Iterative Learning Control and Model Predictive Control for Trajectory Tracking with Model Mismatch

Hangfei Li*

Department of Mechanical Engineering, University of Michigan, Ann Arbour, the United States

*Corresponding author: lhangfei@umich.edu

Abstract. Model predictive control (MPC) is a powerful technique that can be used for various applications, including trajectory tracking. However, MPC requires an accurate model of the system and a long enough prediction horizon to achieve good performance. Although variations of MPC like stochastic MPC and robust MPC can alleviate the problem to a certain degree, they still more or less rely on the knowledge of system models, disturbance and noise. MPC can also suffer from high computational cost or loss of recursive feasibility, particularly in the case of the existence of model mismatch (actual system model is different from the model used by the controller). Iterative learning control (ILC) is another technique that can be used for improving system performance (reducing tracking error) for repetitive tasks. ILC is known for its capability of learning from previous history (iterations of tasks) and less reliance on the model accuracy. In this paper, both MPC and different ILC algorithms are employed for a trajectory tracking problem with model mismatch. The performances of different algorithms are then analyzed and compared. A criterion on the degree of model mismatch to guarantee error convergence is also discussed for model-based ILC.

Keywords: Trajectory tracking; model predictive control; iterative learning control.

1. Introduction

1.1. Motivation

Trajectory tracking refers to following the predefined paths through different controllers. For mobile tasks, trajectory tracking plays a critical role in ensuring the accuracy of movement. From robotic arms conducting high-precision manufacturing operations to mobile robots transporting cargo between fixed way points, all these tasks can be reduced to tracking the trajectory obtained from a high-level motion planning algorithm. While the motion planner plays an important role, the low-level controller is the crucial component that gets any task completed. Therefore, designing robust and efficient control algorithms to achieve high-quality trajectory tracking performance is paramount in many actual applications.

Model Predictive Control (MPC) is widely used for problems such as motion planning, trajectory tracking and so forth. By using an assumed model to predict future behaviors of the control system, MPC computes the optimal control input by solving a constrained optimization problem. This shifts the effort for the design of a controller towards modeling of the to-be-controlled process [1]. Typically, MPC requires long enough horizons and accurate models to achieve the desired performance. However, disturbances and noise usually exist, making it difficult—and in some cases impossible—to obtain an accurate model. Due to factors like part degradation, internal friction and manufacturing tolerance, it is very likely that the model used for the controller becomes inconsistent with the actual model.

Iterative learning control (ILC) is another powerful approach for various tasks, especially in cases where the system needs to perform the same task multiple times. This control method improves the system performance by learning from previous executions [2]. For the trajectory tracking task, ILC uses the history of tracking error and control input from previous iterations to update control input at the next iteration. Besides, ILC is a more flexible method for trajectory tracking as it can be applied in both model-based and model-free scenarios.

In this paper, the performances of MPC and ILC on a trajectory tracking problem are analyzed. A 2D double-integrator model is used for robot dynamics, and the observable states are position in x

and y coordinates. Four approaches, including MPC, two model-based ILC and one model-free ILC, are investigated. All algorithms and simulations are implemented in MATLAB.

1.2. Related Work

A nonlinear model predictive control (NMPC) approach was proposed for surface vessel control, where the unknown hydrodynamic and propulsion parameters were identified experimentally in recent research [3]. Moreover, an adaptive learning model predictive control (ALMPC) scheme is proposed for input-constrained trajectory tracking of perturbed autonomous ground vehicles (AGVs) [4]. Another study about an MPC controller based on a fuzzy adaptive weight control algorithm is proposed for autonomous vehicles since tracking accuracy and dynamic stability are both guaranteed [5]. MPC based trajectory tracking is also applied to autonomous vehicles to compute the optimal forces and moments the vessel needs for the path [6].

ILC is also a powerful method for path tracking. A novel PD-type iterative learning controller is used on a pneumatic X-Y table system to perform path tracking and reject disturbance [7]. ILC is also utilized in perspective dynamic system (PDS) to overcome uncertainties in trajectory tracking of mobile service robots [8]. The combination of adaptive ILC and neural networks is proposed to overcome the limitation of required nominal parameters to improve the efficiency of trajectory tracking [9]. ILC is also employed in autonomous racing for nonlinear vehicle dynamics, like the Audi TTS race car [10]. Both proportional-derivative ILC and quadratic optimal ILC are capable of reducing path-tracking errors.

