2026 IEEE/SICE International Symposium on System Integration (SII)
January 11-14, 2026. Cancún, México

Dynamic Model Updates and Prediction Horizon Optimizations in Self-Tuning Model Predictive Control

Takashi Nammoto[1]

Abstract— **Model Predictive Control (MPC) is increasingly being adopted across various industrial sectors due to its capability to manage constraints and nonlinearities effectively. This paper presents a novel approach for dynamically updating the MPC model and optimizing the prediction horizon during operation, ensuring both real-time performance and robustness. The method seamlessly integrates hardware and software, allowing for continuous operation without any downtime. Experimental results indicate that the proposed approach sustains optimal control performance, even in the face of changes in system characteristics caused by aging and wear.**

I. INTRODUCTION

Model predictive control (MPC) is widely used in various industrial fields due to its ability to handle constraints and nonlinearity [1]. To apply MPC to nonlinear systems, flexible modeling techniques are necessary. Approaches include using the Takagi-Sugeno model [2], Gaussian process regression [3], and neural networks [4]. Real-time control for fast-responding systems requires maintaining real-time capability. Methods include using differential dynamic programming (DDP) [5], solving quadratic programming problems with locally linearized models [6], implementing MPC on FPGA [7], using the Pontryagin minimum principle [8], and applying the bisection method for modular multilevel converters (MMCs) [9]. The prediction horizon is crucial for MPC accuracy and speed. Longer horizons improve accuracy but increase computation time. Solutions include online prediction horizon optimization [10] and combining long and short horizons [11]. Prior studies have discussed MPC's real-time performance and accuracy.

For industrial machines requiring quick and accurate operation despite aging and wear, this research also addresses MPC's real-time performance and robustness [12]. Real-time performance was ensured using the bisection method for efficient control input search, and robustness was achieved through a self-tuning function adapting to environmental changes, making it suitable for industrial machinery. This approach is useful, but in actual factories, there is no room to stop the equipment for updating the MPC model or optimizing the prediction horizon. It is required that these tasks be performed dynamically during operation. Therefore, this paper proposes a method for integrating hardware and software to dynamically perform model updates and prediction horizon optimizations for self-tuning MPC. The advantages of the method proposed in this study over the approach described in Ref. [12] are summarized in Table I.

[1]Takashi Nammoto is with Information Technology R&D Center, Mitsubishi Electric Corporation, Japan. Nammoto.Takashi@ds.MitsubishiElectric.co.jp

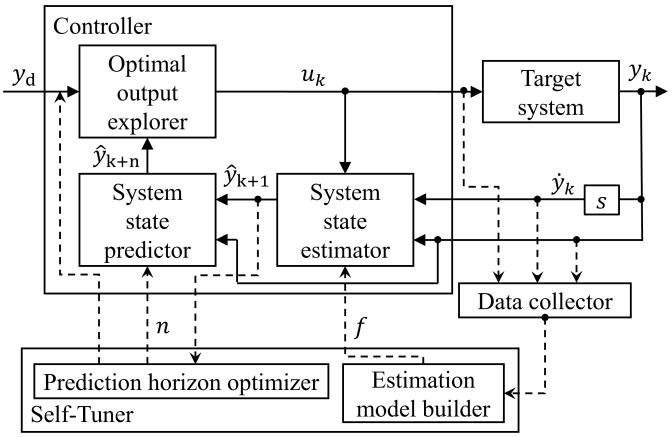

Fig. 1: Overview of self-tuning MPC

TABLE I: Advantages of this paper over Ref. [12]

Item	This paper	Ref. [12]
Model update	Online	Offline
Optimization of prediction horizon	Online	Offline
Transition from PID to MPC	Auto	Manual

In the remainder of this paper, the proposed method will first be explained in Sec. II. Next, the experimental results will be presented in Sec. III, and finally, the conclusions and future works will be discussed in Sec. IV.

