPC controller should solve the optimization problem in (4) to find the optimal control inputs and apply the first control input to the system, at every control cycle. However, the solution time of (4) may exceed \( t_s \), especially for long horizon \( T \), and thus the controller gives a lagging input to the system in this case. In the following, we discuss the idea of parametrization that reduces the computational time of solving (4).
2.1 Parametrization

The optimization function in (4) is solved to find $\mathcal{T}$ vectors of system inputs. By parametrizing the input trajectories with some basis functions, the optimization (4) is solved but for a fewer number of control inputs. Specifically, the system inputs are estimated only at specific sample instances over the horizon in every control cycle, while the rest is evaluated from the parameterized version. Without loss of generality, we build our discussion here upon the parametrization technique in [10], which simply linearizes the trajectories with line segments, as shown in Fig. 1.

Let $\mathcal{P}$ represents the number of sample instances over the horizon $\mathcal{T}$ at which the control inputs are estimated from the optimization (4) (such as the yellow points in Fig. 1). The distance between these sample instances in the unit of time steps can be expressed as

$$\Delta T = \frac{\mathcal{T} - 1}{\mathcal{P} - 1}.$$

The input $u_k$ can be derived by linear interpolation of the inputs at the next sample $k_n$ and the previous one $k_p$ as

$$u_k = (1 - w)u_{k_p} + (w)u_{k_n},$$

where

$$k_p = \left\lfloor \frac{k}{\Delta T} \right\rfloor, \quad k_n = k_p + 1, \quad \text{and} \quad w = \frac{k}{\Delta T} - k_p.$$

By modifying the dynamic equation (3) by $u_k$ in (6), we get

$$x_{k+1} = A_dx_k + (1 - w)B_du_{k_p} + (w)B_du_{k_n},$$

thus, the cost function (5) is modified to be

$$J'(z^*_c, T, \mathcal{P}) = \sum_{k \in \mathcal{T}} \left( (r_k - x_k)^T Q (r_k - x_k) + [v_k - u_k]^T R [v_k - u_k] \right),$$

s.t. the same constraints as in (5), where $\mathcal{T} = \{c, c + \Delta T, \ldots, c + (\mathcal{P} - 1)\Delta T\}$. Now,

$$z^*_c = \arg \min_{z_c} J'(z_c, T, \mathcal{P}).$$

This makes the optimization to be solved on $\mathcal{P} < \mathcal{T}$ steps. Moreover, the dimensions of the matrices of the optimization problem are reduced. Specifically, the dimension of $z_c$ will be $(n\mathcal{P} \times 1)$ instead of $(n\mathcal{T} \times 1)$ and the matrix $S_c$ will have a dimension of $(m\mathcal{T} \times n\mathcal{P})$ instead of $(m\mathcal{T} \times n\mathcal{T})$. Therefore, the parametrization reduces the computational time required in each control cycle. However, this reduction in computational time may be at the expense of the MPC performance, especially for small $\mathcal{P}$. This is because the control inputs that are evaluated using (6) may be different than the inputs estimated from the original optimization in (4). This difference vanishes when $\mathcal{P}$ is close to $\mathcal{T}$.

3 Motivation and problem statement

As shown in (8), there are two factors that strongly affect both the computational burden and the performance of the MPC, which are the horizon length $\mathcal{T}$ and the sample count $\mathcal{P}$. To enhance the performance of the MPC, both $\mathcal{T}$ and $\mathcal{P}$ are fixed to specific large values for all control cycles. When the trajectories of states have rapid changes over future time, the MPC takes the advantage of large $\mathcal{T}$ and $\mathcal{P}$ for better preparing the appropriate inputs $z^*_c$. This allows the MPC to avoid the overshoot that may occur at these rapid changes, which results in better performance but at the expense of more computational cost. However, fixing $\mathcal{T}$ and $\mathcal{P}$ are not optimal for all trajectories in every control cycle. For example, if the trajectories do not show too many variations, then we can use smaller values for $\mathcal{T}$ and $\mathcal{P}$ to solve (8) without noticeable loss in control performance but at much less computational cost. This means that it is better to set a variable value for $\mathcal{T}$ and $\mathcal{P}$ for every control cycle according to the variations exhibited in the trajectories. In the rest of the paper, we represent the variable horizon length and sample count as $T_c$ and $P_c$, respectively, for the $c$th control cycle.

