Iterative Learning Control for Trajectory Tracking with Model Mismatch

Lihan Lian*, Hangfei Li†

***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 long enough prediction horizon to achieve good performance. Although variations of MPC like stochastic MPC and robust MPC can alleviate 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 existence of model mismatch (actual system model is different from the model used by controller). Iterative learning control (ILC) is another technique that can be used for improving system performance (reduce tracking error) for repetitive task. ILC is known for its capability of learning from previous history (iterations of tasks) and less reliance on the model accuracy. In this study, we use both MPC and different ILC algorithm for a trajectory tracking problem with model mismatch, then analyze and compare their performance. Criterion on the degree of model mismatch to guarantee error convergence is also studied for model-based ILC.**

I. Introduction

A. Motivation

The ability to accurately follow predefined paths, or trajectory tracking, is fundamental to various robotics tasks. From robotic arms doing high precision manufacturing (i.e. 3D printing), to mobile robots transporting cargo between fixed waypoints, all these tasks can be boiled down to tracking the trajectory obtained from high-level motion planning algorithm. While motion planner plays an important rule, low-level controller is the crucial component that get any task completed. Thus, designing of robust and efficient control algorithm to achieve good trajectory tracking performance is paramount in many robotics application.

Model Predictive Control (MPC) is widely used for problems such as motion planning, trajectory tracking and so forth. By using assumed model to predict future state, MPC computes the optimal control input by solving constrained optimization problem online.Typically, MPC requires long enough horizons and accurate models to achieve desired performance. However, disturbance and noise almost always exist and it might be difficult or even impossible to get an accurate model, especially for a complex system. Due to factors like part degradation, internal friction and manufacturing tolerance, it is very likely that the model used for mobile robots controller becomes inconsistent with the actual model of robot.

Iterative learning control (ILC) is another powerful approach for various tasks, especially for the case when robot needs to do it repetitively. For trajectory tracking task, ILC uses history of tracking error and control input from previous iterations to update control input at next iteration. We study both mode-based and model-free ILC, as well as the condition required for model-based ILC error convergence.

This project aims to analyze and compare the performance of different methods on a trajectory tracking problem with the existence of model mismatch. 2D double-integrator model is used for robot dynamics and the observable states are position in x and y coordinate. Four approaches, including MPC, two model-based ILC and one model-free ILC are investigated. All algorithms and simulations are implemented in MATLAB.

B. Related Work

1) *Model Predictive Control:* Many MPC schemes have been developed and applied to trajectory tracking. An adaptive learning model predictive control (ALMPC) scheme is proposed for input-constrained trajectory tracking of perturbed autonomous ground vehicles (AGVs) [1]. Another study about MPC controller based on fuzzy adaptive weight control algorithm is proposed for autonomous vehicles since tracking accuracy and dynamic stability are both guaranteed [2]. MPC based trajectory tracking is also applied to autonomous vehicles to compute the optimal forces and moment the vessel needs for path [3].

2) *Iterative Learning Control:* ILC is also a powerful method can be used for path tracking [4]. A novel PD-type iterative learning controller is used on pneumatic X-Y table system to perform path tracking and reject disturbance [5]. ILC is also utilized in perspective dynamic

*Department of Robotics, University of Michigan
†Department of Mechanical Engineering, University of Michigan
Code is available at https://github.com/lihanlian/trajectory-tracking-ilc

system (PDS) to overcome uncertainties in trajectory tracking of mobile service robots [6]. The combination of adaptive ILC and neural networks is proposed to overcome the limitation of required nominal parameters to improve efficiency of trajectory tracking in [7]. By nature of ILC, it shares many similarities as discussed in [8]. However, most of the work does not discuss on the case of model-mismatch or the convergence criterion on model-based ILC when model-mismatch exists.

C. Paper Structure

The report is organized as follows: section II introduces the problem to be solved, including reference trajectory, system dynamics and trajectory tracking methods. Section III explains the algorithms of different methods in detail. Section IV discusses the performance of different strategies and the conditions on model-based ILC error convergence. Section IV concludes the report by pointing out limitations and shed lights on further investigations can be done for improvement.