2. Problem Statement

2.1. Trajectory Reference

The trajectory reference is a circular path starting from the origin. It has a radius of 1 and centers at the coordinate (-1,0). The start point and end point are both at the origin. The mobile target starts and stops at the origin with zero velocity at each iteration. Suppose the moving angle relative to the center between the current position and the starting point is θ and each task takes 10 seconds. Besides, the velocity $\dot{\theta}$ is chosen to be a quadratic equation with respect to time as follows:

$$\dot{\theta} = -\frac{3\pi t^2}{250} + \frac{3\pi t}{25} \tag{1}$$

Note $\dot{\theta}$ is 0 when t = 0 and t = 10. The area enclosed by the curve and the x-axis is 2π. The moving angle θ is calculated by integration. Therefore, the trajectory reference is represented as follows.

$$p_x = \cos(\theta) - 1 \tag{2}$$

$$p_y = \sin(\theta) \tag{3}$$

where p_x and p_y are the positions at x and y directions.

2.2. System Dynamics

A two-dimensional double-integrator model is used for system dynamics, which is common for mobile robots. The state space contains fours state variables:

$$x = [x, \dot{x}, y, \dot{y}]^T \tag{4}$$

where x, y are the positions in x and y directions, $\dot{x}$ and $\dot{y}$ are the velocity in the x and y directions. The idealized continuous double integrator model is represented as

$$\dot{x} = Ax + Bu \tag{5}$$

$$y = Cx \tag{6}$$

where the control input u is the acceleration in x and y directions. The output y is chosen to be the position. The matrices for a nominal (idealized) continuous system can be written as

$$\mathrm{A_{nominal}} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{7}$$

$$\mathrm{B_{nominal}} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}^{\mathrm{T}} \tag{8}$$

$$\mathrm{C_{nominal}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{9}$$

The idealized double-integrator model is usually different from the actual system because of inner friction, vibration, etc. The A matrix for the actual system in continuous time is set to be

$$\mathrm{A_{actual}} = \begin{bmatrix} 0 & 0.8 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.8 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{10}$$

Both the nominal (idealized) model and the actual model are employed to test the performance of different trajectory tracking strategies.

Since the algorithms are implemented in MATLAB with discrete time data, the continuous model is transformed into discrete form. The discrete system model is represented as below.

$$\mathrm{x_{k+1}} = \mathrm{A_d x_k} + \mathrm{B_d u_k} \tag{11}$$

$$\mathrm{y_k} = \mathrm{C x_k} \tag{12}$$

where $\mathrm{A_d}$ and $\mathrm{B_d}$ are the matrices in discrete time corresponding to the matrices A and B of the two-dimensional double-integrator model in continuous time. The sampling time is set to be 0.1 second.

Suppose the discrete system has N time steps in state and control input, the output at step k can be written as

$$\mathrm{y_k} = \mathrm{C A_d^k x_0} + \mathrm{C} \sum_{\mathrm{m=0}}^{\mathrm{k-1}} \mathrm{A_d^{k-1-m}\, B_d u_m} \tag{13}$$

The equation mapping the control input U and output trajectory Y at all time steps is

$$\mathrm{Y} = \mathrm{GU} + \mathrm{d} \tag{14}$$

$$\mathrm{Y} = [\mathrm{y_1} \quad \mathrm{y_2} \quad \cdots \quad \mathrm{y_N}]^{\mathrm{T}} \tag{15}$$

$$\mathrm{U} = [\mathrm{u_0} \quad \mathrm{u_1} \quad \cdots \quad \mathrm{u_{N-1}}]^{\mathrm{T}} \tag{16}$$

$$\mathrm{G} = \begin{bmatrix} \mathrm{CB_d} & 0 & \cdots & 0 \\ \mathrm{CA_dB_d} & \mathrm{CB_d} & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{CA_d^{N-1}B_d} & \mathrm{CA_d^{N-2}B_d} & \ldots & \mathrm{CB_d} \end{bmatrix} \tag{17}$$

$$\mathrm{d} = \begin{bmatrix} \mathrm{CA_d^0 x_0} \\ \mathrm{CA_d^1 x_0} \\ \vdots \\ \mathrm{CA_d^N x_0} \end{bmatrix} \tag{18}$$

The mapping terms $\mathrm{G_{actual}}$, $\mathrm{d_{actual}}$ and $\mathrm{G_{nominal}}$, $\mathrm{d_{nominal}}$ can be calculated from actual dynamic system matrices and nominal dynamic system matrices respectively.