In the following, we first present an experimental validation that illustrates the effect of different fixed settings for $\mathcal{T}$ and $\mathcal{P}$ on both the MPC’s computational time and performance followed by our problem statement that mathematically formulates the problem of adaptive selection of $T_c$ and $P_c$ in every control cycle.

3.1 Motivating examples

We validate our claim on two different models. The first model is a linear vehicle model that is built based on a sim-
The vehicle’s state reference trajectories are presented in a, b. The robot’s state reference trajectories are presented in c–e. The black line represents a reference trajectory that is characterized by slow changes, while the green line represents a reference trajectory that is characterized by rapid changes.

We run our experiments using four different settings for $T$ and $P$ of the MPC. To differentiate between these settings, we denote $\text{MPC}(a,b)$ for a specific setting of $T$ to $a$ and $P$ to $b$. The four settings used in our experiments are $\text{MPC}(40,40)$, $\text{MPC}(5,5)$, $\text{MPC}(40,3)$, and $\text{MPC}(40,25)$. Thus, the first two settings are concerned with both long and short values for $T$, respectively, without parametrization, i.e., $P = T$. The other two settings are concerned with a long $T$, but with different settings for $P$. We evaluate the performance of the MPC with every setting with both the average computational time and the average cost over all control cycles. Assume that we have $H$ control cycles, the average cost $E$ is computed as

$$
E = \frac{1}{H} \sum_{c=1}^{H} \frac{1}{T} J'(z^*_c, T, P).
$$

In Fig. 3, we plot both the average cost $E$ and the average computational time for the four different settings. As shown in the figure, if the reference state trajectories exhibit rapid changes, the average cost becomes very large when using a short horizon or using a long horizon with a small number of sample counts, but the computational time, in this case, is relatively small. For the same trajectories, when using a long horizon or using a long horizon with a sufficient number of sample counts, the average cost becomes smaller but at the expense of more computational time. In the case of reference state trajectories with slow changes, the average cost becomes small, i.e., good performance, for all different settings used in the MPC even with short horizons or a small number of samples. However, this good performance is obtained in a much smaller computational time when using short horizons or a small sample count.

From the results, we can conclude that despite increasing the horizon length and sample count improving the control performance in general, it is not required in all situations. In real scenarios, the state trajectories may exhibit both rapid and slow variations. Therefore, the horizon length and the

---

2 We used the same $H$ for all experiments.
Fig. 3  Plot of the average computational time and the average cost against different settings of MPC for (a) the vehicle and (b) the robot. If the reference state trajectories exhibit rapid changes, the average cost is large when using a short horizon MPC(5,5) or using a long horizon with a small sample count of MPC(40,3). For slow variations, the average cost becomes small for all settings. However, the computational time is much smaller for short horizons MPC(5,5) or for a small sample count MPC(40,3).

Sample count should not be fixed and should be adaptively selected according to the variations encountered in the state trajectories in every control cycle. With this adaptive selection, the computational cost of the MPC is reduced, while it maintains its performance unaffected.

3.2 Problem statement

A pictorial representation of our problem is shown in Fig. 4. According to the situation of the trajectories in the cth control cycle, the values of \( T_c \) and \( P_c \) should be selected as minimum as possible such that the performance of the MPC is not deteriorated compared with its performance when using large values for horizon length and sample count. Let \( T^I \) represents this large value, then our problem can be formulated as

\[
\begin{align*}
\{ T^*_c, P^*_c \} &= \min_{T,P} \{ T, P \} \\
\ell \left( \frac{1}{T} J'(z^*_c, T, P), \frac{1}{T^I} J'(y^*, T^I, T^I) \right) &< \epsilon, \\
\text{s.t.} \\
T &\in \{2, \ldots, T^I\}, \ P < T,
\end{align*}
\]

where \( \epsilon \) is a small positive number and \( y^* \) is the vector of optimal control inputs that is estimated when using \( T^I \) for both the horizon length and the sample count. In our formulation, we assess the loss in performance of MPC by \( \ell \) which measures the relative difference of the average of the cost function \( J'(T) \) when using the values of \( \{ T, P \} \) instead of \( T^I \) for both horizon length and the sample count, i.e.,

\[
\ell(a, b) = \frac{|a - b|}{b}. \tag{11}
\]

A possible solution for (10) is that, in every control cycle, we can start with small values of \( T \) and \( P \) then iteratively increment both values until \( \{ T^*_c, P^*_c \} \) are estimated. However, this solution is not practical in run time (i.e., when the controller is in action) as it involves solving the MPC optimization (8) several times in every control cycle to obtain \( \{ T^*_c, P^*_c \} \). In the following section, we present our proposed efficient approach to estimate \( \{ T^*_c, P^*_c \} \).