II. Problem Statement

Trajectory Reference

The reference trajectory is a circular path starting from origin. It has a radius of 1 and centers at (-1,0). Both start and end point are the origin. The mobile robot starts and stops at origin with zero velocity at each iteration. Suppose the moving angle relative to the center between current robot position and the starting point is θ and each task takes 10 seconds. We choose $\dot{\theta}$ 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 $t = 10$, and area enclosed by the curve and x-axis is 2 pi. The moving angle θ is calculated by integration. Therefore, the trajectory reference is represented as

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

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

where p_x and p_y are the position of the robot at x and y direction. Blue curve in Figure 1 and Figure 2 show the reference against time in 1D and 2D plane.

System Dynamics

This project uses a 2D double-integrator model for system dynamics, which is common for mobile robots. The state space contains fours state variables:

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

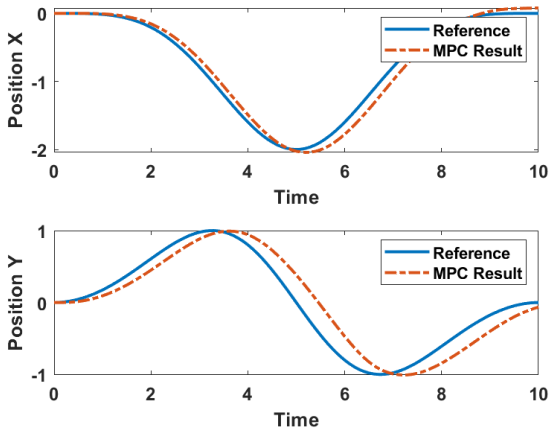

Fig. 1: Trajectory reference and MPC result against time

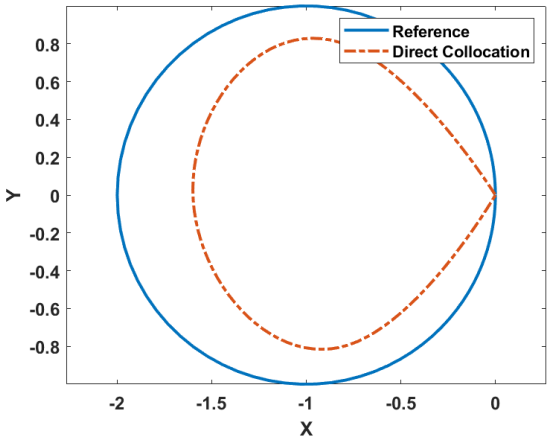

Fig. 2: Trajectory reference and DC result in 2D plane

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

$$\dot{x} = Ax + Bu \tag{4}$$

$$y = Cx \tag{5}$$

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

$$A_{nominal} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad B_{nominal} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}$$

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

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

$$A_{\text{actual}} = \begin{bmatrix} 0 & 0.8 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.8 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

This project uses both the nominal (idealized) model and actual model 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

$$x_{k+1} = A_d x_k + B_d u_k \tag{6}$$

$$y_k = C x_k \tag{7}$$

where A_d and B_d are the matrices in discrete time corresponds to the the matrices A and B of 2D double-integrator model in continuous time. Sampling time is chosen 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:

$$y_k = CA_d^k x_0 + C \sum_{m=0}^{k-1} A_d^{k-1-m} B_d u_m \tag{8}$$

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

$$Y = GU + d \tag{9}$$

where $Y = [y_1\ y_2 \dots\ y_N]^T$, $U = [u_0\ u_1\ \dots\ u_{N-1}]^T$

$$G = \begin{bmatrix} CB_d & 0 & \dots & 0 \\ CA_dB_d & CB_d & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ CA_d^{N-1}B_d & CA_d^{N-2}B_d & \dots & CB_d \end{bmatrix}$$

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

Trajectory Tracking

MPC is one of the control algorithms commonly used for trajectory tracking. Figure 3 shows the working principle of MPC. It solves a series control input by optimizing 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 $k = 0 \dots N$ as shown follows:

$$\begin{aligned} u^* = \min_{k_{0:N-1}, u_{0:N-1}} \quad & J(x_k, u_k) \\ \text{s.t.} \quad & x_{k+1} = A_d x_k + B_d u_k, \\ & u_{min} \le u_k \le u_{max}, \end{aligned}$$

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. The project also uses different algorithms of ILC. This method improves tracking performance by learning the history of control input and tracking error from previous iterations. Figure 4 illustrates the working principle of ILC.

