2024 Tenth Indian Control Conference (ICC)
9 - 11 December, 2024
IISER Bhopal, India

Transfer Learning for Iterative Learning Control Using Gaussian Processes

Letchumanan Muthiah[1], Arun K. Tangirala[2]

Abstract— **Iterative Learning Control (ILC) is a technique used to improve the transient tracking performance of a repetitively operated system, such as a batch process. However, the assumption of iteration invariance of the process conditions limits the extension of ILC to similar processes and often demands process modelling and controller design from scratch. Data-driven modelling strategies are highly effective but they suffer from large model development cost during training. This is undesirable since operating conditions of industrial batch processes can switch rapidly to meet the requirements of the market leaving little time and resources for model development. In this paper, we present a novel method of transferring optimally tuned ILC across processes using the modern ideas of transfer learning (TL). Specifically, we develop a Gaussian process model using ideas of forward adaptive transfer learning (FAT) to facilitate knowledge transfer between processes. A major advantage of the proposed method is that optimal ILC designed for current process can be transferred to a new process using only limited training data. The proposed method is demonstrated on the Cott-Batch Reactor, showing substantial promise for the proposed TL-based FAT-GP.**

I. INTRODUCTION

Most batch processes possess highly nonlinear characteristics with unmodelled disturbances or parameter uncertainties, all of which combine to make the modelling of the process very challenging. Consequently, traditional control techniques may not provide satisfactory transient tracking performance which makes the control of batch processes challenging [1]. If the batch process is operated repetitively under specific conditions, we can iteratively learn to control the process by improving upon trajectory tracking each iteration similar to how humans improve upon their performance on a repetitive task by learning from their mistakes. This forms the basis of the well-known Iterative Learning Control (ILC). ILC requires the initial state of the process to be reset to the same value after each trial, and the process dynamics to be invariant throughout the repetitive operation of the process, among other requirements [2]. Therefore, modifying the batch process conditions renders the existing control strategy ineffective. This severely limits the practical application of ILC since:

a) Real-time-optimization of operating conditions is carried out to reach the specific product specifications in terms of quality and quantity. This means that the initial state of the process can vary rapidly based on the requirement [3].

b) A pilot scale batch process developed for testing purposes can be converted to an industrial scale process if viable during which the process dynamics is bound to change.

Consequently, model development for the process is reinitiated when operating conditions change and the ILC strategy is updated based on the new model. For rapid changes in operating conditions, we need to collect experimental data frequently, which is time-consuming and resource-intensive. However, the mechanisms that govern the dynamics of the process remain the same when operating conditions are changed. This means that there is some knowledge about the process that can be used to accelerate model development under new operating conditions. An ideal learning mechanism suited for this purpose is that of transfer learning (TL), in which a pre-trained model (trained for one task) is re-trained for learning another task with shorter training times (see [12], [13]). There exist several choices for models such as Artificial Neural Networks, Fuzzy Models, etc.; however, they usually require very large amounts of training data before they reach high precision. In cases where only limited data is available, Gaussian Process (GP) models offer efficient representations of the process. GPs have fewer hyperparameters and also provide a measure of confidence in its predictions which distinguishes it from related methods such as Support Vector Regression.

For the case of transfer learning for regression using GPs, [4] presents an Adaptive Transfer Learning Gaussian Process (AT-GP) that uses a transfer kernel to learn task similarity and it tunes the impact that the learned task (source task) has on predictions for the task of interest (target task). But AT-GP constrains the target and transfer kernel to the same kernel as the source (i.e. they share the same mean and covariance function along with their hyperparameters), resulting in an implicit transfer even when the tasks are dissimilar. Additionally, AT-GP only allows for simultaneous and does not allow transfer from previously learned source tasks to new target tasks. To mitigate this [8] proposes Forward Adaptive Transfer Learning (FAT) for GPs that decouples the kernels and hyperparameters of the source and target tasks. The works of [4], [8] and their likes demonstrate the use of GPs in modelling static non-linearities. Extension of GPs to modelling non-linear dynamical systems is given in [7].

The authors are with the Department of Chemical Engineering and Department of Data Science and Artificial Intelligence, Indian Institute of Technology Madras, Chennai, Tamil Nadu, India – 600036, respectively.

[1]letchumanan.mlm@gmail.com
[2]arunkt@iitm.ac.in

In this work we firstly combine the merits of FAT and ideas of using GPs for dynamical systems to develop a GP based TL for dynamical systems. Building on this, we propose a method for designing ILC for dynamic systems under new operating conditions. The main contribution is a novel transfer learning-based method for applying ILC to batch processes that vary in operating conditions or scale.

The rest of the paper is organized as follows: Section II briefly discusses the fundamentals of ILC and GPs. Section III details the methodology for modelling dynamic systems and transfer learning using GPs, followed by Section IV which demonstrates the effectiveness of this method for the Cott-Batch Reactor. Section V concludes the paper.