4 Proposed adaptive regression-based MPC

In this section, we present our adaptive regression-based MPC (ARMPC) scheme which adaptively estimate \( \{ T^*_c, P^*_c \} \) in every control cycle from (10) using a regression model. Our proposed approach is illustrated in Fig. 5. Specifically, we build a dataset to train a regression model, then we use the regression model to predict \( \{ T^*_c, P^*_c \} \). The predicted \( \{ T^*_c, P^*_c \} \) are used to estimate the required control inputs \( z^*_c \) from (8) as usual. In the following, we discuss the feature extraction, the dataset creation, and the regression model in more detail.
4.1 Feature extraction

Let \( \alpha_i^c = [r_{i_1}^c, \ldots, r_{i_{c+T_l-1}}^c] \) denotes the \( i \)th reference state trajectory in the \( c \)th control cycle. We first extract a feature vector \( \beta_i^c \) from each \( \alpha_i^c \). We designed \( \beta_i^c \) to capture the variation in the reference state trajectory as it is the key element that determines the best minimum values of horizon length and sample count, as we indicated earlier. Specifically, the vector \( \beta_i^c \), composed of

- **Curvature** \( C(\alpha_i^c) \) is a value that quantifies the amount by which \( \alpha_i^c \) deviates from being a straight line. Mathematically [30],

\[
C(\alpha_i^c) = [\kappa(r_{i_1}^c), \ldots, \kappa(r_{i_m}^c)].
\]

where

\[
kappa(r_{i_k}^c) = \frac{1}{T_l} \sum_{k=c}^{c+T_l-1} \frac{|\nabla^2 r_{i_k}^c|}{(1 + (\nabla r_{i_k}^c)^2)^{3/2}}, \quad (12)
\]
Thus, the error between the values of the reference state and the current state at the time instant $c$ is large, it is preferred to use large values for both horizon length and sample count. When the error is large, it is preferred to use large values for both horizon length and sample count to perfectly derive the states to their reference.

After we compute $\beta^*_c$ from each $\alpha^*_c$, we concatenate the vectors $\beta^*_c$ for all $i = 1, \ldots, m$ along with the error vector $e_c$ to form the complete feature vector $\beta_c$, i.e.,

$$\beta_c = [\beta^1_c, \ldots, \beta^m_c, e_c]^T.$$  

(14)

To generate the reference states trajectories for feature extraction, we synthetically utilize the discrete state space equation (2) for a given model by randomly varying the control inputs, while satisfying their constraints.

### 4.2 Dataset creation

We build the dataset by associating each feature vector $\beta_c$ of the $c$th control cycle with its corresponding values of the best minimum horizon $T^*_c$ and sample count $P^*_c$. Algorithm 1 illustrates how $\{T^*_c, P^*_c\}$ are obtained. Specifically, our dataset is built in two successive steps. First, we estimate $T^*_c$, then we estimate $P^*_c$ for the specific estimated $T^*_c$. To estimate $T^*_c$, we initially set both the horizon length and the sample count to a small value $T$, then iteratively increment $T$ and solve (8) with both horizon length and sample count set to $T$. In each iteration, the loss in performance (11) of the MPC is computed. If the loss falls below a threshold $\epsilon$, $T^*_c$ is estimated and the iterations are stopped. Otherwise, $T^*_c$ is set to the maximum allowable horizon length $T^g$. Once we obtain $T^*_c$, we estimate $P^*_c$ in the next step. We set the sample count to a small value $P$, then iteratively increment $P$ and solve (8) with horizon length equal to $T^*_c$ and sample count set to $P$. In each iteration, the loss (11) is computed and if it falls below $\epsilon$, the value of $P^*_c$ is returned.