Before using ILC, direct collocation (DC) is used 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 control input. Figure 2 shows the difference between reference trajectory and the path resulting from DC. The corresponding initial control is then used for the first iteration of ILC to refine control input.

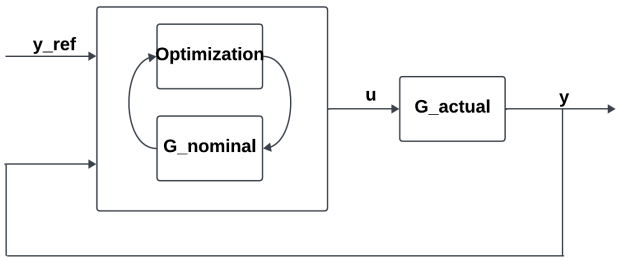

Fig. 3: MPC block diagram

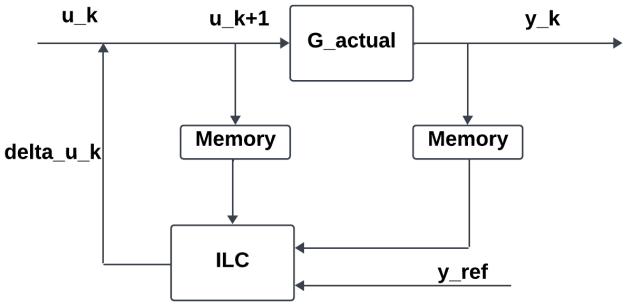

Fig. 4: ILC block diagram

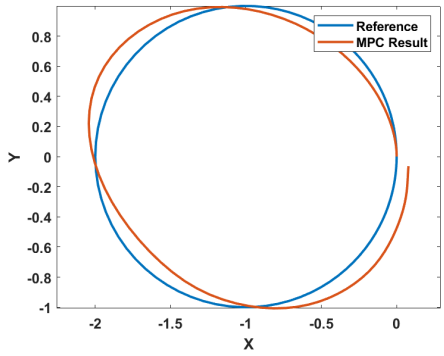

Fig. 5: Resulting trajectory of MPC algorithm and reference trajectory.

III. Methods

A. MPC Formulation

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. Constraints on actuator limits are also applied. The MPC controller uses the nominal model for optimization while the robot states get updated using the actual model. The simulation algorithm is shown as in algorithm 1.

Algorithm 1 MPC

$x_k := x_0$
$J = (y_{ref} - y)^T Q (y_{ref} - y) + u^T R u$
for $k = 0 : N$ **do**
  $y_k = C_{actual} x_k$
  $u_k = mpc(A_{nominal}, B_{nominal}, C_{nominal}, J, y_{ref_k})$
  $x_{k+1} = A_{actual} x_k + B_{actual} u_k$
  $x_k = x_{k+1}$
  $k = k + 1$

Here x_0 and y_{ref} are the same initial state and reference trajectory for each repetitive task. N is the total time step that is also same for finishing each task. Tracking result is shown in Fig. 1 and Fig. 5

B. Direct Collocation

Direct collocation (DC) is a common trajectory optimization method. It discretizes the system dynamics to turn the infinite dimensional optimization problem into a finite dimensional problem. The output of DC is then used as the initial control input u_0 for the first iteration of both model-based and model-free ILC. Below shows the formulation of DC.

$$
\begin{aligned}
\min_{u_{k:N-1}} \quad & \sum_{k=0}^{N-1} (u_k^T u_k) \\
\text{s.t.} \quad & x_{k+1} = A_{nominal} x_k + B_{nominal} u_k, \\
& u_{min} \leq u_k \leq u_{max}, \\
& x_{N/4} = [-1, 1]^T, \\
& x_{N/2} = [-2, 0]^T, \\
& x_{3N/4} = [-1, -1]^T, \\
& x_0 = x_{start}, \\
& x_N = x_{goal},
\end{aligned}
$$