II. FUNDAMENTALS OF ILC AND GPS

A. Iterative Learning Control (ILC)

ILC is a control strategy where the input signal to a system is refined with each trial of a repetitively operated system. The trajectory tracking errors from previous trials are used to construct the input update. Consider a SISO output linear time-invariant system whose dynamics is represented by a discrete-time, state-space, perturbation model shown in equation (1)

$$\mathbf{z_k[i+1]} = \mathbf{A z_k[i]} + \mathbf{B} u_k[i]$$
$$y_k[i] = \mathbf{C z_k[i]} + \mathbf{D} u_k[i] \quad (1)$$

$u_k[i]_{1\times1}$, $y_k[i]_{1\times1}$, $\mathbf{z_k[i]}_{n_z\times1}$ are the input, output and states of the system respectively at the ith time instant of the k^{th} trial, and n_z is the number of states. Suppose the system operates on a finite time horizon $i \in \{0,1,\cdots, t_f\}$ where t_f is the trial length. We can define a 'supervector' by 'lifting' along the time axis in the k^{th} trial to get the following:

$$\mathbf{U_k} = \left[u_k[0], u_k[1], u_k[2],\ldots\ldots, u_k[t_f]\right]^T$$
$$\mathbf{Y_k} = \left[y_k[0], y_k[1], y_k[2],\ldots\ldots, y_k[t_f]\right]^T$$
$$\mathbf{Y_d} = \left[y_d[0], y_d[1], y_d[2],\ldots\ldots, y_d[t_f]\right]^T \quad (2)$$

$\mathbf{Y_d}$ is the desired trajectory. This allows us to represent the dynamic system in $\mathbb{R}^1$ as a static system in $\mathbb{R}^{t_f+1}$ in the following way:

$$\mathbf{Y_k} = \mathbf{G U_k}$$
$$\mathbf{G} = \begin{bmatrix} g_0 & 0 & 0 & \cdots & 0 \\ g_1 & g_0 & 0 & \cdots & 0 \\ g_2 & g_1 & g_0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ g_N & g_{N-1} & g_{N-2} & \cdots & g_0 \end{bmatrix} \quad (3)$$

$\mathbf{G}$ is the matrix of impulse response coefficients ($g_0 = D$, $g_j = \mathbf{CA}^{j-1}\mathbf{B}$ for $j \geq 1$) of the system. Standard ILC update algorithms take the following form [2]:

$$\mathbf{U_{k+1}} = \mathbf{U_k} + \mathbf{L E_k}$$
$$\mathbf{E_k} = \mathbf{Y_d} - \mathbf{Y_k} \quad (4)$$

The ILC design problem is to find a learning gain matrix $\mathbf{L}$ such that trajectory tracking error converges to zero, as more trials are executed. One way to guarantee convergence, given the state-space model of the system, is to use a norm-optimal ILC algorithm [5], which updates the input at the k+1th iteration ($\mathbf{U_{k+1}}$) such that it minimizes a weighted sum of the trajectory tracking error and the controller effort, as shown in equation (5).

$$J_{k+1} = \frac{1}{2}\mathbf{E_{k+1}^T Q E_{k+1}} + \frac{1}{2}(\mathbf{U_{k+1}} - \mathbf{U_k})^T \mathbf{R} (\mathbf{U_{k+1}} - \mathbf{U_k}) \quad (5)$$

$\mathbf{Q}$ and $\mathbf{R}$ are positive definite, weighing matrices for tracking error and controller effort respectively. Calculating the derivative of J_{k+1} with respect to $\mathbf{U_{k+1}}$ and setting it to zero and solving for $\mathbf{L}$ results in equation (6).

$$\mathbf{U_{k+1}} = \mathbf{U_k} + \mathbf{L E_k}$$
$$\mathbf{L} = \left(\mathbf{G^T Q G} + \mathbf{R}\right)^{-1} \mathbf{G^T Q} \quad (6)$$

ILC is a feedback controller in the trial (batch) domain but is a feedforward controller in the time domain. This means that the above ILC algorithm can only handle repetitive disturbances (across trials) but is not equipped to handle non-repetitive disturbances (within a trial). To mitigate this, we use a plug-in-type ILC that acts on the closed-loop system consisting of the process and the feedback controller. The learning gain matrix $\mathbf{L}$ is now designed based on the matrix of impulse responses of the closed-loop system.