**Algorithm 1 Estimation of $T^*_c$ and $P^*_c$ for the $c$th control cycle for dataset creation.**

**Input:** $T^g, \epsilon$  
**Output:** $T^*_c, P^*_c$

1: $T \leftarrow 2$  
2: $a \leftarrow \frac{1}{T} J(y^*_c, T^g, T^g)$  
3: while $T < T^g$ do  
4: $b \leftarrow \frac{1}{T} J(z^*_c, T, T)$  
5: if $\ell(b, a) < \epsilon$ then  
6: $T^*_c \leftarrow T$  
7: break  
8: end if  
9: $T \leftarrow T + 1$  
10: end while  
11: $P^*_c \leftarrow P$  
12: $P \leftarrow 2$  
13: while $P < T^*_c$ do  
14: $c \leftarrow \frac{1}{P} J(z^*_c, T^*_c, P)$  
15: if $\ell(c, a) < \epsilon$ then  
16: $P^*_c \leftarrow P$  
17: break  
18: end if  
19: $P \leftarrow P + 1$  
20: end while  
21: $P^*_c \leftarrow P$

Algorithm 1 has high computational cost as it involves solving the MPC optimization (8) several times to obtain $\{T^*_c, P^*_c\}$. However, all these computations are performed offline, i.e., in the training time. But, once the dataset is built,
we predict the values of \( \{ T^*_c, P^*_c \} \) in run time using very simple calculations, thanks to the regression model that we discuss next.

### 4.3 Regression analysis

Our training dataset contains \( N \) records, one for each control cycle. Every record is composed of the feature vector \( \beta_c \) for the \( c \)th control cycle with its corresponding values of \( \{ T^*_c, P^*_c \} \). Our goal here is to build two regression models, the first is for predicting the horizon length and the second one is for predicting the sample count, from the given feature vector. We used the non-linear support vector regression (SVR) [32] technique for building our regression models because of its robustness to outliers, excellent generalization capability, and high prediction accuracy. The objective of the SVR is to find the hyperplane \( \omega^T \phi(\beta_c) + b \) that holds maximum training observation within the margin \( \tau \) (tolerance level), as shown in Fig. 7, where \( \phi(\beta_c) \) is a transformation that maps \( \beta_c \) to a higher-dimensional space. Mathematically, the goal of the training of the SVR model is to find the best parameters \( \omega^* \) for the hyperplane by solving

\[
\{\omega^*, b^*\} = \arg\min_{\omega, b} \frac{1}{2} \|\omega\|^2 + C \sum_{c=1}^{N} (\zeta^+_c + \zeta^-_c),
\]

\[
\text{s.t.}
\]

\[
\forall c : T^*_c - (\omega^T \phi(\beta_c) + b) \leq \tau + \zeta^+_c,
\]

\[
\forall c : \omega^T \phi(\beta_c) + b - T^*_c \leq \tau + \zeta^-_c,
\]

\[
\forall c : \zeta^+_c, \zeta^-_c \geq 0,
\]

where \( C > 0 \) is a regularization parameter that penalizes the number of deviations larger than \( \tau \). The \( \zeta^+_c \) and \( \zeta^-_c \) are slack variables that allow the regression to have errors, as shown in Fig. 7.

We used the same formulation (15) to train the second SVR model that predicts the sample counts but with a slight modification for the constraints. Since we estimate \( P^*_c \) in our dataset creation for a specific \( T^*_c \), as shown in Sect. 4.2, we augment the feature vector \( \beta_c \) with \( T^*_c \) and treat \( P^*_c \) as the required output.

The training of the regression in (15) is solved using the dual problem form as shown in [33] with a Gaussian kernel \( K(x, y) = \exp \left(-\|x - y\|^2\right) = \phi(x)^T \phi(y) \). We used the cross-validation method [34] with grid search to estimate the values of the hyper-parameters \( C, \tau, \zeta^+_c, \) and \( \zeta^-_c \). As we mentioned, we build two regression models, the first is for predicting the best horizon length \( T^*_c \), and the second one is for predicting the best sample count \( P^*_c \). The output from the training is the best parameters \( \{\omega^*_c, b^*_c\} \) and \( \{\omega^*_p, b^*_p\} \) of the hyperplanes of the SVRs corresponding to both \( T^*_c \) and \( P^*_c \), respectively. The complete control process of the proposed ARMPC is outlined in Algorithm 2.