II. PROPOSED METHOD

Here, self-tuning MPC is explained in Sec. II-A. Next, in Sec. II-B, the integration method for dynamically performing model updates and prediction horizon optimizations is described.

A. Self-tuning MPC

1) Overview of control method: The overview of self-tuning MPC is shown in Fig. 1. The framework consists of a controller, data collector, and a self-tuner. The controller includes a system state estimator, system state predictor, and optimal output explorer. The system state estimator, at time-step k, estimates the next target system output $\hat{y}_{k+1}$ using the current target system output y_k, its time derivative $\dot{y}_k$, and the control input u_k (Sec. II-A.2.a). The system state predictor quickly predicts the n-step-ahead future target system output $\hat{y}_{k+n}$ using inertial motion prediction, where n is the prediction horizon (Sec. II-A.2.b). The optimal output explorer uses the bisection method to determine the control input u_{k+1} that matches the predicted output

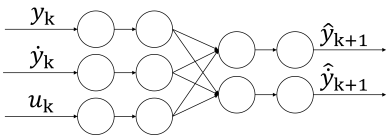

Fig. 2: System state estimator using neural network

with the desired value y_d (Sec. II-A.2.c). This approach gradually aligns the target system output with the desired value, minimizing overshoot and ensuring accurate operation. The calculated control input u_{k+1} is then applied to the target system. The data collector gathers control inputs and target system outputs (Sec. II-A.3). The self-tuner includes a estimation model builder and prediction horizon optimizer. The estimation model builder updates the estimation model of the system state estimator f (Sec. II-A.4.a). The prediction horizon optimizer determines the optimal prediction horizon using the updated model to balance speed and accuracy. At this time, the prediction horizon optimizer provides the desired position to the controller, allowing it to control the system state estimator virtually instead of the target system, while seeking the optimal prediction horizon by observing its behavior (Sec. II-A.4.b). By periodically updating the model and optimizing the prediction horizon, the controller can adapt and optimize itself in response to changes in the target system characteristics.

2) Controller:

a) System state estimator

For systems requiring rapid state changes and real-time responsiveness, it is crucial for the model's computation time to estimate the system's next state to be short. Additionally, a flexible data-driven model is desirable for nonlinear systems. To meet these needs, a shallow neural network is used for modeling as shown in Fig. 2. The input-output relationship of the estimation model is shown in (1).

$$\hat{y}_{k+1}, \hat{\dot{y}}_{k+1} = f\left(y_k, \dot{y}_k, u_k; \boldsymbol{\theta}\right) \tag{1}$$

The neural network is parameterized by $\boldsymbol{\theta}$ and its structure is designed according to the system's behavior.

b) System state predictor

In real-time MPC, a short prediction horizon, such as 1, is often used to enhance performance. However, for systems with inertia, reaching the desired output in one step may cause overshoot due to insufficient time for reverse control. Therefore, it is better to approach the desired value gradually and achieve alignment after the prediction horizon. Calculating the system output after the prediction horizon requires repeating model calculations for each time step, as shown in (2).

$$\begin{aligned} \hat{y}_{k+1}, \hat{\dot{y}}_{k+1} &= f\left(y_k, \dot{y}_k, u_k\right) \\ \hat{y}_{k+2}, \hat{\dot{y}}_{k+2} &= f\left(\hat{y}_{k+1}, \hat{\dot{y}}_{k+1}, u_{k+1}\right) \\ &\cdots \\ \hat{y}_{k+n}, \hat{\dot{y}}_{k+n} &= f\left(\hat{y}_{k+n-1}, \hat{\dot{y}}_{k+n-1}, u_{k+n-1}\right) \end{aligned} \tag{2}$$

This requires n iterations of model calculations, which takes time and affects real-time performance. Therefore, inertial motion prediction is introduced. The system output for the prediction horizon is approximated by assuming inertial motion, as shown in the following equation.

$$\hat{y}_{k+n} \approx y_k + n \times \left(\hat{y}_{k+1}\left(u_k\right) - y_k\right) \tag{3}$$

c) Optimal output explorer

In model predictive control, stable optimization for determining the control input is crucial. The cost function, defined as the difference between the desired value and the predicted future system output, is optimized using the bisection method to find the control input that makes the cost function zero.

$$u_k \quad \text{s. t.} \quad f\left(u_k\right) = y_d - \hat{y}_{k+n} = 0 \tag{4}$$

The controller's processing details are shown in Algo. 1.

