

# Discussions on Inverse Kinematics based on Levenberg-Marquardt Method and Model-Free Adaptive (Predictive) Control

Feilong Zhang

**Abstract**—In this brief, the current robust numerical solution to the inverse kinematics based on Levenberg-Marquardt (LM) method is reanalyzed through control theory instead of numerical method. Compared to current works, the stability of computation and convergence performance of computational error are analyzed much more clearly by analyzing the control performance of the corrected model-free adaptive control (MFAC). Then mainly motivated by minimizing the predictive tracking error, this study suggests a new method of model-free adaptive predictive control (MFAPC) to solve the inverse kinematic problem. At last, we applied the MFAPC as a controller for the robotic kinematic problem. It not only shows an excellent control performance but also efficiently acquires the solution to inverse kinematics.

**Index Terms**—inverse kinematics, Levenberg-Marquardt, model-free adaptive control (MFAC).

## I. INTRODUCTION

There have been tremendous works to solve the inverse kinematics problem [1]-[5]. Among of them, the numerical inverse kinematic solution based on LM [6], [7] method in [1] showed the superior robustness and convergent performance of the computation. It has been utilized in the Robot Toolbox of MATLAB to address the issue of kinematic singularity problem and shows an application prospect in robot controller design. However, this method was just analyzed in numerical method, and did not clearly show the quantitative relationship between the convergence performance of computational error and the damping factor ( $\lambda$  in this brief). To this end, we extend the restricted assumptions of the current compact form MFAC [8], [9], and then interestingly find that the corrected MFAC is exactly the numerical method in [1]. Consequently, it is possible for us to analyze the stability and convergence of computation more clearly by the analysis of the system control performance, such as closed-loop function and convergence of tracking error, rather than by the analysis of numerical method in [1].

Then motivated by minimizing the predictive tracking error and some underlying advantages presented in [c], this study

Manuscript received Dec 3, 2020. This work was supported in part by the XXXXXXXXXXXX.

Feilong Zhang is with the State Key Laboratory of Robotics, Shenyang Institute of Automation, Chinese Academy of Sciences, Shenyang 110016, China (e-mail: zhangfeilong@sia.cn).

proposes the MFAPC and analyzes the system performance by the similar way in MFAC. One main merit in comparison to MFAC is that MFAPC can utilize more desired trajectories in the future time, which helps the robot avoid being concentrated with the isolated singular points in generated path. Furthermore, the outputs of the system and controller will be smoother.

Another motivation of this brief is to point out the crucial problem on MFAC in [8]-[12]. They have showed that the tracking error of the system controlled by MFAC converges to zero on the condition that  $\lambda$ , i.e., the damping factor in [1], is large enough. This conclusion contradicts with the analysis results in [1] and this brief. For more deficiencies of current works, please refer to [a]-[c].

The rest of the paper is organized as follows. In Section II, the modified compact form equivalent dynamic linearization model (EDLM) is presented for the description of robot kinematics. Based on this, the MFAC controller design and performance analysis are presented. Then we review its application in the Robot Toolbox of MATLAB. Similar to Section III, the MFAPC design and its performance analysis are presented in Section IV, and then how to address inverse kinematic problem by MFAPC is shown. In Section V, we applied MFAPC as a controller for the robotic kinematic control in experiments. It not only shows an excellent control performance but also efficiently acquires the solution to inverse kinematics.

## II. EQUIVALENT DYNAMIC LINEARIZATION MODEL FOR MULTIVARIABLE NONLINEAR SYSTEMS

In this section, the robotic kinematic is described by the corrected compact form equivalent dynamic linearization model (EDLM) which is used for controller design and performance analysis.

Consider the kinematic function of robotic manipulator as

$$\mathbf{y}(k+1) = \mathbf{F}(\mathbf{q}(k)) \quad (1)$$

where  $\mathbf{q}(k)$  represents the system input vector (joint angle vector), and  $\mathbf{y}(k)$  represents the system output vector (the position and orientation of robot in task space). The dimension of  $\mathbf{y}(k)$  and  $\mathbf{q}(k)$  are  $M_y$  and  $M_u$ , respectively.  $\mathbf{F}(\bullet)$  represents the kinematic transformation from the joint angle vector  $\mathbf{q}(k)$  to the frame description vector  $\mathbf{y}(k)$  in Cartesian-space.

According to [b], we make a local approximate linearization of (1) to have the compact form EDLM, as shown by

$$\Delta\mathbf{y}(k+1) = \mathbf{J}(k)\Delta\mathbf{q}(k) \quad (2)$$

where  $\mathbf{J}(k) = \begin{bmatrix} \phi_{11}(k) & \phi_{12}(k) & \cdots & \phi_{1M_u}(k) \\ \phi_{21}(k) & \phi_{22}(k) & \cdots & \phi_{2M_u}(k) \\ \vdots & \vdots & \vdots & \vdots \\ \phi_{M_y 1}(k) & \phi_{M_y 2}(k) & \cdots & \phi_{M_y M_u}(k) \end{bmatrix} \in \mathbb{R}^{M_y \times M_u}$ ,

$\Delta = 1 - z^{-1}$  in which  $z^{-1}$  is the backward shift operator.

### III. MFAC DESIGN AND STABILITY ANALYSIS

This section gives the design and stability analysis methods for MFAC. Then its application in robot inverse kinematic is reviewed.

#### A. Design of Model-Free Adaptive Control

We rewrite (2) into (3).

$$\mathbf{y}(k+1) = \mathbf{y}(k) + \mathbf{J}(k)\Delta\mathbf{q}(k) \quad (3)$$

A control input criterion function is given as:

$$J = \left[ \mathbf{y}^*(k+1) - \mathbf{y}(k+1) \right]^T \left[ \mathbf{y}^*(k+1) - \mathbf{y}(k+1) \right] + \Delta\mathbf{q}^T(k)\lambda\Delta\mathbf{q}(k) \quad (4)$$

where  $\lambda = \text{dig}(\lambda_1, \dots, \lambda_{M_u})$  is the weighted diagonal matrix (damping factor matrix in [1]), and we suppose  $\lambda_i$  ( $i=1, \dots, M_u$ ) are equal to  $\lambda$  in accordance with [1] and [9];  $\mathbf{y}^*(k+1) = [y_1^*(k+1), \dots, y_{M_y}^*(k+1)]^T$  is the desired trajectory vector.

Substituting (3) into (4) and solving the equation  $\partial J / \partial \Delta\mathbf{u}(k) = 0$  yield the MFAC controller:

$$[\mathbf{J}^T(k)\mathbf{J}(k) + \lambda]\Delta\mathbf{q}(k) = \mathbf{J}^T(k)[(\mathbf{y}^*(k+1) - \mathbf{y}(k))] \quad (5)$$

*Remark 1:* If  $\lambda = \mathbf{0}$ , (5) will be the optimal solution for the tracking error control. It was also shown in our previous work [a] for SISO systems.

#### B. Stability Analysis of MFAC

This section provides the performance analysis of MFAC.

According to (2) and (5), the closed-loop system equation can be expressed by

$$\begin{aligned} & [\Delta\mathbf{I}_{M_y} + z^{-1}\mathbf{J}(k)[\mathbf{J}^T(k)\mathbf{J}(k) + \lambda]^{-1}\mathbf{J}^T(k)]\mathbf{y}(k) \\ &= \mathbf{J}(k)[\mathbf{J}^T(k)\mathbf{J}(k) + \lambda]^{-1}\mathbf{J}^T(k)\mathbf{y}^*(k) \end{aligned} \quad (6)$$

where  $\mathbf{I}_*$  represents  $\bullet$ -dimensional identity matrix.