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 waypoints are imposed (including initial state and final state) as constraints. Three waypoints $(-1, 1), (-2, 0)$ and $(1, -1)$ are set to be achieve at $k = \frac{N}{4}, \frac{N}{2}, \frac{3N}{4}$ respectively. x_0 and x_N correspond to start and end state. Note that these constraints may not be the exact time when these waypoints are achieved on the reference trajectory. DC is only used for generating initial control input for our purpose. The control input will later be refined by ILC, thus, it has lower requirement on constraints resolution and accuracy. Since DC is computed off-line, it is also less computational intensive than MPC.

C. Model-Based ILC

Two methods are investigated for model-based ILC algorithms. They both essentially solving the same optimization problem using nominal model in the iteration domain. One is based on quadratic programming (ILC-QP) and the other one is based on LQR (ILC-LQR). Here provide the pseudocode for both algorithms.

Algorithm 2 ILC-QP

$u := u_0$
$e_k = 1, k = 0$
$A_{eq} = G_{nominal}, b_{eq} = zeros$
$H = I$
while $e_k > 0.01$ **do**
  $y_k = G_{actual} u_k + d_{actual}$
  $e_k = y_{ref} - y_k$
  $\Delta(u) = quadprog(H, A_{eq}, b_{eq})$
  $u_k = u_k + \Delta(u)$
  $k = k + 1$

In ILC-QP, H is a matrix used for constructing the quadratic cost function, A_{eq} and b_{eq} are the matrices for

constructing the equality constraints. e_k is the tracking error at iteration k. $\Delta(u)$ is then solved by **quadprog** function in MATLAB and added to variable u_k (control input at iteration k) to improve the control input.

Algorithm 3 ILC-LQR

$u := u_0$
$e_k = 1, k = 0$
$A = I, B = -G_{nominal}$
$Q = 10 * I, R = 0.1 * I$
$K = dlqr(A, B, Q, R)$
while $e_k > 0.01$ **do**
  $y_k = G_{actual} u_k + d_{actual}$
  $e_k = y_{ref} - y_k$
  $\Delta(u) = -K e_k$
  $u_k = u_k + \Delta(u)$
  $k = k + 1$

In ILC-LQR, 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 ILC-QP. Gain matrix K is solved by **dlqr** function in MATLAB, and then multiply with tracking error to get $\Delta(u)$.

D. Model-Free ILC

The model-free ILC does not require a model of the robot as its name suggest. It uses the inverse time operator τ to get the adjoint of matrix G_{actual}, and adjust the control input in a gradient-descent manner using learning rate α. Note that since the robot start with zero initial state (zero velocity at origin), the constant term d_{actual} becomes zero, which simplifies the process of calculating matrix adjoint as shown in algorithm 4.

IV. Discussion

Comparison on Performance

MPC is not able to utilize the history of control input and tracking error, thus, it repeats the same error at every iteration, and tracking error does not converge to zero. Thus, in this scenario, MPC is the worst algorithms based on the metric of required model accuracy and convergence speed.

With information provided from nominal model, model-based ILC covnerges much faster than model-free ILC as shown in table I. By solving the optimization problem to get $\Delta(u)$ 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 algorithm called Deep Deterministic Policy Gradient (DDPG) as shown in Fig. 6. In this scenario, robot acts like the critic network that provide the feedback to each action (control input), and the goal is to minimize the tracking error, similar to maximizing the Q value in DDPG. Controller acts as the actor, and it uses the feedback from robot (tracking error) to improve the control policy over iterations.