Algorithm 1 Controller

Input: y_d, y_k, $\dot{y}_k$
Output: u_k

$u_{\text{lower}} = -u_{\text{limit}}$
$u_{\text{upper}} = u_{\text{limit}}$
for $i \leq i_{\text{end}}$ **do**
  $u_{\text{mid}} = \left(u_{\text{lower}} + u_{\text{upper}}\right)/2$
  $\hat{y}_{k+1} = f\left(y_k, \dot{y}_k, u_{\text{mid}}; \boldsymbol{\theta}\right)$
  $\hat{y}_{k+n} = y_k + n \times \left(\hat{y}_{k+1} - y_k\right)$
  $f\left(u_{\text{mid}}\right) = y_d - \hat{y}_{k+n}$
  if $f > 0$ **then**
    $u_{\text{lower}} \leftarrow u_{\text{mid}}$
  else
    $u_{\text{upper}} \leftarrow u_{\text{mid}}$
  end if
end for
$u_k = \left(u_{\text{lower}} + u_{\text{upper}}\right)/2$

3) Data collector: The data collector gathers essential data for the estimation model builder to construct an estimation model.

4) Self-tuner:

a) Estimation model builder

The estimation model builder uses data collected by the data collector to create the estimation model.

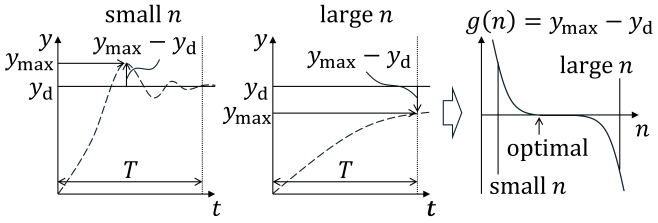

Fig. 3: Relationship between overshoot and prediction horizon

b) Prediction horizon optimizer

First, the optimal prediction horizon is defined. The system's behavior is examined with a rectangular wave-shaped reference value while varying the prediction horizon time-step n, and control is performed based on (4). A short prediction horizon may cause overshoot due to insufficient time for reverse control input. Conversely, a long prediction horizon may prevent the system output from reaching the desired value within the specified time. By defining the maximum system output within a specified time subtracted from the desired value as a function of n, it is expected that n takes positive values when small and negative values when large. This is illustrated in the left and middle plots of Fig. 3. Here, $y_{\max}$ is the maximum system output within period T. The difference between $y_{\max}$ at a certain n and y_{d} is defined as $g(n)$, expected to be positive for small n, approach zero as n increases, remain zero for a while, and then become negative. This transition from overshoot to reaching the desired value within the specified period, and then to not reaching the desired value, is illustrated in the right plot of Fig. 3. The optimal n is the smallest value where $g(n)$ becomes zero, indicated by the arrow in the figure.

The bisection method is applied to find this value, and the smallest n satisfying the conditions is determined.

$$\min(n) \quad \text{s. t.} \quad g(n) = y_{\mathrm{d}} - y_{\max} = 0 \tag{5}$$

The processing details are shown in Algo. 2. This calculation requires repeatedly operating the system, which is impractical. Therefore, the estimated $\hat{y}_{\max}$ obtained by controlling the system state estimator is used instead of the actual $y_{\max}$. Additionally, when $g(n) = 0$ during the update of n, the value of n_{mid} is substituted into n_{upper}, allowing the determination of the smallest n where $g(n)$ becomes zero.