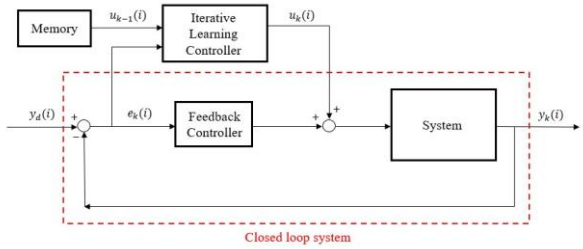


Figure 1. Plug-in-type ILC

B. Gaussian Process (GP)

A GP is completely specified by its mean function $m(\mathbf{x})$ and covariance function $k(\mathbf{x}, \mathbf{x}')$ where $\mathbf{x}$ is the vector of regressors, of dimension D. The Gaussian process is written as follows:

$$f(\mathbf{x}) \sim \mathcal{N}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}'))$$
$$m(\mathbf{x}) = E[f(\mathbf{x})]$$
$$k(\mathbf{x}, \mathbf{x}') = E\left[\left(f(\mathbf{x}) - m(\mathbf{x})\right)\left(f(\mathbf{x}') - m(\mathbf{x}')\right)\right] \quad (7)$$

We typically assume a zero mean or constant mean function ($m(\mathbf{x}) = \bar{\mathbf{m}}$, constant matrix of the same dimensions as $\mathbf{x}$) when we have no knowledge about the system. The covariance function defines the similarity or nearness of the function values, since two inputs that are close to each other are likely to have similar function values [6]. A common choice of covariance function is the Squared Exponential kernel with Automatic Relevance Determination (SE-ARD) structure, shown in equation (8).

$$k(\mathbf{x_i}, \mathbf{x_j}) = \sigma_f^2 \exp\left(-\frac{1}{2}(\mathbf{x_i} - \mathbf{x_j})^T \mathbf{\Sigma^{-1}} (\mathbf{x_i} - \mathbf{x_j})\right) + \sigma_n^2 \delta_{ij} \quad (8)$$

The SE-ARD kernel is infinitely differentiable, meaning it produces smooth functions which is desirable in modelling complex data. The vector of hyperparameters of this covariance function is $\boldsymbol{\theta} = \{\sigma_f, \boldsymbol{\Sigma}, \sigma_n\}$. This choice of covariance function is particularly useful as the hyperparameters can be interpreted as follows: σ_f controls the magnitude of covariance, $\boldsymbol{\Sigma}$ is a diagonal matrix of squared characteristic length scale/relative importance of q^{th} component of x_I and x_j vector (q^{th} input), $\sigma_n^2\delta_{ij}$ represents the covariance between outputs due to white noise (δ_{ij} is the Kronecker Delta function) [7]. Given N training datapoints $(\mathbf{X}_{N\times D}, \mathbf{y}_{N\times 1})$, The GP model is trained by tuning the hyperparameters such that we maximize the log-likelihood given by equation (9)

$$\ln\big(p(\mathbf{y}|\,\boldsymbol{\theta}, \mathbf{X})\big) = -\frac{1}{2}\big(\mathbf{y} - \mathbf{m}(\mathbf{X})\big)^T\mathbf{k}^{-1}(\mathbf{X}, \mathbf{X})\big(\mathbf{y} - \mathbf{m}(\mathbf{X})\big) - \frac{1}{2}\ln\big(\det\big(\mathbf{k}(\mathbf{X}, \mathbf{X})\big)\big) - \frac{N}{2}\ln(2\pi) \quad (9)$$

III. Methodology

A. Modelling dynamic systems using GPs

Gaussian processes, usually used to model static non-linearities, can be modified to model dynamic systems if lagged inputs (u) and outputs (y) are used as regressors [7]. The Gaussian process f($\mathbf{x}$) is shown in equation (10)

$$y[i] = f(\mathbf{x[i]}) + v$$
$$\mathbf{x[i]} = [\,y[i-1], y[i-2], \ldots, y[i-n], u[i-d], u[i-d-1], \ldots, u[i-d-m]\,] \quad (10)$$

m and *n* are the lag of inputs and outputs respectively, *d* is the input-output sample delay, $\mathbf{x}$ is the vector of regressors, *v* is white noise and i is the time instant. Unknowns *m, n* and *d* are chosen such that two performance measures - Mean Squared Error (MSE) and Log Predictive Density Error (LPD) are minimized while ensuring that the model remains parsimonious.

$$\text{MSE} = \frac{\sum_{i=1}^{N}\big(y_i - \hat{y}_i\big)^2}{N}$$
$$\text{LPD} = \frac{1}{2}\ln(2\pi) + \sum_{i=1}^{N}\left(\ln\left(\hat{\sigma}_i^2\right) + \frac{\big(y_i - \hat{y}_i\big)^2}{\hat{\sigma}_i^2}\right) \quad (11)$$

$\hat{y}_I$, $\hat{\sigma}_i^2$ are the predicted mean and variance of output corresponding to the i^{th} datapoint, $i \in \{0,1,\cdots, N\}$. LPD is a suitable measure for GPs as it weighs the prediction error more heavily when there is high confidence in the estimate.