#### Algorithm 2 Adaptive regression-based model predictive control

1. Create the dataset as in Algorithm 1.
2. Train two SVRs by solving Eq. (15) to estimate \( \{\omega^*_c, b^*_c\} \) and \( \{\omega^*_p, b^*_p\} \).
3. \( c \leftarrow 0 \).
4. while \( c < H \) do
5. Compute the feature vector \( \beta^i_c \) for each \( \omega^i_c \) using Eq. (13).
6. Compute the error vector \( e_c \leftarrow [e^1_c, \ldots, e^n_c] \), where \( e^i_c = |r^i_c - x^i_c| \).
7. Compute the feature vector \( \beta_c \) using Eq. (14).
8. \( T^*_c \leftarrow \omega^T \phi(\beta_c) + b^* \).
9. \( P^*_c \leftarrow \omega^p \phi(\beta_c, T^*_c) + b^p \).
10. Solve \( z^*_c \leftarrow \arg\min_{\zeta^+_c, \zeta^-_c} J' (z^*_c, T^*_c, P^*_c) \).
11. Get the first \( n \) elements from \( z^*_c \) as the optimal input \( u_c \leftarrow z^*_c[1 : n] \).
12. Apply the optimal input \( u_c \) and update states \( x_{c+1} \leftarrow A_x x_c + B_u u_c \).
13. \( c \leftarrow c + 1 \).
14. end while

### 5 Experimental results

We evaluate our proposed ARMPC on linear vehicle and non-linear robot models used in Sect. 3.1. We are interested in controlling both position and orientation for both models. For the vehicle, we control the lateral position only, while for the robot, we control both longitudinal and lateral positions, assuming the states of both models are measurable to the MPC.

Independent of our experiments, we build the dataset to train our regression model as we discussed in Sect. 4.2. We set the parameters of Algorithm 1 as \( \epsilon = 0.05 \) and \( T^I = 40 \). The generated training datasets contain \( N = 2, 000, 000 \) for both models. Also, we used the Daubechies wavelet with order 2 [35] to perform the wavelet decomposition with \( L = 3 \) levels. The parameters of our regression model are estimated with 5-fold cross-validation and their values are \( C = 7.4129 \), \( \tau = 0.62 \), and \( \zeta^+_c = \zeta^-_c = 0.1 \). Figure S.3 in the supplementary material shows the reference spatial trajectories for both the vehicle and the robot. Also, Fig. S.4 in the supplementary material shows their reference state trajectories, which as in the reality, they may contain both rapid and slow variations as shown in the figure. In all conducted experiments, the output measurements from both models are obtained with added noise to simulate the modeling errors and/or disturbances that may occur in a real scenario. An extended Kalman filter [36] is used in state estimation. All experiments are carried out for 40s. We used Matlab/Simulink software running under windows 10, with a PC (i7-8550U CPU @ 1.80 GHz, 16 GB RAM).
In the following three subsections, we first present the performed experiments that compare the proposed ARMPC with other state-of-the-art techniques. Then, we present an ablation study that compares the proposed ARMPC with defeatured versions obtained by introducing some modifications to the proposed technique. Finlay, we discuss the behavior of the proposed ARMPC when encountering a non-optimal control situation that may arise due to the infeasibility of solving the MPC optimization before the control cycle time is over or due to system characteristics and constraints.

5.1 Comparison with other techniques

We compare the proposed ARMPC with three different state-of-the-art MPC techniques, which are the parametrized MPC (PMPC) [10], the adaptive dual MPC (ADMPC) [15], and the adaptive neural network MPC (ANMPC) [28] techniques. Additionally, we include the standard MPC (SMPC) with a long horizon as a baseline for performance. We used the same notations in Sect. 3.1 to denote both the PMBC and SMPC, which are MPC(40, ) and MPC(40,40), respectively. Our comparison is performed using two experiments. In the first experiment, we tune the parameters of every technique to provide the best performance so we can compare the amount of speed-up of each technique with the baseline. In the second experiment, we tune the parameters to give the same speed-up as our proposed ARMPC so we can compare the performance of each technique. The parameters of all techniques for both experiments are listed in Sect. S.III in the supplementary material. To ensure a fair comparison, the parameters of the MPC’s optimization (7) and constraints settings in all compared techniques are kept the same as shown in Table S.II and Table S.III, respectively, in the supplementary material. We used the quadprog [37] convex solver for solving the MPC’s optimization (7) for all techniques. For every technique, we measure the time span from the start of every control cycle till the estimation of the control inputs, then we report the average computational time across all cycles.