Algorithm 2 Prediction horizon optimizer

Output: n

$n_{\mathrm{lower}} = n_{\mathrm{lower}}^{\mathrm{init}}$
$n_{\mathrm{upper}} = n_{\mathrm{upper}}^{\mathrm{init}}$
for $j \leq j_{\mathrm{end}}$ **do**
  $n_{\mathrm{mid}} = \mathrm{ceil}\left(\left(n_{\mathrm{lower}} + n_{\mathrm{upper}}\right)/2\right)$
  The maximum value of the system output, $\hat{y}_{\max}$, can be obtained by setting the prediction horizon to n_{mid} and the desired value to y_{d}, and controlling the system state estimator for a specified period.
  $g\left(n_{\mathrm{mid}}\right) = \hat{y}_{\max} - y_{\mathrm{d}}$
  if $g \leq 0$ **then**
    $n_{\mathrm{upper}} \leftarrow n_{\mathrm{mid}}$
  else
    $n_{\mathrm{lower}} \leftarrow n_{\mathrm{mid}}$
  end if
end for
$n = \mathrm{ceil}\left(\left(n_{\mathrm{lower}} + n_{\mathrm{upper}}\right)/2\right)$

B. Dynamic model updates and prediction horizon optimizations

1) Overview of integration method: The proposed integration method of dynamic model updates and prediction horizon optimizations in self-tuning MPC is shown in Fig. 4. As illustrated, the controller is implemented on the real-time OS INtime because real-time performance is required. The data collector is also implemented on INtime to collect outputs from both the controller and the target system. On the other hand, the self-tuner, which requires computational power for neural network training and inference, is implemented using PyTorch on Windows with GPU support. These real-time control and self-tuning processes are carried out concurrently, and communication between them occurs through inter-process communication mediated by files that store the results of the processing. This architecture enables dynamic model updates and prediction horizon optimizations.

2) Operation of the controller and self-tuner: The controller incorporates both an MPC control method and a PID control method, with a selector determining which control input to send to the target system. At the beginning of the operation, the selector chooses the control input based on the PID. Once the data collector has finished gathering data and the estimation model builder has completed the estimation model using that data, the selector switches to the control input based on the MPC. Additionally, when the prediction horizon n is optimized by the prediction horizon optimizer, this value is incorporated into the MPC and reflected in the control. As described above, the proposed integration method achieves optimal control for the current situation through a two-step process: switching the control method from PID to MPC and further optimizing the prediction horizon.

III. EXPERIMENTS

A. Experimental setup

An overview of the experimental setup is shown in Fig. 5. The experimental apparatus consists of a PC, an equipment box (the white box on the left side of the middle of the image), and a linear guide (suspended in the center of the top of the image). A position board, which is a PCI card, is installed in the PC. A servo amplifier is installed in the equipment box. A parts list for this apparatus is shown on Table II.

B. Experimental conditions

The experimental conditions are described below.