B. Forward Adaptive Transfer (FAT) Learning using GPs

Let the original process be the source task (S) and the process under new operating conditions be the target task (T). Typically, the original process is well-studied with a large number (N_S) of experimental datapoints ($\mathbf{X_S}, \mathbf{y_S}$) which allows us to train a very reliable GP with a constant mean $\overline{\mathbf{m_S}}$ and a SE-ARD kernel with hyperparameters $\boldsymbol{\theta_S}=\{\sigma_{fS}, \boldsymbol{\Sigma}_S, \sigma_{nS}\}$. The process under new operating conditions has limited ($N_T \ll N_S$) experimental data ($\mathbf{X_T}, \mathbf{y_T}$) which is not suitable for building a reliable GP model. For some scale-ups of systems (e.g initial concentration, cross sectional area of a reactor) the order of the dynamics and delay of the system will remain the same. Hence, we can say that *m*, *n* and *d* also remain the same for such scale-ups. The mean $\overline{\mathbf{m_T}}$ and hyperparameters $\boldsymbol{\theta_T} = \{\sigma_{fT}, \boldsymbol{\Sigma}_T, \sigma_{nT}\}$ of the GP for the target task are unknown. To transfer relevant knowledge from the source task to the target task, both tasks are assumed to be jointly distributed as follows:

$$\begin{bmatrix}\mathbf{y_T}\\ \mathbf{y_S}\end{bmatrix} \sim N\left(\begin{bmatrix}\overline{\mathbf{m_T}}\\ \overline{\mathbf{m_S}}\end{bmatrix}, \mathbf{K_{joint}}\right),\ \mathbf{K_{joint}} = \begin{bmatrix}\mathbf{K_{TT}} & \mathbf{K_{TS}}\\ \mathbf{K_{ST}} & \mathbf{K_{SS}}\end{bmatrix} \quad (12)$$

$\mathbf{K_{TT}} = k(\mathbf{X_T}, \mathbf{X_T}; \boldsymbol{\theta_T}) \in \mathbb{R}^{N_T\times N_T}$, $\mathbf{K_{SS}} = k(\mathbf{X_S}, \mathbf{X_S}; \boldsymbol{\theta_S}) \in \mathbb{R}^{N_S\times N_S}$ are SE-ARD kernels for the target data and source data respectively. The cross-covariance matrix $\mathbf{K_{TS}}$ must account for the similarity between the tasks, while maintaining boundaries between them. This is done by defining the cross-covariance matrix as follows:

$$\mathbf{K_{TS}} = \lambda\mathbf{K_{cross}} = \mathbf{K_{ST}^T} \quad (13)$$

$|\lambda| \le 1$, is an additional hyperparameter, that captures the correlation between the source and target task, and it is to be learned along with $\overline{\mathbf{m_T}}$ and $\boldsymbol{\theta_T}$ [8]. A value of λ close to zero signifies that the source and target task are highly dissimilar with no scope for knowledge transfer. The GP model for the target task must then be learned from scratch, possibly after extensive data collection. $\mathbf{K_{cross}}$ must be chosen such that $\mathbf{K_{joint}}$ is positive semidefinite since it is a kernel covariance matrix. We can construct a valid cross-covariance kernel through a convolution of the source and target kernels [8],[9] as shown in equation (14).

$$k_{cross}\big(\mathbf{x_{T,i}}, \mathbf{x_{S,j}}\big) = \sigma_{f,cross}^2 \exp\left[\big(\mathbf{x_{T,i}} - \mathbf{x_{S,j}}\big)^T\left(\frac{\boldsymbol{\Sigma_T} + \boldsymbol{\Sigma_S}}{2}\right)^{-1}\big(\mathbf{x_{T,i}} - \mathbf{x_{S,j}}\big)\right]$$

$$\sigma_{f,cross}^2 = 2^{\frac{D}{2}}\sqrt{\sigma_{fT}^2\sigma_{fS}^2}\,\frac{|\Sigma_T|^{\frac{1}{4}}\,|\Sigma_S|^{\frac{1}{4}}}{|\Sigma_T + |\Sigma_S|^{\frac{1}{2}}},\ D = m + n$$

$$\mathbf{K_{cross}} = \mathbf{k_{cross}}(\mathbf{X_T}, \mathbf{X_S}) \quad (14)$$

The unknown hyperparameters $\boldsymbol{\theta_U} = \{\boldsymbol{\lambda}, \overline{\mathbf{m_T}}, \boldsymbol{\theta_T}\}$ are tuned to maximize the log marginal likelihood of only the target response variables given the source data. We use the Bayes rule on the distribution given in equation (12) to get the marginal distribution of the target task outputs, conditioned on the source data as follows:

$$p\big(\mathbf{y_T}\big|\,\mathbf{X_T}, \mathbf{y_S}, \mathbf{X_S}, \boldsymbol{\theta_S}, \boldsymbol{\theta_T}\big) = \mathcal{N}(\boldsymbol{\mu_{T|S}}, \mathbf{K_{T|S}})$$

$$\boldsymbol{\mu_{T|S}} = \overline{\mathbf{m_T}} + \mathbf{K_{TS}}\mathbf{K_{SS}^{-1}}\big(\mathbf{y_S} - \mathbf{m_S}\big)$$

$$\mathbf{K_{T|S}} = \mathbf{K_{TT}} - \mathbf{K_{TS}}\mathbf{K_{SS}^{-1}}\mathbf{K_{ST}} \quad (15)$$