We assume  $\text{rank}[Z(z^{-1})] = M_y$  ( $M_u \geq M_y$ ). By tuning  $\lambda$ , we can obtain inequation such that

$$\mathbf{T} = \Delta\mathbf{I}_{M_y} + z^{-1}\mathbf{J}(k)[\mathbf{J}^T(k)\mathbf{J}(k) + \lambda]^{-1}\mathbf{J}^T(k) \neq \mathbf{0}, \quad |z| > 1 \quad (7)$$

, which determines the poles of the system. Then (6) guarantees the system stability according to [15]-[16].

Herein, the singular value decomposition is conducted to  $\mathbf{J}(k)$  as follow:

$$\mathbf{J}(k) = \mathbf{U}\Sigma\mathbf{V}^T \quad (8)$$

where  $\Sigma = [\Sigma_1 \ \mathbf{0}]_{M_y \times M_u}$ , and  $\Sigma_1 = \text{diag}(\sigma_1, \dots, \sigma_{M_y})$  is a matrix in which the singular values are diagonally assigned  $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_{M_y}$ ;  $\mathbf{U}$  and  $\mathbf{V}$  are orthonormal matrices.

Moreover, from (7), we have the poles of the system:

$$\begin{aligned} \mathbf{z} &= \mathbf{I}_{M_y} - \mathbf{J}(k)[\mathbf{J}^T(k)\mathbf{J}(k) + \lambda]^{-1}\mathbf{J}^T(k) \\ &= \mathbf{I}_{M_y} - \mathbf{U}\Sigma[\Sigma^T\Sigma + \lambda]^{-1}\Sigma^T\mathbf{U}^T \\ &= \mathbf{I}_{M_y} - \mathbf{U}\Lambda(\mathbf{U}\Lambda)^T \\ &= \mathbf{U}(\mathbf{I}_{M_y} - \Lambda\Lambda^T)\mathbf{U}^T \end{aligned} \quad (9)$$

where  $\Lambda = \text{dig}(\frac{\sigma_1}{\sqrt{\lambda + \sigma_1^2}}, \dots, \frac{\sigma_{M_y}}{\sqrt{\lambda + \sigma_{M_y}^2}})$ .

It is obvious that all the poles of the system will be placed in unit circle under the condition that  $\lambda$  is sufficiently large. Moreover, the robustness or stability of the system improves as  $\lambda$  increases, including the situation that the robot approaches the singular points. Nevertheless, the convergence of tracking error will be worse, which is demonstrated as follow:

Suppose the desired trajectory is  $k^p \cdot \mathbf{E}_{M_y \times 1}$  ( $\mathbf{E}_{M_y \times 1} = [1, \dots, 1]^T$ ,  $p = 1, 2, \dots$ ), the static error (steady-state error) will be

$$\begin{aligned} & \lim_{k \rightarrow \infty} \mathbf{e}(k) \\ &= \lim_{z \rightarrow 1} \frac{z-1}{z} (\mathbf{I}_{M_y} - \mathbf{T}^{-1}\mathbf{J}(k)[\mathbf{J}^T(k)\mathbf{J}(k) + \lambda]^{-1}\mathbf{J}^T(k)) \frac{C(z)}{(z-1)^{p+1}} \mathbf{E}_{M_y \times 1} \\ &= \lim_{z \rightarrow 1} \mathbf{T}^{-1}(\mathbf{I}_{M_y} - \mathbf{J}(k)[\mathbf{J}^T(k)\mathbf{J}(k) + \lambda]^{-1}\mathbf{J}^T(k)) \frac{C(z)}{(z-1)^{p-1}} \mathbf{E}_{M_y \times 1} \\ &= \lim_{z \rightarrow 1} \mathbf{T}^{-1}\mathbf{U}[\mathbf{I}_{M_y} - \Lambda^2]\mathbf{U}^T \frac{C(z)}{(z-1)^{p-1}} \mathbf{E}_{M_y \times 1} \\ &= \lim_{z \rightarrow 1} \mathbf{T}^{-1}\mathbf{U} \text{dig}(\frac{\lambda}{\lambda + \sigma_1^2}, \dots, \frac{\lambda}{\lambda + \sigma_{M_y}^2}) \mathbf{U}^T \frac{C(z)}{(z-1)^{p-1}} \mathbf{E}_{M_y \times 1} \end{aligned} \quad (10)$$

where  $Z(k^p \mathbf{E}_{M_y \times 1}) = \frac{C(z)}{(z-1)^{p+1}} \mathbf{E}_{M_y \times 1}$  and  $C(z)$  is the polynomial with the highest power  $p$ ;  $Z(\bullet)$  denotes  $z$ -transformation.

Evidently, the static error and  $\lambda$  are positively correlated. Further, through  $\lambda = \mathbf{0}$  the steady-state error may be eliminated ( $\lim_{k \rightarrow \infty} \mathbf{e}(k) = \mathbf{0}$ ) and the poles of the system will be also placed in coordinate origin to attain the optimal control effect for the tracking error, which means the fastest convergence speed and no overshoot when the robot does not pass through the singular points.

However, the above conclusion contradicts with the result in [8]-[12]. They have showed that the tracking error of the system controlled by MFAC converges to zero on the condition that  $\lambda$  is large enough.

#### C. The application of MFAC in robot inverse kinematics

As to application, someone may refer to the `ikine.m` of MATLAB Robotic Toolbox. In this part, we will review the theory thereof how to calculate the inverse kinematic solution for the desired position  $\mathbf{y}^*(k+1)$  in task space. For more details and knowledge, please refer to [13]-[14].

Based on (1), the finite  $N$ -step forward iteration is given as

$$\mathbf{y}(k)_{(i)} = \mathbf{F}(\mathbf{q}(k)_{(i-1)}), \quad i=1, \dots, N \quad (11)$$

According to [14], we can easily obtain the Jacobian matrix

$$\mathbf{J}(k)_{(i)} = \mathbf{f}(\mathbf{q}(k)_{(i-1)}), \quad i=1, \dots, N-1 \quad (12)$$

where  $\mathbf{f}(\bullet)$  represents the transformation from the joint angles to the Jacobian matrix.  $i$  represents the step of iteration before the robot moves at the time of  $k+1$  in experiment; The initializations are  $\mathbf{y}(k)_{(0)} = \mathbf{y}(k)$  and  $\mathbf{q}(k)_{(0)} = \mathbf{q}(k)$ . To save the room, the symbol  $\mathbf{J}(k)_{(i)}$  is shorthand for  $\mathbf{J}(k+i|k)$ ,  $\mathbf{y}(k)_{(i)}$  for  $\mathbf{y}(k+i|k)$  and  $\mathbf{u}(k)_{(i)}$  for  $\mathbf{u}(k+i|k)$ , and so on.

Based on (5), we can obtain  $\Delta\mathbf{q}(k)_{(i)}$  through

$$\begin{aligned} & [\mathbf{J}^T(k)_{(i)} \mathbf{J}(k)_{(i)} + \lambda(k)_{(i)}] \Delta\mathbf{q}(k)_{(i)} \\ &= \mathbf{J}^T(k)_{(i)} [(\mathbf{y}^*(k+1) - \mathbf{y}(k)_{(i)})] \end{aligned} \quad (13)$$

where

$$\lambda(k)_{(i+1)} = \begin{cases} a_1 \cdot \lambda(k)_{(i)}, & \text{if } \frac{\|\mathbf{y}^*(k+1) - \mathbf{y}(k)_{(i)}\|_2}{\|\mathbf{y}^*(k+1) - \mathbf{y}(k)_{(i-1)}\|_2} > 1 \\ \lambda(k)_{(i)} / a_2, & \text{if } \frac{\|\mathbf{y}^*(k+1) - \mathbf{y}(k)_{(i)}\|_2}{\|\mathbf{y}^*(k+1) - \mathbf{y}(k)_{(i-1)}\|_2} \leq 1 \end{cases} \quad (14)$$

, in which  $a_1 \geq 1$  and  $a_2 \geq 1$  are the coefficients for  $\lambda(k)_{(i)}$  to make a balance between the convergence speed and the robustness;

Then we have

$$\mathbf{q}(k)_{(i)} = \mathbf{q}(k)_{(i-1)} + \Delta\mathbf{q}(k)_{(i)} \quad (15)$$

Thus, the iteration among (11)-(15) yields our concerned  $\mathbf{y}(k)_{(i)}$  and  $\mathbf{q}(k)_{(i)}$ ,  $i=1, 2, \dots, N$  ( $N \leq N_{up}$ ). The iteration process above will continue until  $N = N_{up}$  or

$$\|\mathbf{y}^*(k+1) - \mathbf{y}(k)_{(N)}\|_2 \leq \delta \quad (16)$$

where  $\delta$  represents the final error tolerance, and  $N_{up}$  represents the maximum number of iterations, and the defaults  $\delta = 10^{-10}$  and  $N_{up} = 500$  are given in ikine.m of MATLAB.