2.3. Trajectory Tracking

Model Predictive Control (MPC) is one of the control algorithms commonly used for trajectory tracking. Fig.1 shows the working principle of MPC. It solves a series of control inputs by optimizing the cost function at each time step in a receding horizon fashion. The prediction horizon N is often set in advance, and the cost function J is a function of state variable x and control input u at time step k as follows:

$$u^* = \min_{k_0:N-1,u_0:N-1} J(x_k, u_k) \quad \text{s.t.}\, x_{k+1} = A_d x_k + B_d u_k, u_{min} \le u_k \le u_{max} \tag{19}$$

where $x_{k+1} = A_d x_k + B_d u_k$ stands for the constraints of system dynamics and $u_{min} \le u_k \le u_{max}$ stands for the constraints on actuator limits.

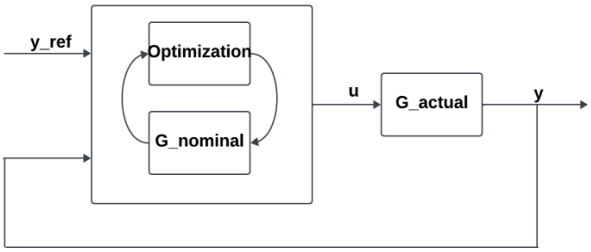

Fig. 1 Model predictive control block diagram

Different Iterative learning control (ILC) algorithms are also adopted for trajectory tracking. ILC improves tracking performance by learning the history of control input and tracking error from previous iterations. Fig. 2 illustrates the working principle of ILC.

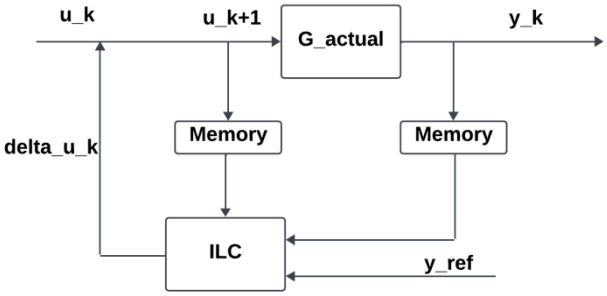

Fig. 2 Iterative learning control block diagram

Before ILC algorithms, direct collocation (DC) is utilized to generate the initial control input. DC is a typical trajectory optimization technique by choosing a series of states and constraints along the reference to calculate the control input. Fig.3 shows the difference between the reference trajectory and the path resulting from DC. The corresponding initial control is then used for the first iteration of ILC to refine the control input.

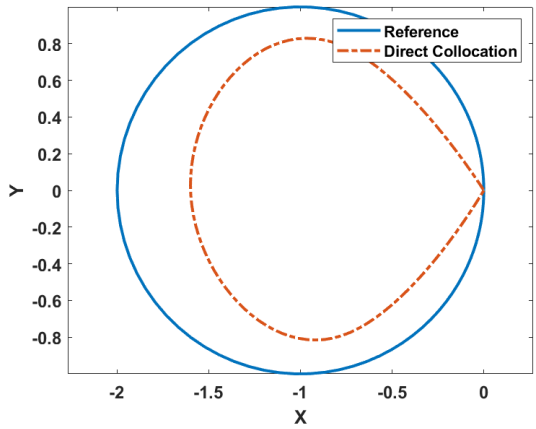

Fig. 3 Trajectory reference and MPC result against time

3. Methods

3.1. MPC Formation

MPC is an optimal control technique that utilizes the plant model and objective function to calculate control input by solving optimization problems. In this case, the cost function is designed to be a quadratic function to penalize both tracking error and control effort.

$$J = (y_{ref} - y)^T Q(y_{ref} - y) + u^T Ru \tag{20}$$

Here y_{ref} is the reference trajectory. Q is the state weighting matrix penalizing the deviation of the system output from the reference trajectory. R is the control weighting matrix penalizing control effort. The weighting matrices are set to be

$$Q = 10 * I \tag{21}$$

$$R = 0.1 * I \tag{22}$$

where I is the identity matrix.

The MPC controller uses the nominal model for optimization, while the system states get updated using the actual model. N is the total time steps, and at each step k, the system state is updated as

$$x_{k+1} = A_{actual}x_k + B_{actual}u_k \tag{23}$$

$$y_k = C_{actual}x_k \tag{24}$$

The tracking result is shown in Fig.4.