Maximizing the log-likelihood of $p\big(\mathbf{y_T}\big|\,\mathbf{X_T}, \mathbf{y_S}, \mathbf{X_S}, \boldsymbol{\theta_S}, \boldsymbol{\theta_T}\big)$ with respect to $\boldsymbol{\theta_U}$ as described in Section II.B tunes the hyperparameters $\boldsymbol{\theta_U}$ which can then be used to predict a target task response y_T' for any new input $\mathbf{x_T'}$. The joint distribution of y_T' with $\mathbf{y_T}$ and $\mathbf{y_S}$ can be written as shown in equation (16) and (17) where $\overline{m_T'}$, $\overline{\mathbf{m_T}}$, $\overline{\mathbf{m_S}}$ are of appropriate dimensions.

$$\begin{bmatrix} y_T' \\ \mathbf{y_T} \\ \mathbf{y_S} \end{bmatrix} \sim N\left(\begin{bmatrix} \overline{m_T'} \\ \overline{\mathbf{m_T}} \\ \overline{\mathbf{m_S}} \end{bmatrix}, \mathbf{K}_{\text{joint, new}} \right)$$

$$\mathbf{K}_{\text{joint, new}} = \begin{bmatrix} k_T(\mathbf{x_T'}, \mathbf{x_T'}) & \mathbf{k_T}(\mathbf{x_T'}, \mathbf{X_T}) & \mathbf{k_{TS}}(\mathbf{x_T'}, \mathbf{X_S}) \\ \mathbf{k_T}(\mathbf{X_T}, \mathbf{x_T'}) & \mathbf{K_{TT}} & \mathbf{K_{TS}} \\ \mathbf{k_{ST}}(\mathbf{X_S}, \mathbf{x_T'}) & \mathbf{K_{ST}} & \mathbf{K_{SS}} \end{bmatrix} \quad (16)$$

$$k_T(\mathbf{x_T'}, \mathbf{x_T'}) = k(\mathbf{x_T'}, \mathbf{x_T'}; \boldsymbol{\theta_T})$$
$$\mathbf{k_T}(\mathbf{x_T'}, \mathbf{X_T}) = \mathbf{k}(\mathbf{x_T'}, \mathbf{X_T}; \boldsymbol{\theta_T}) = \mathbf{k_T}(\mathbf{X_T}, \mathbf{x_T'})$$
$$\mathbf{k_{TS}}(\mathbf{x_T'}, \mathbf{X_S}) = \lambda \mathbf{k_{cross}}(\mathbf{x_T'}, \mathbf{X_S}; \lambda, \boldsymbol{\theta_T}, \boldsymbol{\theta_S}) = \mathbf{k_{ST}}(\mathbf{X_S}, \mathbf{x_T'}) \quad (17)$$

The prediction for y_T' is given by its conditional distribution shown in equation (18). This result for the prediction of a single output can be extended to the case of multiple output predictions.

$$p(\mathbf{y_T'} | \mathbf{x_T'}, \mathbf{y_T}, \mathbf{X_T}, \mathbf{y_S}, \mathbf{X_S}, \boldsymbol{\theta_U}) = \mathcal{N}(\mu_T', K_T')$$
$$\mu_T' = \overline{m_T'} + \mathbf{K_{cross,new}} \mathbf{K_{joint}^{-1}} \left(\begin{bmatrix} \mathbf{y_S} \\ \mathbf{y_T} \end{bmatrix} - \begin{bmatrix} \overline{\mathbf{m_S}} \\ \overline{\mathbf{m_T}} \end{bmatrix} \right)$$
$$K_T' = k_T(\mathbf{x_T'}, \mathbf{x_T'}) - \mathbf{K_{cross,new}} \mathbf{K_{joint}^{-1}} \mathbf{K_{cross,new}^{T}}$$
$$\mathbf{K_{cross,new}} = \begin{bmatrix} k_T(\mathbf{x_T'}, \mathbf{X_T}) & k_{TS}(\mathbf{x_T'}, \mathbf{X_S}) \end{bmatrix} \quad (18)$$

C. ILC for Target System

The updated GP model for the new dynamical system, available through FAT, can be used for re-tuning the ILC in two different ways. In the first approach, one would directly use the predictions and optimally design ILC moves for the next trial without having to tune a learning matrix. Alternatively, we derive a state-space or IR coefficient model via a system identification approach using predictions from the GP model on a freshly designed input to be used to tune the learning gain matrix **L** for ILC. We follow the second approach in this work.