The results\(^3\) of the first experiment are shown in Fig. 8. Specifically, we plot both the average cost \(E\) in (9) and the average computational time for all techniques under comparison. As shown in the figure, the proposed ARMPC provides a much smaller average computational time with comparable performance to the MPC(40,40). Specifically, the proposed ARMPC reduces the average computational time by 61% and 64.5% compared with MPC(40,40), for the vehicle and the robot, respectively. Also, the proposed ARMPC shows a 35%-52% reduction in the average computational time compared with the other techniques without loss in performance. This reduction in computational time is obtained because our proposed ARMPC does not estimate the horizon length only and it does not rely on raw values of the reference states but employs feature engineering to prevent overfitting in the learning phase. Additionally, the proposed ARMPC extracts these features over a future span of the horizon which leads to a more accurate estimation. Conversely, both ADMPC and PMPC techniques fix the sample count so it is required to use a large enough sample count to give good performance in all situations which reflects in the overall average computational time. Also, the ANMPC technique fixes the ratio between the predicted sample count and the time span of the horizon, uses only the instantaneous values of the states, and does not employ any feature engineering for making predictions. These drawbacks make the technique provide a poor estimation of the sample count which is reflected in the figure where the ANMPC technique shows the poorest performance with high computational time.

In Fig. 9, we plot the histograms of the predicted sample count that is obtained by the proposed ARMPC and the ANMPC techniques for the first experiment. The histograms show that the proposed ARMPC provides a smaller number of occurrences of a large sample count which reinforces the conclusion drawn from Fig. 8 that the proposed ARMPC has a smaller average computational time compared with the ANMPC technique. Note that, we omit the other techniques

---

\(^3\) A video that shows simulation for the vehicle controlled by the proposed ARMPC is shown in Sect. S.IV in the supplementary material.
Fast adaptive regression-based model…

Fig. 8 The results of the first experiment in which we tune the parameters of every technique to provide the best performance. The proposed ARMPC provides a much smaller average computational time with comparable performance to the MPC with long horizon MPC(40,40). Also, the proposed ARMPC shows a 35–52% reduction in the average computational time compared with the other techniques without loss in performance.

Fig. 9 Density distribution for the estimated sample count by the ARMPC and ANMPC techniques. The proposed ARMPC provides a smaller number of occurrences of a large sample count, while the ANMPC technique provides large number of occurrences of a large sample count due to the wrong estimation from the histograms because these techniques do not estimate variable sample count.

The results of the second experiment are shown in Fig. 10 where we plot both $E$ and the average computational time for all techniques. As shown in the figure, all techniques fail to provide comparable performance with the proposed ARMPC for the same average computational time. For the PMPC technique, we decrease its sample count till we reach the same average computational time as our proposed ARMPC. However, this negatively affects the performance, since there are some situations (states with large variations), and the MPC needs enough samples to represent the horizon length. Also,
5.2 Ablation study: comparison with defeatured versions

In this section, we study the effect of each feature proposed in Sect. 4.1 on the overall performance and computational time of the proposed ARMPC. Specifically, we compare the proposed ARMPC with three de-featured versions. Each de-featured version is obtained by dropping one of the proposed features from the training of the SVR and consequently from the online prediction. We denote (ARMPC$^W$), (ARMPC$^C$), and (ARMPC$^E$) for the de-featured versions of the proposed ARMPC that is obtained by dropping the wavelet, the curvature, and the error features, respectively. Additionally, we study the effect of predicting only the horizon length (without the sample count). We denote (ARMPC$^P$) for the de-featured version that trains only one SVR to predict the horizon length only. For this version, we always set $P = T$.