At last, we will send the final iterative inverse kinematic solution (17) to the robot actuator:

$$\mathbf{q}(k) = \mathbf{q}(k-1) + \sum_{i=1}^N \Delta\mathbf{q}(k)_{(i)} \quad (17)$$

The presentation of current numerical solution to inverse kinematic is finished.

*Remark 2:* We can modify the criteria (14) to

$$\lambda(k)_{(i+1)} = \begin{cases} a_1 \cdot \lambda(k)_{(i)}, & \text{if } \|\mathbf{y}^*(k+1) - \mathbf{y}(k)_{(i)}\|_2 > t_1 \\ \lambda(k)_{(i)} / a_2, & \text{if } \|\mathbf{y}^*(k+1) - \mathbf{y}(k)_{(i)}\|_2 \leq t_1 \end{cases} \quad (18)$$

where  $t_1$  represents the adjustment threshold for  $\lambda(k)$ .

Further, it is suggested that when  $\|\mathbf{y}^*(k+1) - \mathbf{y}(k)_{(r)}\|_2$  changing from  $\|\mathbf{y}^*(k+1) - \mathbf{y}(k)_{(r)}\|_2 \leq t_1$  to  $\|\mathbf{y}^*(k+1) - \mathbf{y}(k)_{(r+1)}\|_2 > t_1$  at step  $r$ , we may reset

$\lambda(k)_{(r)} = \lambda(k)_{(0)}$ . And [1] recommends that computation robustly converges with  $\lambda(k)_{(0)} = 0.1l^2 \sim 10^{-3}l^2$ , and  $l$  is the length of a typical link  $l = 0.1 \sim 100[m]$ .

*Remark 3:* Actually, since the desired position  $\mathbf{y}^*(k+1)$  is obtained as the known constant vector in the trajectory planning process [13]-[14], we can choose an appropriate constant matrix  $\lambda(k)_{(i)} \neq \mathbf{0}$  in (13) for easy applications when the robot does not approach the singular points. According to (10), the tracking error still converges when  $p=0$ , however, the convergence speed is sacrificed inevitably. Alternatively, we can use the lookup table method to tune  $\lambda(k)_{(i)}$ . When the Jacobian matrix is ill-conditioned or close to singular, we choose a set of large values of  $\lambda(k)_{(i)}$  for the computational stability, otherwise, we prefer the small values for the faster convergence speed of the computation. One of our preferred lookup table methods like the Table I is to select the  $\lambda(k)_{(i)}$  in accordance with the tracking error which positively correlates with the change from  $\mathbf{J}(k)_{(i)}$  to  $\mathbf{J}(k)_{(i+1)}$  and condition number  $cond(\mathbf{J}(k)_{(i)})$  which normally indicates the degree of ill condition for  $\mathbf{J}(k)_{(i)}$ . The Table I should be determined by trial and error.

TABLE I Matrix  $\lambda$  tune table

$\ \epsilon\ $	Cond	[1, $c_1$ ]	[ $c_1, c_2$ ]	...	[ $c_{n-1}, c_n$ ]
[0, $e_1$ ]	$\mathbf{0}$	$\lambda_{12}$	...	...	$\lambda_{1n}$
[ $e_1, e_2$ ]	$\lambda_{21}$	$\lambda_{22}$	...	...	$\lambda_{2n}$
$\vdots$	$\vdots$	$\vdots$	...	...	$\vdots$
[ $e_{m-1}, e_m$ ]	$\lambda_{m1}$	$\lambda_{m2}$	...	...	$\lambda_{mn}$

*Remark 4:* Now we want to organize the iteration (11)-(17) into one integrated formula.

Based on (3), the finite  $N$ -step forward iteration model is given as

$$\begin{aligned} \mathbf{y}(k)_{(1)} &= \mathbf{y}(k) + \mathbf{J}(k) \Delta\mathbf{q}(k) \\ \mathbf{y}(k)_{(2)} &= \mathbf{y}(k)_{(1)} + \mathbf{J}(k)_{(1)} \Delta\mathbf{q}(k)_{(1)} \\ &= \mathbf{y}(k) + \mathbf{J}(k) \Delta\mathbf{q}(k) + \mathbf{J}(k)_{(1)} \Delta\mathbf{q}(k)_{(1)} \\ &\vdots \\ \mathbf{y}(k)_{(N)} &= \mathbf{y}(k)_{(N-1)} + \mathbf{J}(k)_{(N-1)} \Delta\mathbf{q}(k)_{(N-1)} \\ &= \mathbf{y}(k) + \sum_{i=1}^N \mathbf{J}(k)_{(i-1)} \Delta\mathbf{q}(k)_{(i-1)} \end{aligned} \quad (19)$$

which is the linearization of (11). Herein, we define

$$\Psi(k) = \begin{bmatrix} \mathbf{J}(k) \\ \mathbf{J}(k) & \mathbf{J}(k)_{(1)} \\ \vdots & \ddots & \ddots \\ \mathbf{J}(k) & \cdots & \mathbf{J}(k)_{(N-2)} & \mathbf{J}(k)_{(N-1)} \end{bmatrix}_{\times(N \cdot M_y)},$$

$$\mathbf{E}_{(N \cdot M_y) \times M_y} = [\mathbf{I}_{M_y} \ \cdots \ \mathbf{I}_{M_y}]^T,$$

$$\mathbf{Y}_N(k+1) = \begin{bmatrix} \mathbf{y}(k)_{(1)} \\ \vdots \\ \mathbf{y}(k)_{(N)} \end{bmatrix}_{(N \cdot M_y) \times 1}, \quad \Delta\mathbf{Q}_N(k) = \begin{bmatrix} \Delta\mathbf{q}(k) \\ \vdots \\ \Delta\mathbf{q}(k)_{(N-1)} \end{bmatrix}_{(N \cdot M_u) \times 1}.$$

Then we can rewrite (19) as

$$\mathbf{Y}_N(k+1) = \mathbf{E}_{(N \times M_y)} \mathbf{y}(k) + \boldsymbol{\Psi}(k) \Delta \mathbf{Q}_N(k) \quad (20)$$

Further, we can organize the results calculated by (11)-(16) into one formula

$$\begin{aligned} \Delta \mathbf{Q}_N(k) &= [\boldsymbol{\Psi}^T(k) \boldsymbol{\Psi}(k) + \lambda_{(M_u \times N)}]^{-1} \\ &\cdot \boldsymbol{\Psi}^T(k) [\mathbf{Y}^*(k+1) - \mathbf{E}_{(N \times M_y)} \mathbf{y}(k)] \end{aligned} \quad (21)$$

where  $[\tilde{\mathbf{Y}}^*(k+1)]^T = [(\mathbf{y}^*(k+1))^T, \dots, (\mathbf{y}^*(k+n))^T]$  is the desired system output vector; the diagonal matrix  $\lambda_{(M_u \times N)} = \text{diag}(\lambda(k)_{(1)}, \dots, \lambda(k)_{(N)})$  is obtained from (14);

Obviously, when we substitute (20) into (22), (21) will be the optimal solution of (22).

$$\begin{aligned} J &= [\mathbf{Y}^*(k+1) - \mathbf{Y}_N(k+1)]^T [\mathbf{Y}^*(k+1) - \mathbf{Y}_N(k+1)] \\ &+ \Delta \mathbf{Q}_N^T(k) \lambda_{(M_u \times N)} \Delta \mathbf{Q}_N(k) \end{aligned} \quad (22)$$

Then (17) is rewritten as

$$\mathbf{q}(k) = \mathbf{q}(k-1) + \mathbf{G} \Delta \mathbf{Q}_N(k) \quad (23)$$

where  $\mathbf{G}_{M_u \times (M_u \times N)} = [\mathbf{I}_{M_u} \ \dots \ \mathbf{I}_{M_u}]$ .