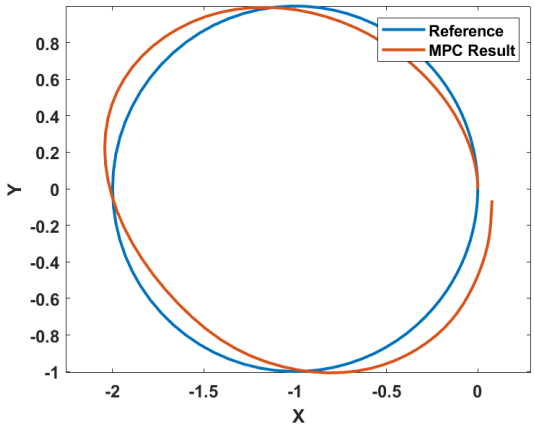

Fig. 4 Resulting trajectory of the MPC algorithm and reference trajectory

3.2. Direct Collocation

Direct collocation (DC) is a common trajectory optimization method. By using system dynamics in discrete form, DC turns the infinite dimensional optimization problem into a finite dimensional problem. The output of DC is then used as the initial control input for the first iteration of both model-based and model-free ILC. Below shows the formulation of DC.

$$\min_{u_k:N-1} \sum_{k=0}^{N-1} \left(u_k^T u_k\right) \quad \text{s.t.}\, x_{k+1} = A_{nominal}x_k + B_{nominal}u_k,$$

$$u_{min} \le u_k \le u_{max},$$

$$x_{N/4} = [-1, 1]^T,$$

$$x_{N/2} = [-2, 0]^T,$$

$$x_{3N/4} = [-1, -1]^T,$$

$$x_0 = x_{start},$$

$$x_N = x_{goal} \tag{25}$$

where N is the same total number of time steps as MPC. Besides the constraints on system dynamics (using nominal model), control input (same actuator limit as MPC, only five additional way-points are imposed (including initial state and final state) as constraints. Three way-points (−1, 1), (−2, 0) and (1, −1) are set to be achieved at $\frac{N}{4}, \frac{N}{2}, \frac{3N}{4}$ respectively. x_0 and x_N correspond to the start and end state. These constraints may not be the exact time when these way- points are achieved on the reference trajectory. DC is only used for generating the initial control input. The control input will later be refined by ILC. Therefore, it has a lower requirement on constraint resolution and accuracy. Since DC is computed off-line, it is also less computationally intensive than MPC.

3.3. Model-based ILC

Two methods are investigated for model-based ILC algorithms. They both essentially solve the same optimization problem using a nominal model in the iteration domain. One is based on Quadratic Programming (QP) and the other one is based on Linear Quadratic Regulator (LQR). The model-based iterative learning process at each step is shown below.

$$y_k = G_{actual} u_k + d_{actual} \tag{26}$$

$$e_k = y_{ref} - y_k \tag{27}$$

$$u_{k+1} = u_k + \Delta u \tag{28}$$

where e_k represents the tracking error at each time step. The iteration continues until e_k is less than 0.01.

For the quadratic programming algorithm, Δu is calculated from matrices H, A_{eq}, B_{eq}. H is a matrix used for constructing the quadratic cost function, and A_{eq} and B_{eq} are the matrices for constructing the equality constraints. In MATLAB, Δu is solved by the quadprog function and H, A_{eq}, B_{eq} are set to be

$$H = I \tag{29}$$

$$A_{eq} = G_{nominal} \tag{30}$$

$$b_{eq} = zeros \tag{31}$$

For the linear quadratic regulator, Δu is calculated as follows.

$$\Delta u = -K e_k \tag{32}$$

K is the Gain matrix. Matrices A, B, Q, R are required to compute K. Q and R are matrices that assign weight to the penalty on tracking error and change of control input respectively. Matrices A and B essentially impose the same equality constraint as QP. In MATLAB, the gain matrix K is solved by dlqr function and A, B, Q, R are set to be

$$A = I \tag{33}$$

$$B = -G_{nominal} \tag{34}$$

$$Q = 10 * I \tag{35}$$

$$R = 0.1 * I \tag{36}$$

3.4. Model-Free ILC

The model-free ILC(ILC-MF) does not require a model of the system. The output at each time step is directly measured from the actual system. This algorithm uses the inverse time operator τ to get the adjoint form of matrix G_{actual}, and adjust the control input in a gradient-descent manner by the learning rate α. Since the path starts with a zero initial state (zero velocity at the origin), the constant term d_{actual} becomes zero, which simplifies the process of calculating the matrix adjoint. The iterative process at each step k is shown below.