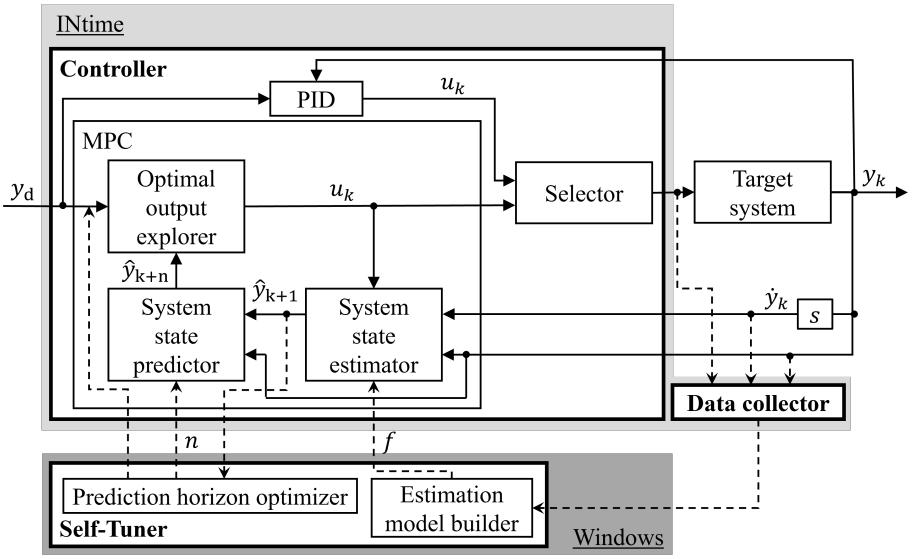

Fig. 4: Overview of dynamic model updates and prediction horizon optimizations

TABLE II: Parts list

Item	Product	Manufacturer	Function
PC	DAIV FX-I9G90	Mouse Computer	Equipped with the real-time operating system INtime, it performs real-time control while utilizing PyTorch and GPU on Windows to update models and optimize the prediction horizon.
Position Board	MR-MC341	Mitsubishi Electric	Interfacing between PC and servo amplifier
Servo Amplifier	MR-J4W2-22B	Mitsubishi Electric	2-axis integrated servo amplifier
Linear Guide	KYL06200M-C2-UA-3A	SURUGA SEIKI	200mm stroke 2-axis linear guide

- One cycle was set to 10.5 seconds, during which the target positions of the linear guide at each time were as follows: 10 mm at 0 seconds, 0 mm at 1 second, -10 mm at 2 seconds, 0 mm at 3 seconds, 20 mm at 4 seconds, 0 mm at 5 seconds, -20 mm at 6 seconds, and 0 mm at 7 seconds.
- The final 3.5 seconds of the cycle was a stationary period, during which tasks such as replacing the workpiece were expected to be performed.
- The aforementioned operation was performed for 5 cycles.
- The control period was 1 millisecond, and the gains for the PID were set to $k_p = 15$, $k_i = 0$ and $k_d = 0$. Additionally, the initial value of the prediction horizon for the MPC was set to 50.
- The activation function of the neural network shown in Fig. 2 was set to the identity function, and the loss function and optimizer used during training were mean squared error (MSE) and Adam, respectively.

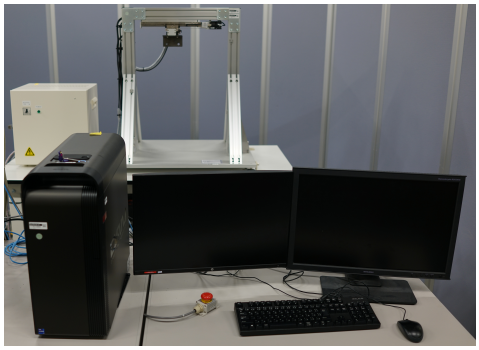

Fig. 5: Overview of experimental setup

C. Results

1) Without friction: The linear guide without friction is shown in Fig. 7, with experimental results in Fig. 6. The upper and lower left sections display the desired position, current position, and control input u (torque command). The desired position is a dashed line, and the current position is a solid line. The lower right section shows the prediction horizon for each cycle. Settling time is in Table III. Data collection and model creation occurred during the 1st and 2nd cycles, using PID control, which was slow due to unadjusted gains. In the 3rd cycle, MPC was used, improving operation but lacking optimized prediction horizon. By the 4th cycle, the prediction horizon was optimized from 50 to 29, enhancing convergence. This method allows for autonomous switching of control methods and further optimization while keeping the device operational.

2) With friction: The linear guide with friction is shown in Fig. 9. As illustrated, a brush was installed to make contact with the tool of the linear guide in order to simulate the conditions of aging and wear. The experimental results for