Interestingly, both (11) and (19) already show the predictive conception. How about we change (13) into the prediction method to minimize the predictive tracking error instead of (4), with naturally utilization of more future desired trajectory (setting points), predictive model and rolling optimization? Motivated by this idea, the MFAPC is studied in Section IV.

#### IV. MFAPC DESIGN AND STABILITY ANALYSIS

This section gives the design and stability analysis for MFAPC and its application in inverse kinematic of robot.

##### A. Design of Model-Free Adaptive Predictive Control

Based on (3), the finite  $n$ -step forward prediction model is given as

$$\begin{aligned} \mathbf{y}(k+1) &= \mathbf{y}(k) + \mathbf{J}(k) \Delta \mathbf{q}(k) \\ \mathbf{y}(k+2) &= \mathbf{y}(k+1) + \mathbf{J}(k+1) \Delta \mathbf{q}(k+1) \\ &\quad \vdots \\ \mathbf{y}(k+n) &= \mathbf{y}(k+n-1) + \mathbf{J}(k+n-1) \Delta \mathbf{q}(k+n-1) \\ &= \mathbf{y}(k) + \sum_{i=1}^n \mathbf{J}(k+i-1) \Delta \mathbf{q}(k+i-1) \end{aligned} \quad (24)$$

where  $n$  is the given prediction step. Then (24) is rewritten as

$$\mathbf{Y}_n(k+1) = \mathbf{E}_{(n \times M_y)} \mathbf{y}(k) + \bar{\boldsymbol{\Psi}}(k) \Delta \mathbf{Q}(k) \quad (25)$$

where

$$\bar{\boldsymbol{\Psi}}(k) = \begin{bmatrix} \mathbf{J}(k) & & & \\ \mathbf{J}(k) & \mathbf{J}(k+1) & & \\ \vdots & \ddots & \ddots & \\ \mathbf{J}(k) & \cdots & \mathbf{J}(k+n-2) & \mathbf{J}(k+n-1) \end{bmatrix}_{\frac{(n \times M_y)}{(n \times M_u)}},$$

$$\mathbf{Y}_n(k+1) = \begin{bmatrix} \mathbf{y}(k+1) \\ \vdots \\ \mathbf{y}(k+n) \end{bmatrix}_{(n \times M_y) \times 1}, \quad \Delta \mathbf{Q}(k) = \begin{bmatrix} \Delta \mathbf{q}(k) \\ \vdots \\ \Delta \mathbf{q}(k+n-1) \end{bmatrix}_{(n \times M_u) \times 1},$$

$$\mathbf{E}_{(n \times M_y)} = [\mathbf{I}_{M_y} \ \dots \ \mathbf{I}_{M_y}]^T$$

A control input criterion function is given as:

$$J = [\mathbf{Y}_n^*(k+1) - \mathbf{Y}_n(k+1)]^T [\mathbf{Y}_n^*(k+1) - \mathbf{Y}_n(k+1)] + \Delta \mathbf{Q}^T(k) \bar{\lambda} \Delta \mathbf{Q}(k) \quad (26)$$

where  $\bar{\lambda}$  is the weighted diagonal matrix with  $\bar{\lambda} = \text{dig}(\lambda_1, \dots, \lambda_{M_u \times n})$  and we suppose all  $\lambda_i$  are equal to  $\lambda$ ;  $[\tilde{\mathbf{Y}}_n^*(k+1)]^T = [(\mathbf{y}^*(k+1))^T, \dots, (\mathbf{y}^*(k+n))^T]$  is the desired system output vector and  $\mathbf{y}^*(k+i) = [y_1^*(k+i), \dots, y_{M_y}^*(k+i)]^T$  is the desired system output at the future time  $k+i$  ( $i=1, 2, \dots, n$ ).

We substitute (25) into (26) and solve  $\partial J / \partial \Delta \mathbf{Q}_n(k) = 0$  to have

$$[\bar{\boldsymbol{\Psi}}^T(k) \bar{\boldsymbol{\Psi}}(k) + \bar{\lambda}] \Delta \mathbf{Q}(k) = \bar{\boldsymbol{\Psi}}^T(k) [\mathbf{Y}_n^*(k+1) - \mathbf{E}_{(n \times M_y)} \mathbf{y}(k)] \quad (27)$$

Then we retain the current inputs as the local optimal solution:

$$\mathbf{q}(k) = \mathbf{q}(k-1) + \mathbf{g}^T \Delta \mathbf{Q}(k) \quad (28)$$

where  $\mathbf{g}^T = [\mathbf{I}, \mathbf{0}, \dots, \mathbf{0}]$ .

##### B. Stability Analysis of MFAC

This section provides the performance analysis of MFAPC. From (3), (27) and (28), we have the following closed-loop system equation:

$$[\Delta \mathbf{I}_{M_y} + z^{-1} \mathbf{J}(k) \mathbf{g}^T [\bar{\boldsymbol{\Psi}}^T(k) \bar{\boldsymbol{\Psi}}(k) + \bar{\lambda}]^{-1} \bar{\boldsymbol{\Psi}}^T(k) \mathbf{E}_{(n \times M_y)}] \mathbf{y}(k) \quad (29)$$

$$= \mathbf{J}(k) \mathbf{g}^T [\bar{\boldsymbol{\Psi}}^T(k) \bar{\boldsymbol{\Psi}}(k) + \bar{\lambda}]^{-1} \bar{\boldsymbol{\Psi}}^T(k) \mathbf{Y}_n^*(k+1)$$

We assume  $\text{rank}[\mathbf{J}(z^{-1})] = M_y$  ( $M_u \geq M_y$ ). By tuning  $\bar{\lambda}$ , we can obtain inequality as follow:

$$\mathbf{T} = \Delta \mathbf{I}_{M_y} + z^{-1} \mathbf{J}(k) \mathbf{g}^T [\bar{\boldsymbol{\Psi}}^T(k) \bar{\boldsymbol{\Psi}}(k) + \bar{\lambda}]^{-1} \bar{\boldsymbol{\Psi}}^T(k) \mathbf{E}_{(n \times M_y)} \neq \mathbf{0}, |z| > 1 \quad (30)$$

, which determines the poles of the system. Then (30) guarantees the stability of the system according to [15]-[16].