$$y_k = G_{actual} u_k + d_{actual} \tag{37}$$

$$e_k = y_{ref} - y_k \tag{38}$$

$$G^*(e_k) = \tau G_{actual} \tau e_k \tag{39}$$

$$u_{k+1} = u_k + \alpha G^*(e_k) \tag{40}$$

To improve system stability and ensure the convergence of iteration, the learning rate α is calculated as below.

$$\Delta h_k = G^*(e_k) - G^*(e_{k-1}) \tag{41}$$

$$\Delta u_k = e_k - e_{k-1} \tag{42}$$

$$\alpha = \begin{cases} 0.01, \ \Delta h_k{}^T \Delta u_k \geq 0 \\ \frac{-\Delta h_k{}^T \Delta u_k}{\Delta h_k{}^T \Delta h_k}, \Delta h_k{}^T \Delta u_k < 0 \end{cases} \tag{43}$$

4. Discussion

4.1. Comparison on Performances

According to Fig.4, MPC is not able to utilize the history of control input and tracking error. It repeats the same error at every iteration, and the tracking error does not converge to zero. Therefore, in this scenario, MPC is the worst algorithm based on the metric of required model accuracy and convergence speed.

Table 1. Performance comparison for all algorithms

Controller	Model Required	Convergence Speed
MPC	Most Accurate	No Convergence
ILC-QP	Less Accurate	< 10 iterations
ILC-LQR	Less Accurate	< 10 iterations

Fig.5-7 shows the change of tracking error over iterations for ILC-QP, ILC-LQR and ILC-MF algorithms, and Table 1 concludes the performance comparison for these algorithms. With information provided from the nominal model, model-based ILC converges much faster than model-free ILC, as shown in Table 1. By solving the optimization problem to get Δu in the process of model-based ILC, it can be viewed as a type of model-based policy gradient method. This is similar to one of the reinforcement learning algorithms called Deep Deterministic Policy Gradient (DDPG). In this scenario, the system acts like the critic network providing the feedback to each action (control input), and the goal is to minimize the tracking error, similar to maximizing the Q value in DDPG. The controller acts as the actor, and it uses the feedback from tracking error to improve the control policy over iterations.

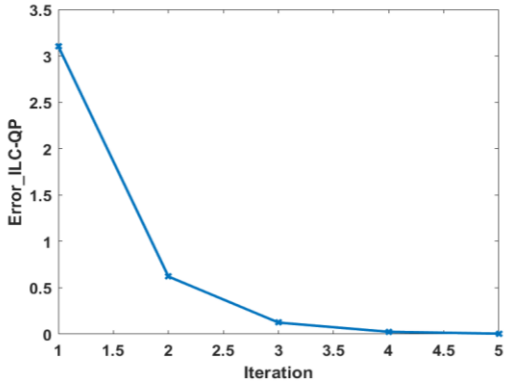

Fig. 5 Change of tracking error over iterations for the ILC-QP algorithm

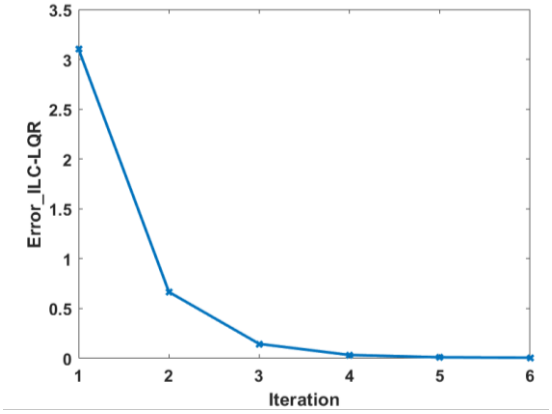

Fig. 6 Change of tracking error over iterations for the ILC-LQR algorithm

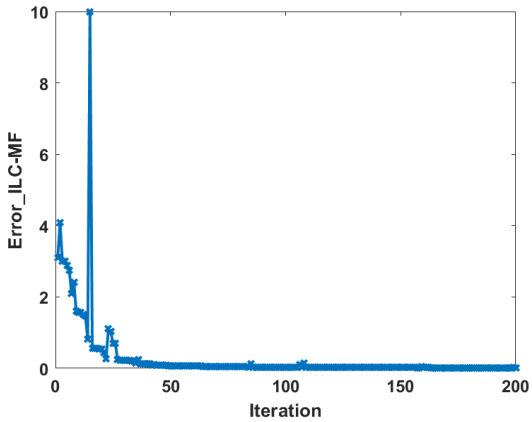

Fig. 7 Change of tracking error over iterations for the ILC-MF algorithm

4.2. Convergence Criterion on Model-Based ILC

Model-based ILC has been shown to have a better convergence rate compared to model-free ILC. However, it does not work on any nominal model, especially when the nominal model is too far from the actual model. Here, the criteria of the nominal model are quantified to guarantee model-based ILC algorithm tracking error convergence.