Algorithm 4 ILC-MF

$u := u_0, u_{prev} = 0$
$e = 1, k = 0$
$G^* e_{prev} = 0, \tau = fliplr(I)$
while $e > 0.01$ **do**
  $y_k = G_{actual} u_k + d_{actual}$
  $e = y_{ref} - y_k$
  $G^* e = \tau G_{actual} \tau e$
  $\Delta h_i = G^* e - G^* e_{prev}$
  $\Delta u_i = u - u_{prev}$
  if $\Delta h_i^T \cdot \Delta u_i \geq 0$ **then**
    $\alpha = 0.01$
  else
    $\alpha = -\Delta h_i^T \cdot \Delta u_i / (\Delta h_i^T \cdot \Delta h_i)$
  $u_{prev} = u$
  $G^* e_{prev} = G^* e$
  $u = u + \alpha G^* e$
  $k = k + 1$

Controller	Model Required	Convergence Speed
MPC	Most Accurate	No Convergence
ILC-QP	Less Accurate	< 10 iterations
ILC-LQR	Less Accurate	< 10 iterations
ILC-MF	Not Needed	> 200 iterations

TABLE I Performance comparison for all algorithms.

Convergence Criterion on Model-Based ILC

Model-based ILC has been shown to have better convergence rate compare to model-free ILC. However,

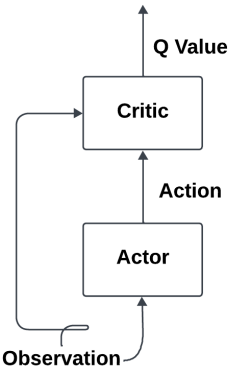

Fig. 6: Actor-Critic algorithm illustration.

it does not work on any nominal model, especially when the nominal model is too far from the actual model. Here we quantify the criteria of nominal model to guarantee model-based ILC algorithm trakcing error convergence.

For system with actual plant model G_{actual}, state update can be described as follows:

$$x_k = G_{actual}u_k + d_{actual} \tag{10}$$

$$x_{k+1} = G_{actual}u_{k+1} + d_{actual} \tag{11}$$

Subtracting Eq. 10 from Eq. 11 we can then get the error dynamics in iteration domain as shown in Eq. 14:

$$x_{k+1} - x_k = G_{actual}(u_{k+1} - u_k) \tag{12}$$

$$x_{k+1} - x_{ref} + x_{ref} - x_k = G_{actual}\Delta u_k \tag{13}$$

$$e_{k+1} = e_k - G_{actual}\Delta u_k \tag{14}$$

Here $\Delta u_k = u_{k+1} - u_k$, $e_k = x_{ref} - x_k$ and $e_{k+1} = x_{ref} - x_{k+1}$. In the previously discussed ILC-QP and ILC-LQR algorithms, they ultimately boil down to satisfying the fundamental constraint shown as Eq. 15.

$$e_k = G_{nominal}\Delta u \tag{15}$$

Assuming $G_{nominal}$ is invertible, then the follow relation can be obtained.

$$\Delta u = G_{nominal}^{-1}e_k \tag{16}$$

Substitute Eq. 16 into Eq. 14, the error dynamics in iteration domain can be rewritten as follows.

$$e_{k+1} = e_k - G_{actual}G_{nominal}^{-1}e_k \tag{17}$$

Thus, based on the theorem of contraction mapping, the condition for error convergence is

$$||I - G_{actual}G_{nominal}^{-1}|| < 1 \tag{18}$$

and error is guaranteed to be monotonically decreasing if Eq. 18 is satisfied.

V. Conclusion

MPC and three types of ILC algorithms are studied for a robot reference trajectory tracking task with the existence of model mismatch in this study. ILC demonstrates the advantage for the case of doing repetitive task and model-based ILC can have much better convergence property compare to model-free ILC, if the condition on nominal model is satisfied. One limitation of this study is that 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 may not be appropriate at each iteration. Thus, future work can be designing learning function $L(q)$ and Q filter [9] to improve model-free ILC algorithm. Disturbance and noise can also be added to extend the work.

References

[1] K. Zhang, Q. Sun, and Y. Shi, "Trajectory tracking control of autonomous ground vehicles using adaptive learning mpc," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 12, pp. 5554–5564, 2021.