When the desired trajectory is  $\mathbf{Y}_n^*(k+1)$  with its component  $\mathbf{y}^*(k+i) = (k+i-1)^p \cdot [1, \dots, 1]_{1 \times M_y}^T$  ( $i=1, \dots, n$ ), the static error will be

$$\lim_{k \rightarrow \infty} \mathbf{e}(k)$$

$$\begin{aligned} &= \lim_{z \rightarrow 1} \frac{z-1}{z} (\mathbf{I}_{M_y} - \mathbf{T}^{-1} \mathbf{J}(k) \mathbf{g}^T [\bar{\boldsymbol{\Psi}}^T(k) \bar{\boldsymbol{\Psi}}(k) + \bar{\lambda}]^{-1} \bar{\boldsymbol{\Psi}}^T(k) \mathbf{P}) \frac{C(z)}{(z-1)^{p+1}} \\ &= \lim_{z \rightarrow 1} \mathbf{T}^{-1} (\Delta \mathbf{I}_{M_y} - \mathbf{J}(k) \mathbf{g}^T [\bar{\boldsymbol{\Psi}}^T(k) \bar{\boldsymbol{\Psi}}(k) + \bar{\lambda}]^{-1} \bar{\boldsymbol{\Psi}}^T(k) \\ &\quad \cdot (\mathbf{P} - z^{-1} \mathbf{E}_{(n \times M_y)})) \frac{C(z)}{(z-1)^p} \end{aligned} \quad (31)$$

where

$$\begin{aligned} z(\mathbf{Y}_n^*(k+1)) &= \text{diag}(\mathbf{I}_{My}, \dots, z^n \mathbf{I}_{My}) \mathbf{E}_{(n, My)} z(\mathbf{y}^*(k+1)) \\ &= \mathbf{P} \frac{C(z)}{(z-1)^{p+1}} z(\mathbf{y}^*(k+1)) \end{aligned} \quad (32)$$

and  $\mathbf{P}^T = [\mathbf{I}_{My}, z\mathbf{I}_{My}, \dots, z^{n-1}\mathbf{I}_{My}]$ . Further, under the condition that the robot does not pass through the singular points, through choosing  $\bar{\lambda} = \mathbf{0}$  the steady-state error will be eliminated since

$$\begin{aligned} \lim_{k \rightarrow \infty} \mathbf{e}(k) &= \lim_{z \rightarrow 1} \mathbf{T}^{-1} (\mathbf{I}_{My} - \mathbf{J}(k) \mathbf{g}^T [\bar{\Psi}(k)]^{-R} \mathbf{E}_{(n, My)}) \frac{C(z)}{(z-1)^{p+1}} \\ &= \lim_{z \rightarrow 1} \mathbf{T}^{-1} (\mathbf{I}_{My} - \mathbf{J}(k) \mathbf{g}^T [\bar{\Psi}(k)]^{-R} \mathbf{E}_{(n, My)}) \frac{C(z)}{(z-1)^{p+1}} \\ &= \lim_{z \rightarrow 1} \mathbf{T}^{-1} (\mathbf{I}_{My} - \mathbf{J}(k) [\mathbf{J}(k)]^{-R} \mathbf{0}, \dots, \mathbf{0}) \mathbf{E}_{(n, My)} \frac{C(z)}{(z-1)^{p+1}} = \mathbf{0} \end{aligned} \quad (33)$$

where

$$[\bar{\Psi}(k)]^{-R} = \begin{bmatrix} [\mathbf{J}(k)]^{-R} & \mathbf{0} & \dots & \mathbf{0} \\ -[\mathbf{J}(k+1)]^{-R} & [\mathbf{J}(k+1)]^{-R} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & -[\mathbf{J}(k+2)]^{-R} & [\mathbf{J}(k+2)]^{-R} & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ \mathbf{0} & \dots & \mathbf{0} & -[\mathbf{J}(k+n-1)]^{-R} & [\mathbf{J}(k+n-1)]^{-R} \end{bmatrix}$$

and  $[\bullet]^{-R}$  represents the right inverse matrix of  $\bullet$ .

### C. The application of MFAPC in robot inverse kinematics

The application of MFAPC is similar to that of MFAC in Section III.

The finite  $n$ -step forward prediction model at the  $i$ -th iteration is given as

$$\mathbf{y}(k+j)_{(i)} = \mathbf{F}(\mathbf{q}(k+j)_{(i-1)}), \quad j = 0, \dots, n-1 \quad (34)$$

where  $i = 1, \dots, N$ . According to [14], we can easily obtain the Jacobian matrix

$$\mathbf{J}(k+j)_{(i)} = \mathbf{f}(\mathbf{q}(k+j)_{(i-1)}), \quad j = 0, \dots, n-1 \quad (35)$$

Then we let

$$\bar{\Psi}(k)_{(i)} = \begin{bmatrix} \mathbf{J}(k)_{(i)} & & & \\ \mathbf{J}(k)_{(i)} & \mathbf{J}(k+1)_{(i)} & & \\ \vdots & \ddots & \ddots & \\ \mathbf{J}(k)_{(i)} & \dots & \mathbf{J}(k+n-2)_{(i)} & \mathbf{J}(k+n-1)_{(i)} \end{bmatrix}$$

Based on (27), we can obtain  $\Delta Q(k)_{(i)}$  through

$$\begin{aligned} &[\bar{\Psi}^T(k)_{(i)} \bar{\Psi}(k)_{(i)} + \bar{\lambda}(k)_{(i)}] \Delta Q(k)_{(i)} \\ &= \bar{\Psi}^T(k)_{(i)} [\mathbf{Y}_n^*(k+1) - \mathbf{E}_{(n, My)} \mathbf{y}(k)_{(i)}] \end{aligned} \quad (36)$$

And we have tuned  $\bar{\lambda}(k)_{(i+1)}$  in accordance with (37) in the experiment.

$$\bar{\lambda}(k)_{(i)} = \begin{cases} \mathbf{0} & \text{if } 1 < c(\mathbf{J}(k)_{(i)}) < c_1 \\ \bar{\lambda} & \text{if } c_1 \leq c(\mathbf{J}(k)_{(i)}) < c_2 \\ \vdots & \\ \bar{\lambda}_n & \text{if } c(\mathbf{J}(k)_{(i)}) \geq c_n \end{cases} \quad (37)$$

where

$$c(\mathbf{J}(k)_{(i)}) = \max \{ \text{cond}(\mathbf{J}(k)_{(i)}), \dots, \text{cond}(\mathbf{J}(k+n-1)_{(i)}) \}, \quad c_i, \quad (i=1, \dots, n)$$

are the constants. If we choose  $\bar{\lambda}(k)_{(i)} = \mathbf{0}$  to reduce the amount of calculation, we will have

$$[\bar{\Psi}^T(k)_{(i)} \bar{\Psi}(k)_{(i)} + \bar{\lambda}(k)_{(i)}]^{-1} \bar{\Psi}^T(k)_{(i)} = [\bar{\Psi}(k)_{(i)}]^{-R} = \begin{bmatrix} [\mathbf{J}(k)_{(i)}]^{-R} & \mathbf{0} & \dots \\ -[\mathbf{J}(k+1)_{(i)}]^{-R} & [\mathbf{J}(k+1)_{(i)}]^{-R} & \mathbf{0} \\ \mathbf{0} & -[\mathbf{J}(k+2)_{(i)}]^{-R} & [\mathbf{J}(k+2)_{(i)}]^{-R} \\ \vdots & \vdots & \ddots \\ \mathbf{0} & \dots & \mathbf{0} & -[\mathbf{J}(k+n-1)_{(i)}]^{-R} & [\mathbf{J}(k+n-1)_{(i)}]^{-R} \end{bmatrix} \quad (38)$$

Then we have

$$\mathbf{q}(k)_{(i)} = \mathbf{q}(k)_{(i-1)} + \mathbf{g}^T \Delta Q(k)_{(i)} \quad (39)$$

The iteration described by (34)-(39) yields our concerned  $\mathbf{Y}_n(k+1)_{(i)} \triangleq [\mathbf{y}^T(k)_{(i)}, \dots, \mathbf{y}^T(k+n-1)_{(i)}]^T$  and  $\mathbf{q}(k)_{(i)}$ ,  $i=1, \dots, N$  ( $N \leq N_{up}$ ). The above iteration will continue until  $N = N_{up}$  or

$$\|\mathbf{Y}_n^*(k+1) - \mathbf{Y}_n(k+1)\|_2 \leq \delta \quad (40)$$

At last, we will send the final iterative inverse kinematic solution (41) to the robot actuator.

$$\mathbf{q}(k) = \mathbf{q}(k-1) + \sum_{i=1}^N \Delta \mathbf{q}(k)_{(i)} \quad (41)$$

*Remark 5:* Though (33) demonstrates that the tracking error converges to zero on the condition that  $\bar{\lambda} = \mathbf{0}$ , the motivation and effect of MFAPC is to make a balance between the minimization of  $\|\mathbf{Y}_n^*(k+1) - \mathbf{Y}_n(k+1)\|_2$  and the system robustness. As to the difference between MFAPC and MFAC, MFAC is to make a balance between the minimization of  $\|\mathbf{y}^*(k+1) - \mathbf{y}(k+1)\|_2$  and the system robustness. If we let  $\mathbf{Y}_n^*(k+1) = \mathbf{E} \mathbf{y}^*(k+1)$ , the MFAPC will have the same control effect as the MFAC. On the other hand, through  $n=1$  the MFAPC degenerates into the MFAC. Therefore, the MFAPC incorporates the MFAC method in [1] and [8].