Figure 11 shows $E$ and the average computational time of the proposed ARMPC in comparison with the four defeatured versions. As shown, predicting the horizon length only without estimating the best minimum sample count as in ARMPC$^P$ increases the average computational time with no significant enhancement in the performance. Thus, estimating both the sample count and the horizon length has a very important impact on the computational time reduction.

Also, dropping any feature from the proposed features affects the quality of the estimation which is negatively reflected in the performance. This is because the dropped features make the rest indistinctive and in this case, the dataset may have records with the same feature values but with two different associated values of $\{T, P\}$. For example, suppose that there is a situation where the reference state trajectories have slow variations and small curvature. If the instantaneous error is large in this case, then Algorithm 1 will choose a large value for $\{T, P\}$ and vice-versa, even with such slow variations and small curvature. Thus if the error feature is dropped, then several records will appear in the dataset with similar values of the curvature and wavelet features but with different associated $\{T, P\}$. This results in low fitting for the SVR and will lead to non-optimal predictions. Similarly, dropping the wavelet features decreases the performance as the remaining features can not capture whether the state trajectories have rapid or slow variations. Therefore, a dataset record may be constructed with a small value of $T, P$ if the instantaneous error is small regardless of the variations encountered in the trajectories. So we can conclude that all parts of the proposed features are very important for obtaining the best performance in the least amount of computational time.

5.3 Infeasibility and non-optimal control

In this subsection, we discuss the behavior of the proposed ARMPC when encountering a non-optimal control situation that may arise due to the in-feasibility of solving the MPC optimization before the control cycle time is over or due to system characteristics and constraints.

When the in-feasibility happens because the optimization incurs high computational costs due to a large horizon and

---

Fig. 10 The results of the second experiment in which we tune the parameters of each technique to give the same speed-up as our proposed ARMPC. All techniques fail to provide comparable performance with the proposed ARMPC for the same average computational time.
Fast adaptive regression-based model… 567

Fig. 11 Plot of the average computational time and the average cost of the proposed ARMPC in comparison with four defeatured versions. Drooping any of the proposed features has a negative effect on the performance and increases the computational time due to the wrong estimation of optimal \( [T, P] \).

Fig. 12 A plot of the percentage of converging to the optimal solution \( \sigma \) for different values of the MPC solver’s allowable maximum number of iterations \( N \). For large values of \( N \), both proposed ARMPC and SMPC approaches show high values of \( \sigma \). However, for small values of \( N \), ARMPC shows a significantly higher \( \sigma \) than SMPC. This is because the proposed ARMPC adaptively estimates the best minimum horizon in every control cycle, and therefore, it has smaller calculations involved in finding the optimal solution.

can not find the optimal solution before the control cycle time is over, the optimization returns a sub-optimal solution. To compare our proposed ARMPC approach with the SMPC approach, we conduct the following experiment on both the vehicle and the robot models. We limit the maximum number of iterations \( N \) the MPC solver allows to find the optimal solution and quantify the percentage of the number of times both the ARMPC and the SMPC approaches converge to the optimal solution from the total number of control cycles \( H \). We denote this percentage by \( \sigma \). We repeat the same experiment by varying \( N \) and record the associated \( \sigma \) for both ARMPC and SMPC approaches and plot \( \sigma \) against \( N \) in Fig. 12. As shown in the figure, when \( N \) is large, both ARMPC and SMPC converge to the optimal solution often. However, when \( N \) is smaller, the proposed ARMPC shows a significantly higher percentage of converging to the optimal solution. This is because the ARMPC approach estimates the optimal \( [T^*_c, P^*_c] \), and therefore, it has smaller calculations involved in finding the optimal solution. Thus, the proposed ARMPC approach shows a significant advantage over SMPC in this case.

In the second case, when the in-feasibility happens due to system characteristics and constraints (for example, when the constraints are too tight), we compare the behavior of the ARMPC and the SMPC approaches through the following experiment. We alter the two models’ characteristics by adding more tightened constraints to the MPC optimization problem to force the optimization problem to be infeasible.
The added constraints are presented for both the vehicle and the robot models in more detail in Table S.IV in the supplementary material. In Fig. 13, we plot $\sigma$ for different values of $N$. The figure shows that both ARMPC and SMPC fail to get acceptable $\sigma$ values. Specifically, neither has a value of $\sigma$ greater than 2% even with higher values of $N$. This is simply because the solution can not be found due to the MPC’s optimization problem infeasibility.