For a system with an actual plant model G_{actual}, the error dynamics in the iteration domain can be obtained from the state update.

$$e_{k+1} = e_k - G_{actual}\Delta u_k \tag{44}$$

In the previously discussed ILC-QP and ILC-LQR algorithms, the tracking error and control input ultimately boil down to satisfying the fundamental constraint shown below.

$$e_k = G_{nominal}\Delta u_k \tag{45}$$

Therefore, the error dynamics in the iteration domain can be rewritten as follows.

$$e_{k+1} = e_k - G_{actual}{G_{nominal}}^{-1}e_k \tag{46}$$

Based on the theorem of contraction mapping, the tracking error is guaranteed to be monotonically decreasing if the condition for error convergence is satisfied.

$$\left\| I - G_{actual}{G_{nominal}}^{-1} \right\| < 1 \tag{47}$$

5. Conclusion

MPC and three types of ILC algorithms are studied for a reference trajectory tracking task with the existence of model mismatch in this study. ILC demonstrates the advantage for the case of doing

repetitive tasks, and model-based ILC can have a much better convergence property compared to model-free ILC, if the condition on the nominal model is satisfied. One limitation of this study is that the control input constraint is not considered in the ILC algorithm. In addition, tracking error is not monotonically decreasing for model-free ILC, potentially due to the learning rate not being appropriate at each iteration. Thus, future work can be designing a learning function and a filter to improve the model-free ILC algorithm. Disturbance and noise can also be added to extend the work.

References

[1] Schwenzer, M., Ay, M., Bergs, T. et al. Review on model predictive control: an engineering perspective. Int J Adv Manuf Technol 117, 1327–1349 (2021).

[2] D. A. Bristow, M. Tharayil and A. G. Alleyne, "A survey of iterative learning control," in IEEE Control Systems Magazine, vol. 26, no. 3, pp. 96-114, June 2006.

[3] Leticia Mayumi Kinjo, Stefan Wirtensohn, Johannes Reuter, Tomas Menard, Olivier Gehan, Trajectory Tracking of a Fully-actuated Surface Vessel using Nonlinear Model Predictive Control, IFAC-PapersOnLine, Volume 54, Issue 16, 2021, Pages 51-56, ISSN 2405-8963.

[4] Zhang Kunwu,Sun Qi & Shi Yang. (2021). Trajectory Tracking Control of Autonomous Ground Vehicles Using Adaptive Learning MPC.IEEE transactions on neural networks and learning systems,32(12),5554-5564.

[5] H. Wang, B. Liu, X. Ping and Q. An, "Path Tracking Control for Autonomous Vehicles Based on an Improved MPC," in IEEE Access, vol. 7, pp. 161064-161073, 2019.

[6] Huarong Zheng, Rudy R. Negenborn, Gabriel Lodewijks, Trajectory tracking of autonomous vessels using model predictive control, IFAC Proceedings Volumes, Volume 47, Issue 3, 2014, Pages 8812-8818, ISSN 1474-6670, ISBN 9783902823625.

[7] Chih-Keng CHEN & James HWANG. (2006). PD-Type Iterative Learning Control for the Trajectory Tracking of a Pneumatic X-Y Table with Disturbances.JSME International Journal Series C Mechanical Systems, Machine Elements and Manufacturing,49(2),520-526.

[8] Yugang Wang,Fengyu Zhou,Yang Zhao,Ming Li & Lei Yin. (2020). Iterative learning control for path tracking of service robot in perspective dynamic system with uncertainties.International Journal of Advanced Robotic Systems,17(6),1729881420968528-1729881420968528.

[9] Zhang Yueqing, Chu Bing & Shu Zhan. (2019). A Preliminary Study on the Relationship Between Iterative Learning Control and Reinforcement Learning.IFAC-PapersOnLine,52(29),314-319.

[10] N. R. Kapania and J. C. Gerdes, "Path tracking of highly dynamic autonomous vehicle trajectories via iterative learning control," 2015 American Control Conference (ACC), Chicago, IL, USA, 2015, pp. 2753-2758.