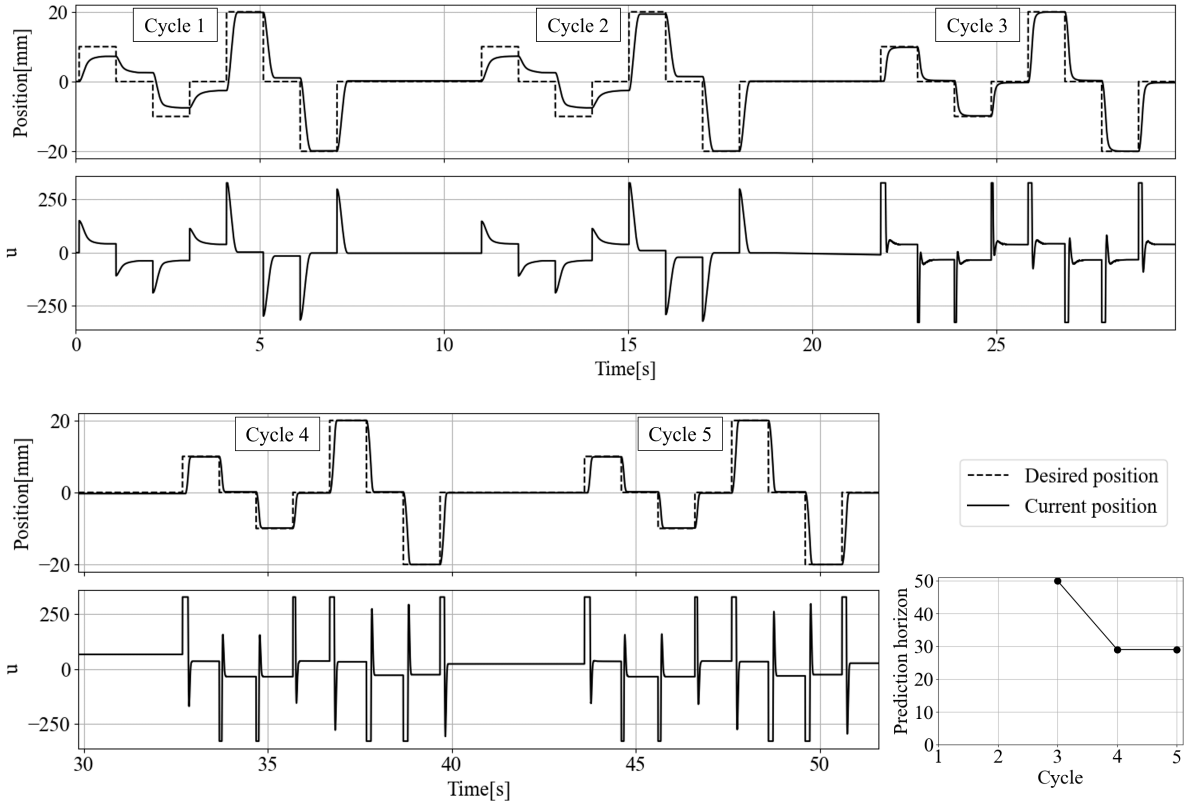

Fig. 6: Experimental results without friction

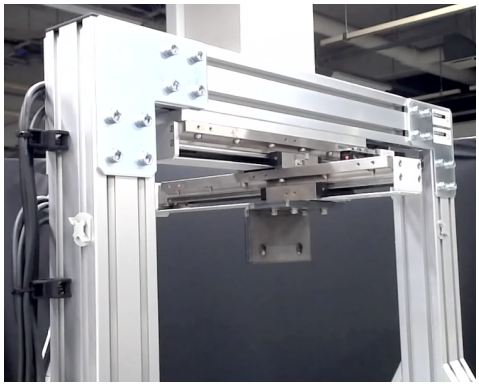

Fig. 7: Linear guide without friction

	PID	MPC ($n = 50$)	MPC ($n = 29$)
Settling time ($\pm 5\%$) [s]	0.453	0.336	0.278
Settling time ($\pm 2\%$) [s]	0.476	0.405	0.289

TABLE III: Settling time without friction

this case are shown in Fig. 8 and Table IV. The operation is roughly similar to the case without friction; however, due to the occurrence of friction, the PID control fails to reach the desired position, whereas the proposed method reliably achieves the desired position, demonstrating its superiority. The difference in this case is that the prediction horizon was changed to the optimal value of 27. This is because the friction suppresses the movement, prompting the prediction horizon optimizer to shorten the prediction horizon in order to reduce the convergence time.