An Inverse Repeat Sequence (IRS) is the recommended input signal for nonlinear system identification [11], which is obtained by taking a Pseudorandom Binary Sequence and negating every other input in the sequence. The predictions from the GP model for a freshly designed IRS input are used to identify an open-loop state-space model through the *n4sid* algorithm. Then the *loopsens* function in MATLAB is used to get a closed-loop state-space model with a feedback controller (Plant times input-to-plant sensitivity function) following which the optimal learning gain matrix for plug-in-type ILC is updated using equation (6), as explained in Section II.A.

IV. Simulation and Results

The above discussed method is demonstrated for a benchmark batch reactor model developed by Cott and Macchietto [10], modelled in MATLAB and SIMULINK. Transfer learning in the GP framework is used to accelerate model development & ILC when the reactor is to be scaled up.

A. Process Description

A liquid phase reaction system is considered, in which two exothermic reactions are modelled:

$$A + B \rightarrow C$$

$$A + C \rightarrow D \quad (19)$$

The aim is to control the reactor temperature such that it follows a pre-determined trajectory, by manipulating the jacket inlet temperature. The desired trajectory shown in equation (20) for the reactor temperature consists of a temperature ramp phase, hold phase, and then a cooling phase.

$$y_d(t) = \begin{cases} T_{r0} + \left(\dfrac{T_h - T_{r0}}{t_R} \right) t & t \le t_r \\ T_h & t > t_r \text{ and } t \le t_f - t_r \\ T_h - \left(\dfrac{T_h - T_{r0}}{t_R} \right) (t - (t_f - t_r)) & t > t_f - t_r \end{cases} \quad (20)$$

Here, initial temperature of reactor (T_{r0}) = 20°C, hold temperature (T_h) = 90°C, ramp-time (t_r) = 40 min, trial length (t_f) = 200 min. The control strategy adopted is a plug-in-type ILC which acts on the closed-loop system with an aggressively tuned discrete-time PI controller shown in equation (21), with sampling time (T_s) = 0.2 min

$$P + \frac{I \cdot T_s}{z\text{-}1}, \; P = 1.327, I = 0.141 \quad (21)$$

B. ILC for original reactor

We assume the reactor is initially operated at conditions specified in [10], and that an open-loop state-space model from sufficient data is available. The closed-loop state space model is obtained using the *loopsens* function in MATLAB. Choosing **Q = I** and **R = 0.1 I**, the learning gain matrix **L** for plug-in-type ILC is designed by following the method described in Section II.A, after which ILC can be performed. Figure 2 plots the objective J_{k+1} defined in equation (5) for each trial k and shows that the reactor temperature converges to the desired trajectory within 10 trials

C. Scaling the reactor

Suppose the initial amounts of reactants are scaled up by a factor of 25 and the geometry of reactor and cooling jacket is scaled up by a factor of 1000. Figure 3 indicates that the reactor dynamics changes. If the same learning gain is used for ILC of the scaled reactor, the tracking error does not converge to zero even after multiple trials. Figure 7 plots the

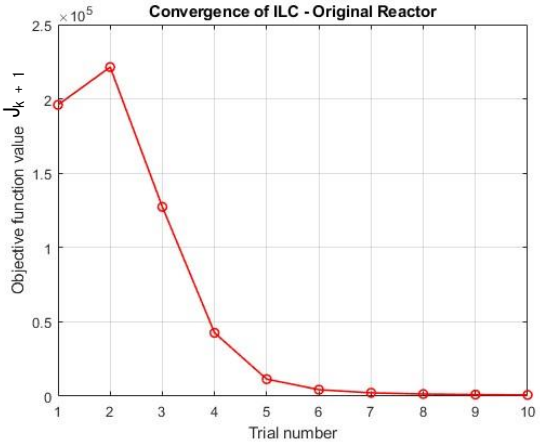


Figure 2. Convergence of ILC for the original reactor

objective J_{k+1} for each trial k (in comparison with that from the proposed method), which is a measure of tracking error. Therefore, ILC re-tuning is required for the scaled reactor.

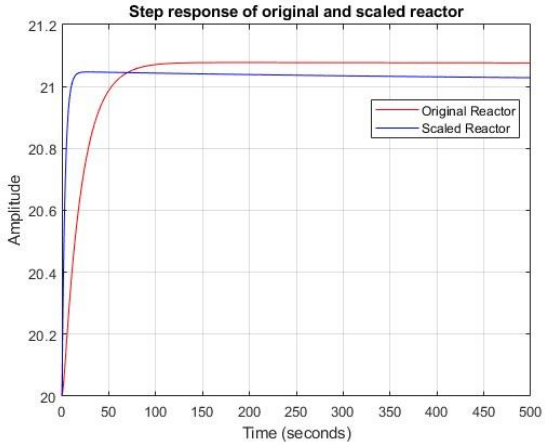


Figure 3. Comparison of step response of the original and scaled reactor

D. Transfer learning using surrogate GP model

We assume that a well-trained model for the original reactor from sufficient data is available. This allows us to build a reliable GP for the original reactor (source task S) with $N_S = 1500$ IRS input-output datapoints. The regressors for the GP model can be found using the method detailed in Section III.A. The model has 3 parameters ($\overline{m_S}$, σ_f, σ_n) by default with one additional parameter per regressor, for a total of $m+n+3$ parameters. Table 1 shows that $n = 2$, $m = 0$, $d = 1$ are the best choices for lags and delay. Therefore, the GP model for the original reactor (GP_{org}) is of the form:

$$y[i] = f(y[i-1], y[i-2], u[i-1]) + v$$
$$f(\cdot) \sim GP_{org} \quad (22)$$

TABLE 1. GP MODEL PERFORMANCE MEASURES FOR DIFFERENT LAGS AND DELAYS

Output Lag (n)	Input Lag (m)	Delay	MSE	LPD	No. of parameters (m+n+3)
1	0	0	0.0240	0.5699	4
1	0	1	0.0237	0.5679	4
2	0	0	0.0265	0.3018	5
1	1	0	0.0246	0.4906	5
2	**0**	**1**	**0.0270**	**0.2974**	**5**
2	1	0	14.2769	4.0821	6

The scaled reactor is the target task *T*. We assume that very limited experimental data of $N_T = 150$ IRS input-output datapoints are available for the scaled reactor. Since the scaled reactor is governed by the same mechanisms as the original reactor, we can say that *m*, *n* and *d* remain the same. We follow the procedure detailed in Section III.B to build a surrogate GP model for the scaled reactor (GP_{sca}^{FAT}). For comparison, we make predictions on fresh validation data from the scaled reactor using 3 GP models - a) GP model trained using only the limited scaled reactor data (GP_{sca}), b) GP model for the original reactor (GP_{org}), c) Forward Adaptive Transfer learning-based GP model (GP_{sca}^{FAT}).

The results are shown in Figures 4-6. The grey region in the figures is the 95% confidence interval for the GP predictions.

a) Figure 4 shows that GP_{sca} performs poorly on the scaled reactor validation data. This is due to the lack of sufficient excitation of the system, a consequence of limited training data.
b) Figure 5 shows that GP_{org} performs better than GP_{sca}, but is still unsatisfactory. Its predictions are good in the regions where the scaled reactor output is within the range of the original reactor output.
c) The limitations of the other two GPs are overcome by GP_{sca}^{FAT} as seen in Figure 6. It combines the output range information from the limited scaled reactor data with the extensive input excitation data from the original reactor to give excellent predictions.

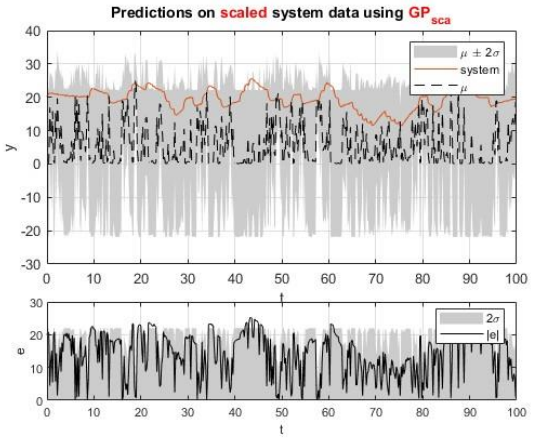


Figure 4. GP_{sca} predictions on validation data of scaled reactor

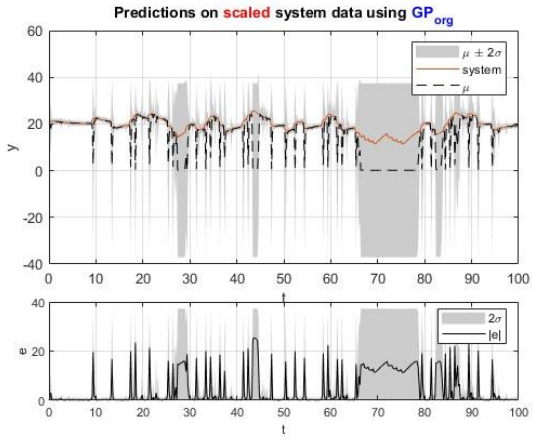


Figure 5. GP_{org} predictions on validation data of scaled reactor

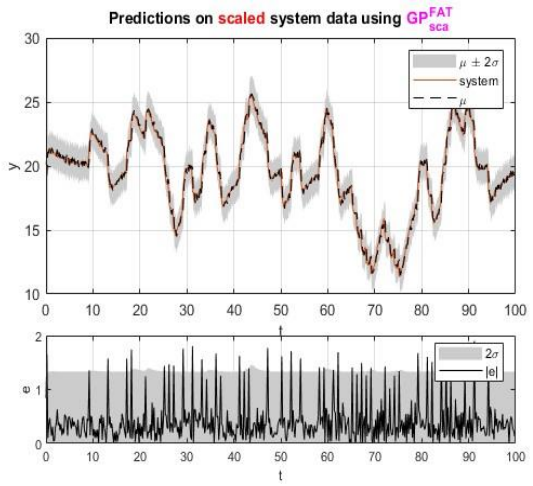


Figure 6. GP_{sca}^{FAT} predictions on validation data of scaled reactor

Table 2 compares the GPs using standard performance measures described in Section III.A