[2] H. Wang, B. Liu, X. Ping, and Q. An, "Path tracking control for autonomous vehicles based on an improved mpc," *IEEE Access*, vol. 7, pp. 161 064–161 073, 2019.

[3] H. Zheng, R. R. Negenborn, and G. Lodewijks, "Trajectory tracking of autonomous vessels using model predictive control," *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 8812–8818, 2014, 19th IFAC World Congress. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1474667016430041

[4] K. L. Moore, Y. Chen, and H.-S. Ahn, "Iterative learning control: A tutorial and big picture view," in *Proceedings of the 45th IEEE Conference on Decision and Control*. San Diego, CA, USA: IEEE, Dec. 2006.

[5] C.-K. Chen and J. Hwang, "Pd-type iterative learning control for trajectory tracking of a pneumatic x-y table with disturbances," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, vol. 4, 2004, pp. 3500–3505 Vol.4.

[6] W. Yugang, Z. Fengyu, Z. Yang, L. Ming, and Y. Lei, "Iterative learning control for path tracking of service robot in perspective dynamic system with uncertainties," *International Journal of Advanced Robotic Systems*, vol. 17, no. 6, p. 1729881420968528, 2020. [Online]. Available: https://doi.org/10.1177/1729881420968528

[7] M. Yamakita, M. Ueno, and T. Sadahiro, "Trajectory tracking control by an adaptive iterative learning control with artificial neural networks," in *Proceedings of the 2001 American Control Conference. (Cat. No.01CH37148)*, vol. 2, 2001, pp. 1253–1255 vol.2.

[8] Y. Zhang, B. Chu, and Z. Shu, "A preliminary study on the relationship between iterative learning control and reinforcement learning," in *IFAC-PapersOnLine*, vol. 52, no. 29, IFAC (International Federation of Automatic Control). Elsevier Ltd., 2019, pp. 314–319.

[9] D. A. Bristow, M. Tharayil, and A. G. Alleyne, "A survey of iterative learning control," *IEEE Control System Magazine*, 2006.

VI. Appendix

A. Error Convergence Results

Convergence results are shown for ILC-QP, ILC-LQR and ILC-MF algorithms. All figures can be reproduced by the code at https://github.com/lihanlian/trajectory-tracking-ilc.

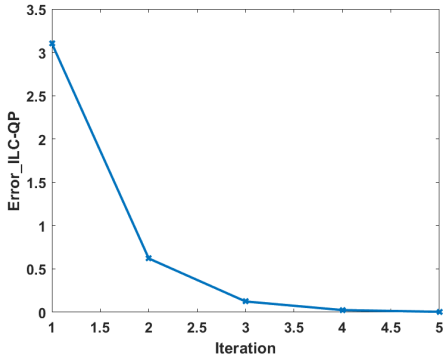

Fig. 7: Change of tracking error over iterations for ILC-QP algorithm.

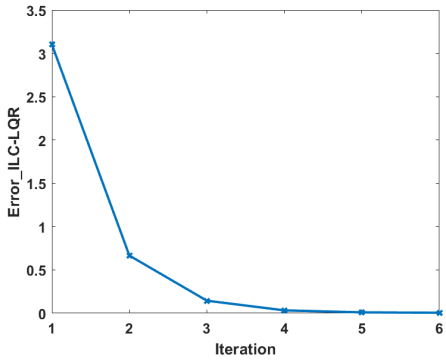

Fig. 8: Change of tracking error over iterations for ILC-LQR algorithm.

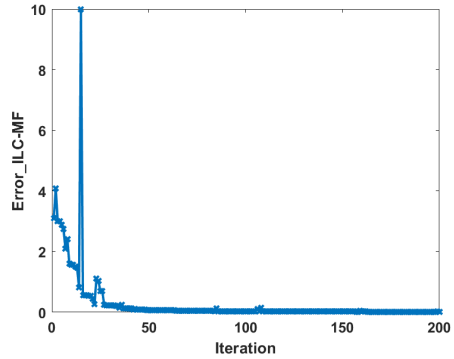

Fig. 9: Change of tracking error over iterations for ILC-MF algorithm.