6 Conclusion and future work

In this paper, we propose an adaptive regression-based MPC (ARMPC) technique that predicts the best minimum horizon length and the sample count from several features extracted from the reference state trajectories of the system being controlled. The features are designed to capture the variation in the trajectories by using the wavelet decomposition coefficients and the curvature value, in addition to the instantaneous error between the reference state and the current state. We conducted several experiments on both linear and non-linear models to compare the proposed technique with three different state-of-the-art techniques. The results show that the proposed technique provides a superior reduction in computational time with a reduction of about 35–65% compared with the other techniques without introducing a noticeable loss in performance. Additionally, we showed experimentally that dropping any of the proposed features makes our regression model not provide an accurate estimation for the best minimum horizon length and the sample count which affects both the performance and the computational time.

In the future, we plan to apply the proposed approach to non-linear MPC with non-linear MPC solvers, such as the genetic algorithm. Another direction is to apply machine learning techniques to the genetic algorithm to adaptively select the best parameters according to some features extracted from the reference and state trajectories.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s11768-023-00153-y.

Code Availability The code is available here https://github.com/ahmed-elliethy/fast-regression-mpc

Declarations

Conflict of interest The author declares no competing interests.

References

1. Garcia, C. E., Prett, D. M., & Morari, M. (1989). Model predictive control: Theory and practice—A survey. Automatica, 25(3), 335–348.
2. Hrovat, D., Di Cairano, S., Tseng, H. E., & Kolmanovsky, I. V. (2012). The development of model predictive control in automotive industry: A survey. In 2012 IEEE international conference on control applications (pp. 295–302). IEEE.
3. Bakošová, M., & Oravec, J. (2014). Robust MPC of an unstable chemical reactor using the nominal system optimization. Acta Chimica Slovaca, 7(2), 87–93.
4. Wang, C., & Song, Z. (2018). Convex model predictive control for rocket vertical landing. In: 2018 37th Chinese control conference (CCC) (pp. 9837–9842). IEEE. https://doi.org/10.23919/ChiCC.2018.8483147
5. Song, D., Chang, Q., Zheng, S., Yang, S., Yang, J., & Joo, Y. H. (2020). Adaptive model predictive control for yaw system of variable-speed wind turbines. Journal of Modern Power Systems and Clean Energy, 9(1), 219–224.
6. Mostafa, E., Elliethy, A., & Aly, H. A. (2022). Response-time enhancement of a hybrid controller of unmanned aerial vehicles using parametrization. In: 2022 13th International conference on electrical engineering (ICEENG) (pp. 150–153). IEEE.
Fast adaptive regression-based model...
Hussein A. Aly received a B.Sc. (excellent with honors) in computer engineering, an M.Sc. degree in electrical engineering from the Military Technical College (MTC), Egypt, in 1993 and 1997, and a Ph.D. in electrical engineering from the University of Ottawa in 2004. He has been an Assistant/Associate/Full Professor of Electrical and computer engineering at MTC (2004–2021) and the chief of the department (2010–2013). He was a visiting professor in the electrical engineering department at the University of Rochester (Sep 2012–Mar 2013). Entirely shifted to the industry since Jan 2022, he is leading AI and Cloud Computing at Egabi Solutions. His research interests are AI/ deep learning, cloud computing, high-quality image magnification, interpolation of color filter array data, data fusion, video steganography, and embedded computer systems for video processing. Prof. Aly is a senior member of the IEEE (since 2011) and the Society for optics photonics (SPIE).

Ahmed Elliethy received a B.Sc. degree (excellent with honors) in computer engineering and an M.Sc. in electrical engineering from the Military Technical College (MTC), Cairo, Egypt, in 2003 and 2010, respectively. He received his Ph.D. in Electrical Engineering from the University of Rochester, NY, USA, in 2017. In March 2017, he joined the Department of Computer Engineering, MTC, as an Assistant Professor. Currently, he is the head of the Embedded Systems Research Center in the MTC. Dr. Ahmed’s research interests include image processing and computer vision, embedded systems, machine learning, linear and non-linear control optimization, multi-modal image fusion, multiple object tracking, multimedia security, and forensics.