Due to experimental constraints, the focus was on simulating the startup phase. However, the proposed method allows for continuous model updates and optimizes the prediction horizon during operation. This ensures optimal control even when transitioning from a frictionless to a friction state, as the method autonomously adjusts based on current conditions. By implementing this approach, industrial machines, including machine tools, logistics equipment, and robots, can efficiently repeat operations despite aging and wear.

IV. CONCLUSIONS AND FUTURE WORKS

This paper presents a method for dynamic model updates and prediction horizon optimization in self-tuning MPC. It is suitable for real factory operations, enhancing machinery efficiency. Future challenges include improving system adaptability, optimizing performance in various scenarios, and advancing control through modeling techniques.

REFERENCES

[1] J. B. Rawlings and D. Q. Mayne, *Model Predictive Control: Theory and Design*. San Francisco, CA, USA: Nob Hill Pub., 2009.

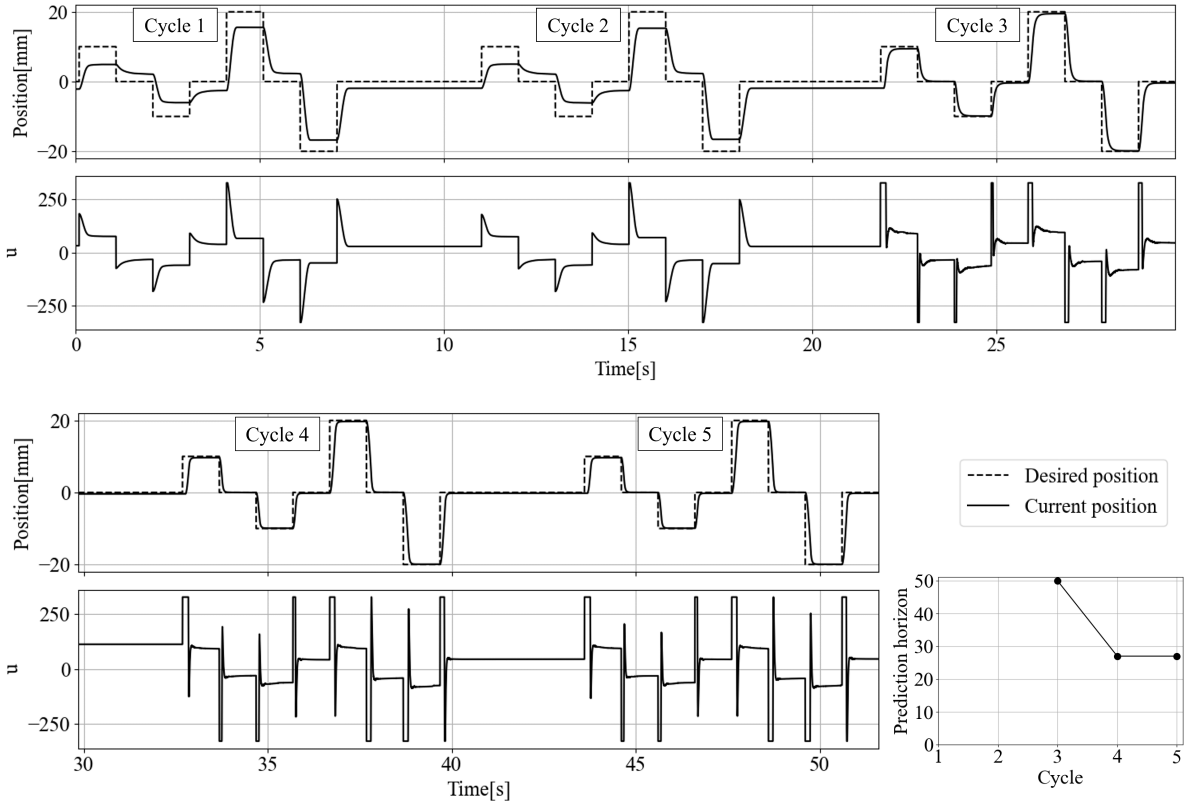

Fig. 8: Experimental results with friction

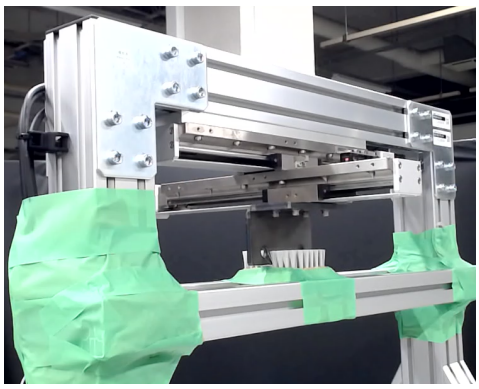

Fig. 9: Linear guide with friction

	PID	MPC ($n = 50$)	MPC ($n = 27$)
Settling time ($\pm 5\%$) [s]	Not achieved	0.399	0.300
Settling time ($\pm 2\%$) [s]	Not achieved	0.457	0.349

TABLE IV: Settling time with friction

[2] Mai Van Sy and Phan Xuan Minh, "Fuzzy model predictive control using Takagi-Sugeno model," *2008 Int. Conf. on Control, Automation and Systems*, pp. 632–637, 2008.

[3] J. Kocijan, R. Murray-Smith, C. E. Rasmussen and A. Girard, "Gaussian process model based predictive control," *Proceedings of the 2004 American Control Conference*, vol. 3, pp. 2214–2219, 2004.

[4] J. Saint-Donat, N. Bhat, and T. J. McAvoy, "Neural net based model predictive control," *Int J. Control*, vol. 54, no. 6, pp. 1453–1468, 1991.