## V. SIMULATION

**Example 1:** In this example, the MFAPC is used as a controller (27), (28) for the robotic kinematic control problem. By this way, we can not only verify its control performance but also exhibit its effectiveness on acquiring the solution to inverse kinematic.

We consider a robot with three links  $l_1=5$ ,  $l_2=l_3=7$  as shown in Fig. 1.

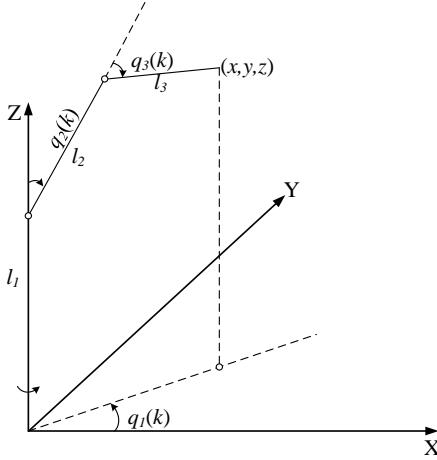


Fig. 1 Robot model

The outputs of the system are the position  $(x, y, z)$  of the robot manipulator in task space. The inputs of the system are the angles  $q_1(t), q_2(t), q_3(t)$  of the robot in joint space. The system model is

$$\begin{aligned} x(k+1) &= (l_2 \sin(q_2(k)) + l_3 \sin(q_2(k) + q_3(k)) \cos(q_1(k))) \\ y(k+1) &= (l_2 \sin(q_2(k)) + l_3 \sin(q_2(k) + q_3(k)) \sin(q_1(k))) \\ z(k+1) &= l_1 + l_2 \cos(q_2(k)) + l_3 \cos(q_2(k) + q_3(k)) \end{aligned} \quad (42)$$

We take partial derivation of equation (42) to have the equivalent-dynamic-linearization model:

$$\Delta \mathbf{y}(k+1) = \begin{bmatrix} \Delta x(k+1) \\ \Delta y(k+1) \\ \Delta z(k+1) \end{bmatrix} = \mathbf{J}(k) \begin{bmatrix} \Delta q_1(k) \\ \Delta q_2(k) \\ \Delta q_3(k) \end{bmatrix} \quad (43)$$

where  $\mathbf{J}(k)$  represents the (pseudo) Jacobian matrix. The desired output trajectory is considered as a helical curve:

$$\begin{aligned} x^*(k) &= 4 + 3 \sin(\pi k / 50) \\ y^*(k) &= 3 \cos(\pi k / 50) \quad 1 \leq k \leq 800 \\ z^*(k) &= 5 + k / 200 \end{aligned}$$

The initial values are  $[x(1), y(1), z(1)] = 0$ ,

$q_1(1) = q_2(1) = q_3(1) = 0$ , nevertheless these initial settings do not suffice the actual forward kinematics of the robot system.

The initial controller parameter is  $\lambda = 2$ ;

In order to make a balance between the convergence of the tracking error and the system stability, the MFAPC with prediction step  $n=5$  is applied as follows:

$$\begin{aligned} \Delta \mathbf{u}(k) &= \mathbf{g}^T [\bar{\Psi}^T(k) \bar{\Psi}(k) + \lambda \mathbf{I}]^{-1} \bar{\Psi}^T(k) \\ &\cdot [\mathbf{Y}^*(k+1) - \mathbf{E}_{(N \times M_y)} \mathbf{y}(k)] \end{aligned} \quad (44)$$

$$\text{If } \left\| \mathbf{Y}_n^*(k+1) - \mathbf{Y}_n(k+1) \right\|_2 > 10,$$

$$\lambda(k) = 1.1 \lambda(k-1)$$

$$\text{else } \lambda(k) = \lambda(k-1) / 1.02$$

where  $\mathbf{y}^*(k+1) = [x^*(k+1), y^*(k+1), z^*(k+1)]^T$  is the desired trajectory, and the current position is  $\mathbf{y}(k) = [x(k), y(k), z(k)]^T$ . We make a local linear

$$\text{approximation } \bar{\Psi}(k) = \begin{bmatrix} \mathbf{J}(k) & & & \\ \mathbf{J}(k) & \mathbf{J}(k) & & \\ \vdots & \ddots & \ddots & \\ \mathbf{J}(k) & \cdots & \mathbf{J}(k) & \mathbf{J}(k) \end{bmatrix}_{(n \times M_y) \times (n \times M_u)}$$

this simulation.

The outputs of the system controlled by MFAC are shown in Fig. 2. The control inputs and the value of controller parameter  $\lambda$  are shown in Fig. 3. Fig. 4 shows the elements in  $\mathbf{J}(k)$ .

Since the initial value of inputs and outputs of the system violates the kinematics of robot, the beginning tracking performance is not well. Simultaneously, the  $\lambda$  increases to enhance the robustness of the system. After the system is stable with the tracking error of the system lower than 0.1 at time of 42, the  $\lambda$  decreases to guarantee the convergence of the tracking error.

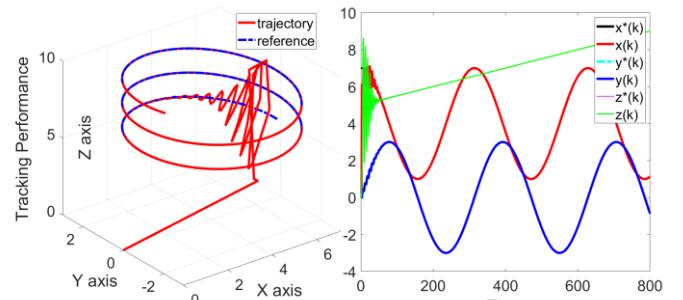
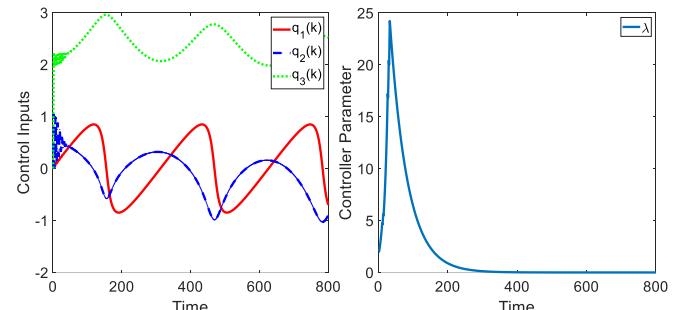
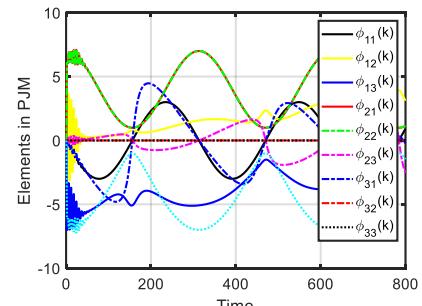


Fig. 2 Tracking performance

Fig. 3 Control inputs and controller parameter  $\lambda$ Fig. 4 Elements in  $\mathbf{J}(k)$ 

Example 2: A six dimensional industrial robot is shown in Fig. 5.