TABLE 2. MODEL PERFORMANCE MEASURES FOR DIFFERENT GPS

Model	MSE	LPD
GP_{sca}	243.81	4.217
GP_{org}	62.94	2.026
GP_{sca}^{FAT}	0.0265	0.3018

E. ILC for scaled reactor

The method described in Section III.C is used to re-tune ILC for the scaled reactor, given at least 500 fresh IRS inputs and the updated model GP_{sca}^{FAT}. From Figure 7, we see that the reactor temperature converges to the desired trajectory within 10 trials.

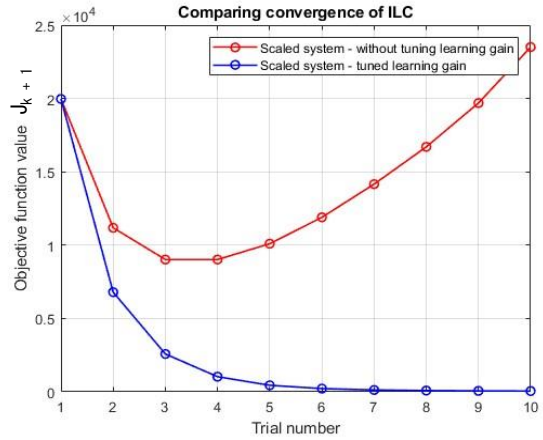


Figure 7. Convergence of ILC for the scaled reactor

V. CONCLUSIONS

In this paper a method to improve ILC for a batch process under new operating conditions is presented. This is achieved by extending Forward Adaptive Transfer Learning using Gaussian Processes (FAT-GP) to dynamic systems. It is demonstrated that using very limited data of the process in new operating conditions (scale up of the process), and sufficient data for the existing (original) process we can transfer relevant knowledge between processes to build a reliable model for the process under new operating conditions. By choosing an appropriate mean and covariance function for the GP, this method can be extended to different classes of systems. The re-tuned ILC with minimal experimental data not only shows excellent performance but also highlights the value of TL in significantly reducing the number of trials required for a fresh design of ILC. A future direction of interest is along the lines of directly using the TL-based FAT-GP instead of deriving a linear model.

REFERENCES

[1] H.-S. Ahn, Y. Chen, and K. L. Moore, "Iterative learning control: Brief survey and categorization," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 37, no. 6, pp. 1099–1121, 2007.

[2] 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*, IEEE, 2006, pp. 2352–2357.

[3] N. Aziz and I. M. Mujtaba, "Optimal operation policies in batch reactors," *Chemical Engineering Journal*, vol. 85, no. 2–3, pp. 313–325, 2002.

[4] C. Xu, K. Yang, X. Chen, X. Luo, and H. Yu, "Transfer Learning for Regression through Adaptive Gaussian Process," in *2022 IEEE 34th International Conference on Tools with Artificial Intelligence (ICTAI)*, IEEE, 2022, pp. 42–47.

[5] N. Amann, D. H. Owens, and E. Rogers, "Iterative learning control for discrete-time systems with exponential rate of convergence," *IEE Proceedings-Control Theory and Applications*, vol. 143, no. 2, pp. 217–224, 1996.

[6] C. K. I. Williams and C. E. Rasmussen, *Gaussian processes for machine learning*, vol. 2, no. 3. MIT press Cambridge, MA, 2006.

[7] J. Kocijan, *Modelling and control of dynamic systems using Gaussian process models*. Springer, 2016.

[8] N. Wagle and E. W. Frew, "Forward adaptive transfer of Gaussian process regression," *Journal of Aerospace Information Systems*, vol. 14, no. 4, pp. 214–231, 2017.

[9] A. Melkumyan and F. Ramos, "Multi-kernel Gaussian processes," in *Twenty-second international joint conference on artificial intelligence*, 2011.

[10] B. J. Cott and S. Macchietto, "Temperature control of exothermic batch reactors using generic model control," *Ind Eng Chem Res*, vol. 28, no. 8, pp. 1177–1184, 1989.

[11] R. Srinivasan and R. Rengaswamy, "Use of inverse repeat sequence (IRS) for identification in chemical process systems," *Ind Eng Chem Res*, vol. 38, no. 9, pp. 3420–3429, 1999.

[12] S.J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22:1345–1359, 2009.

[13] K. Weiss, T.M. Khoshgoftaar and D. Wang. A survey of transfer learning, 3(9). https://doi.org/10.1186/s40537-016-0043-6