[5] S. Kleff, A. Meduri, R. Budhiraja, N. Mansard and L. Righetti, "High-Frequency Nonlinear Model Predictive Control of a Manipulator," *2021 IEEE Int. Conf. on Robotics and Automation*, pp. 7330–7336, 2021.

[6] J. Lee, M. Seo, A. Bylard, R. Sun and L. Sentis, "Real-Time Model Predictive Control for Industrial Manipulators with Singularity-Tolerant Hierarchical Task Control," *2023 IEEE Int. Conf. on Robotics and Automation*, pp. 12282–12288, 2023.

[7] K. Basterretxea and K. Benkrid, "Embedded high-speed model predictive controller on a FPGA," *Proc. NASA/ESA Conf. Adapt. Hardw. Syst.*, pp. 327–335, Jun. 2011.

[8] E. Pagot, M. Piccinini and F. Biral, "Real-time optimal control of an autonomous RC car with minimum-time maneuvers and a novel kineto-dynamical model," *2020 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 2390–2396, 2020.

[9] S. Hamayoon, M. Hovd and J. A. Suul, "Bisection Algorithm based Indirect Finite Control Set Model Predictive Control for Modular Multilevel Converters," *2021 IEEE 22nd Workshop on Control and Modelling of Power Electronics*, pp. 1–6, 2021.

[10] M. Cai, W. Wu and X. Zhou, "Trajectory Tracking Control for Autonomous Parking Based on Adaptive Reduced-horizon Model Predictive Control," *IEEE International Conference on Networking, Sensing and Control*, pp. 1–6, 2022.

[11] T. Brüdigam, D. Prader, D. Wollherr and M. Leibold, "Model Predictive Control with Models of Different Granularity and a Non-uniformly Spaced Prediction Horizon," *American Control Conference*, pp. 3876–3881, 2021.

[12] T. Nammoto and T. Mariyama, "Robust Real-Time Model Predictive Control using Inertial Motion Prediction, Bisection, and Self-Tuning," *International Conference on Automation Science and Engineering (CASE)*, pp. 2281–2286, 2024.