Fig. 5 Industrial robot

Herein, we will design the MFAPC controller for the industrial robot manipulator. The Denavit-Hartenberg parameters of the manipulator are listed in [b] to obtain the frame in Cartesian-space.

#### Part A: Iterative MFAPC controller design

The path generator is designed for the linear spline with parabolic blends path according to [b]. The desired path (trajectory) point in Cartesian-space at path-planning time  $k$  is  $\mathbf{y}^*(k) = [x^*(k), y^*(k), z^*(k), \alpha^*(k), \beta^*(k), \gamma^*(k)]^T$  (45)

Then the description of corresponding desired frame at path-planning time  $k+j$  ( $j=1, \dots, n$ ) is written as

$$\mathbf{T}^*(k+j) = \begin{pmatrix} \mathbf{A}^*(k+j) & \mathbf{p}^*(k+j) \\ \mathbf{0}_{3 \times 1} & 1 \end{pmatrix} \quad (46)$$

where

$$\mathbf{A}^*(k+j) = \begin{bmatrix} t_{11}^*(k+j) & t_{12}^*(k+j) & t_{13}^*(k+j) \\ t_{21}^*(k+j) & t_{22}^*(k+j) & t_{23}^*(k+j) \\ t_{31}^*(k+j) & t_{32}^*(k+j) & t_{33}^*(k+j) \end{bmatrix} \quad (47)$$

denotes the desired rotation matrix and

$$\mathbf{p}^*(k+j) = [x^*(k+j), y^*(k+j), z^*(k+j)]^T \quad (48)$$

represents the desired position vector at path-planning time  $k+j$ . According to [14] and [b], we can easily calculate  $t_{11}^*(k+j), \dots, t_{33}^*(k+j)$  in accordance with (45) at time  $k+j$ .

Assume that the calculated robot frame at the  $i$ -th iteration for the inverse kinematics of the desired path vector  $\mathbf{Y}_n^*(k+1)$  is described by

$$\mathbf{T}(k)_{(i)} = \begin{bmatrix} t(k)_{11(i)} & t(k)_{12(i)} & t(k)_{13(i)} & x(k)_{(i)} \\ t(k)_{21(i)} & t(k)_{22(i)} & t(k)_{23(i)} & y(k)_{(i)} \\ t(k)_{31(i)} & t(k)_{32(i)} & t(k)_{33(i)} & z(k)_{(i)} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (49)$$

The corresponding orientation matrix is written as

$$\mathbf{A}(k)_{(i)} = \begin{bmatrix} t(k)_{11(i)} & t(k)_{12(i)} & t(k)_{13(i)} \\ t(k)_{21(i)} & t(k)_{22(i)} & t(k)_{23(i)} \\ t(k)_{31(i)} & t(k)_{32(i)} & t(k)_{33(i)} \end{bmatrix} \quad (50)$$

The desired orientation matrix at path-planning time  $k+j$  relative to the calculated orientation matrix of manipulator at the  $i$ -th iteration will be

$$\begin{aligned} \mathbf{D}(k+j)_{(i)} &= \begin{bmatrix} d(k+j)_{11(i)} & d(k+j)_{12(i)} & d(k+j)_{13(i)} \\ d(k+j)_{21(i)} & d(k+j)_{22(i)} & d(k+j)_{23(i)} \\ d(k+j)_{31(i)} & d(k+j)_{32(i)} & d(k+j)_{33(i)} \end{bmatrix} \\ &= \mathbf{A}^*(k+j) (\mathbf{A}(k)_{(i)})^{-1} \end{aligned} \quad (51)$$

Here, the orientation matrix is converted into equivalent angle-axis representation to calculate the difference among the Euler angles  $\alpha^*(k+j) - \alpha(k)_{(i)}$ ,  $\beta^*(k+j) - \beta(k)_{(i)}$  and  $\gamma^*(k+j) - \gamma(k)_{(i)}$ . The equivalent angle is

$$\theta(k+j)_{(i)} = \arccos \left( \frac{d(k+j)_{11(i)} + d(k+j)_{12(i)} + d(k+j)_{33(i)} - 1}{2} \right) \quad (52)$$

The equivalent axis of a finite rotation is

$$\hat{\mathbf{K}}(k+j)_{(i)} = \begin{pmatrix} d(k+j)_{32(i)} - d(k+j)_{23(i)} \\ d(k+j)_{13(i)} - d(k+j)_{31(i)} \\ d(k+j)_{21(i)} - d(k+j)_{12(i)} \end{pmatrix} \Bigg/ 2 \sin \theta(k+j)_{(i)} \quad (53)$$

Then the relative rotational Euler angles vector is calculated by

$$\begin{bmatrix} \alpha^*(k+j) - \alpha(k)_{(i)} \\ \beta^*(k+j) - \beta(k)_{(i)} \\ \gamma^*(k+j) - \gamma(k)_{(i)} \end{bmatrix} = \hat{\mathbf{K}}(k+j)_{(i)} \theta(k+j)_{(i)} \quad (54)$$

The control law is

$$\begin{aligned} \Delta \mathbf{Q}(k)_{(i)} &= [\Delta \mathbf{q}^T(k)_{(i)}, \dots, \Delta \mathbf{q}^T(k+n-1)_{(i)}]^T \\ &= [\bar{\Psi}^T(k)_{(i)} \bar{\Psi}(k)_{(i)} + \bar{\lambda}(k)_{(i)}]^{-1} \bar{\Psi}^T(k)_{(i)} [\mathbf{Y}_n^*(k+1) - \mathbf{E}_{(n, My)} \mathbf{y}(k)_{(i)}] \\ &= [\bar{\Psi}^T(k)_{(i)} \bar{\Psi}(k)_{(i)} + \bar{\lambda}(k)_{(i)}]^{-1} \bar{\Psi}^T(k)_{(i)} [x^*(k+1) - x(k)_{(i)}, \\ &\quad y^*(k+1) - y(k)_{(i)}, z^*(k+1) - z(k)_{(i)}, \alpha^*(k+1) - \alpha(k)_{(i)}, \\ &\quad \beta^*(k+1) - \beta(k)_{(i)}, \gamma^*(k+1) - \gamma(k)_{(i)}, \dots, x^*(k+n) - x(k)_{(i)}, \\ &\quad y^*(k+n) - y(k)_{(i)}, z^*(k+n) - z(k)_{(i)}, \alpha^*(k+n) - \alpha(k)_{(i)}, \\ &\quad \beta^*(k+n) - \beta(k)_{(i)}, \gamma^*(k+n) - \gamma(k)_{(i)}]^T \end{aligned} \quad (55)$$

where

$$\begin{bmatrix} \mathbf{q}(k)_{(i)} \\ \vdots \\ \mathbf{q}(k+n-1)_{(i)} \end{bmatrix} = \begin{bmatrix} \mathbf{q}(k)_{(i-1)} \\ \vdots \\ \mathbf{q}(k)_{(i-1)} \end{bmatrix} + \begin{bmatrix} \mathbf{I} & & \\ \vdots & \ddots & \\ \mathbf{I} & \cdots & \mathbf{I} \end{bmatrix} \Delta \mathbf{Q}(k)_{(i)} \quad (56)$$

and we can easily obtain the Jacobian matrix  $\mathbf{J}(k+j)_{(i+1)}$ , ( $j=1, \dots, n-1$ ) in accordance with (35) and (56) to constitute  $\bar{\Psi}^T(k)_{(i+1)}$ .

Then we choose the local optimal solution for the controller

$$\mathbf{q}(k)_{(i)} = \mathbf{q}(k)_{(i-1)} + \mathbf{g}^T \Delta \mathbf{Q}(k)_{(i)} \quad (57)$$

According to the robotic kinematics (34), we can easily transform (56) to

$$\mathbf{Y}_n(k+1)_{(i+1)} \triangleq [\mathbf{y}^T(k)_{(i+1)}, \dots, \mathbf{y}^T(k+n-1)_{(i+1)}]^T \quad (58)$$

and  $\mathbf{y}(k)_{(i+1)}$  can be converted into (49) for the  $i+1$ -th iteration.

Then (49)-(58) constitutes a closed-loop to find a solution for the minimum of  $\|\mathbf{Y}_n^*(k+1) - \mathbf{Y}_n(k+1)_{(N)}\|_2$ .

#### Part B: Test MFAPC

We have tried to design MFAPC with prediction step  $n=3$ . Since the calculation time of  $[\bar{\Psi}^T(k)_{(i)} \bar{\Psi}(k)_{(i)} + \bar{\lambda}(k)_{(i)}]_{18 \times 18}^{-1}$  exceeds 1ms which is the implementation period of the robot,

the industrial computer does not send any signals to the robot. To reduce the amount of calculation of inverse operation, we design the iterative MFAPC with  $n=2$  and  $\bar{\lambda} = \mathbf{0}$  as follow

$$\begin{aligned}\Delta Q(k)_{(i)} &= [\Delta q(k)_{(i)}, \Delta q(k)_{(i+1)}] \\ &= \bar{\Psi}^{-1}(k)_{(i)} [Y_n^*(k+1) - E_{(n, My)} y(k)_{(i)}]\end{aligned}\quad (59)$$

where  $\bar{\Psi}^{-1}(k)_{(i)} = \begin{bmatrix} J^{-1}(k)_{(i)} & \mathbf{0} \\ -J^{-1}(k+1)_{(i)} & J^{-1}(k+1)_{(i)} \end{bmatrix}$  for further

simplifying the calculation of the inverse operation. And the maximum number of iterations is set to 10.

The beginning joint angle vector of manipulator is  $A[-\pi/4, 0, 0, 0, -\pi/2, 0]^T$  and the goal frame is set to  $B$  in Cartesian space whose corresponding joint angle vector is  $[\pi/4, 0, 0, 0, -\pi/2, 0]^T$ .

Fig. 6 shows the tracking performance of the manipulator and Fig. 7 shows the corresponding tracking error  $y^*(k+1) - y(k)$ . Fig. 8 shows the measured joint angles and the iteration count.

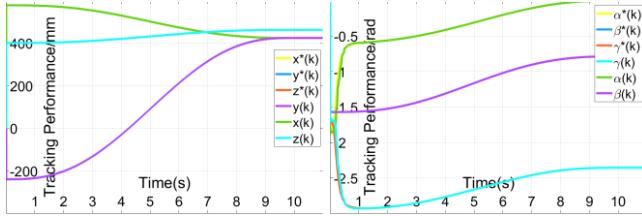


Fig. 6 Tracking performance

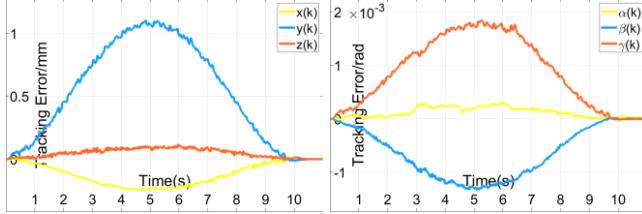


Fig. 7 Tracking error

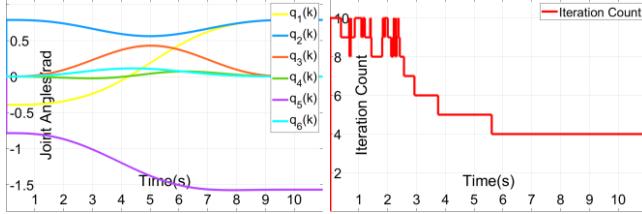


Fig. 8 Measured joint angles and the iteration count

## VI. CONCLUSION

In this brief, we have reanalyzed current robust numerical solution to the inverse kinematics based on Levenberg-Marquardt (LM) method through conventional control theory and reviewed its application in robotics. Compared to the current numerical analysis method, the relationship between the convergence performance of computational error and the damping factor is analyzed more clearly through analyzing the system control performance in this brief. Furthermore, by this effort, we also showed that the current works about MFAC are not studied in a reasonable way. Then we redesign the MFAC into MFAPC to utilize more desired trajectories in the future

time. It not only shows an excellent control performance but also efficiently acquires the solution to the inverse kinematic of robotics.

## REFERENCES

- [1] Sugihara T. Solvability-Unconcerned Inverse Kinematics by the Levenberg-Marquardt Method[J]. IEEE Transactions on Robotics, 2011, 27(5):984-991.
- [2] D. E. Orin and W. W. Schrader, "Efficient computation of the Jacobian for robot manipulators," I International Journal of Robotics Research[J]. 1984, 3(4): 66-75.
- [3] Wang L T , Chen C C. A combined optimization method for solving the inverse kinematics problems of mechanical manipulators[J]. Robotics & Automation IEEE Transactions on, 1991, 7(4):489-499.
- [4] Zhao J and Badler N I. "Inverse kinematics positioning using nonlinear programming for highly articulated figures," ACM Transactions on Graphics[J]. 1994, 13(4): 313-336.
- [5] Anderson K , Angeles J . Kinematic Inversion of Robotic Manipulators in the Presence of Redundancies[J]. International Journal of Robotics Research, 1989, 8(6):80-97.
- [6] Levenberg K. "A method for the solution of certain non-linear problems in least squares," Quarterly of applied mathematics[J]. 1944, 2: 4164-4168.
- [7] D. W. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," Journal of the society for Industrial and Applied[J]. 1963, 11(2), 431-441.
- [8] Hou Z S, Jin S T, Model Free Adaptive Control: Theory and Applications, CRC Press, Taylor and Francis Group, 2013
- [9] Hou Z S, Jin S T. Data-Driven Model-Free Adaptive Control for a Class of MIMO Nonlinear Discrete-Time Systems[J]. IEEE Transactions on Neural Networks, 2011, 22(12): 2173-2188.
- [10] Guo Y, Hou Z S, Liu S D, Jin S T. Data-Driven Model-Free Adaptive Predictive Control for a Class of MIMO Nonlinear Discrete-Time Systems With Stability Analysis[J]. IEEE Access, 2019: 102852-102866.
- [11] Hou Z S, Jin S T, "A novel data-driven control approach for a class of discrete-time nonlinear systems," IEEE Transaction on Control Systems Technology, 2011, 19(6):1549-1558.
- [12] Hou Z S, Xiong S S. On Model Free Adaptive Control and its Stability Analysis[J]. IEEE Transactions on Automatic Control, 2019, 64(11): 4555-4569.
- [13] Lynch K M, Park F C. Modern Robotics [M]. Cambridge University Press, 2017.
- [14] Craig J J. Introduction to Robotics: Mechanics and Control[M], Addison-Wesley, Boston, 2005.
- [15] Chai T Y. Multivariable adaptive decoupling control and its application [M]. Beijing: Science Press, 2001.
- [16] Chai T Y, Yue H. Adaptive Control [M]. Beijing: Tsinghua University Press, 2015.
- [17] Goodwin G C, Sin K S. Adaptive Filtering Prediction and Control [M]. Dover Publications, Inc, 2009.
- [a] Performance Analysis of Model-Free Adaptive Control
- [b] Discussion on a Class of Model-Free Adaptive Control for Multivariable Systems
- [c] A New Model-Free Method for MIMO Systems and Discussion on Model-Free or Model-Based