10 Neural Network Sliding Mode Control

Jinkun Liu
Beijing University of Aeronautics and Astronautics
P.R.China
E-mail: ljk@buaa.edu.cn

Xinhua Wang
National University of Singapore
Singapore
E-mail: wangxinhua04@gmail.com

Abstract This chapter introduces two kinds of neural network sliding mode controllers, including a sliding mode controller design based on RBF neural network approximation and an adaptive RBF network sliding mode control for manipulator.

Keywords sliding mode control, RBF neural network, manipulator

Past research of the universal approximation theorem[1, 2] show that any nonlinear function over a compact set with arbitrary accuracy can be approximated by the RBF neural network. There have been significant research efforts on the RBF neural control for nonlinear systems[3]. In section 10.1 an adaptive neural sliding mode control algorithm is proposed for a class of continuous time unknown nonlinear systems. This is in contrast to the existing sliding mode control design where the presence of hitting control may introduce problems to the controlled systems. These unknown nonlinearities are approximated by the RBF neural network whose weight value parameters are adjusted on-line according to some adaptive laws. The purpose of controlling the output of the nonlinear system is to track a given trajectory. Based on the RBF model, the Lyapunov synthesis approach is used to develop an adaptive control algorithm. The chattering action is attenuated and a robust performance can be ensured. The stability analysis for the proposed control algorithm is provided. In section 10.2 the RBF network is used to approximate the unknown part of the manipulator dynamic equation. This does not require modeling. Also, the approximation error and disturbance can be compensated by the sliding mode control.

© Tsinghua University Press, Beijing and Springer-Verlag Berlin Heidelberg 2011

10.1 Sliding Mode Control Based on RBF Neural Network Approximation

10.1.1 Problem Statement

Consider a second-order nonlinear system as follow:

$$\ddot{\theta} = f(\theta,\dot{\theta}) + g(\theta,\dot{\theta})u + d(t) \tag{10.1}$$

where $f(\cdot)$ and $g(\cdot)$ are all nonlinear functions, $u \in \mathbf{R}$ and $y \in \mathbf{R}$ are the input control and output respectively, $d(t)$ is the outer disturbance and $|d(t)| \leqslant D$.

Let the desired output be θ_d and denote

$$e = \theta_\mathrm{d} - \theta$$

Design sliding mode function as

$$s = \dot{e} + ce \tag{10.2}$$

where $c > 0$, then

$$\dot{s} = \ddot{e} + c\dot{e} = \ddot{\theta}_\mathrm{d} - \ddot{\theta} + c\dot{e} = \ddot{\theta}_\mathrm{d} - f - gu - d(t) + c\dot{e} \tag{10.3}$$

If f and g are known, we can design control law as

$$u = \frac{1}{g}(-f + \ddot{\theta}_\mathrm{d} + c\dot{e} + \eta\,\mathrm{sgn}(s)) \tag{10.4}$$

Then Eq. (10.3) becomes

$$\dot{s} = \ddot{e} + c\dot{e} = \ddot{\theta}_\mathrm{d} - \ddot{\theta} + c\dot{e} = \ddot{\theta}_\mathrm{d} - f - gu - d(t) + c\dot{e} = -\eta\,\mathrm{sgn}(s) - d(t)$$

Therefore, if $\eta \geqslant D$, we have

$$s\dot{s} = -\eta|s| - s \cdot d(t) \leqslant 0$$

If $f(x)$ is unknown, we should estimate $f(x)$ by some algorithms. In the following, we will simply recall RBF neural network approximate uncertain item $f(x)$.

10.1.2 Controller Design Based on a Radial Basis Function Neural Network

RBF networks are adaptively used to approximate the uncertain f. The algorithm of a radial basis function (RBF) networks is[2]:

$$h_j = g(\|\boldsymbol{x} - c_{ij}\|^2 / b_j^2)$$

$$f = \boldsymbol{W}^\mathrm{T}\boldsymbol{h}(\boldsymbol{x}) + \varepsilon$$

where $\boldsymbol{x}$ is the input state of the network, i is the input number of the network,

j is the number of hidden layer nodes in the network, $\boldsymbol{h}=[h_1 \quad h_2 \quad \cdots \quad h_n]^{\mathrm{T}}$ is the output of Gaussian function, $\boldsymbol{W}$ is the neural network weights, ε is approximation error of neural network, and $\varepsilon \leqslant \varepsilon_{\mathrm{N}}$.

RBF network approximation f is used. The network input is selected as $\boldsymbol{x}=[e \quad \dot{e}]^{\mathrm{T}}$, and the output of RBF neural network is

$$\hat{f}(\boldsymbol{x})=\hat{\boldsymbol{W}}^{\mathrm{T}}\boldsymbol{h}(\boldsymbol{x}) \tag{10.5}$$

where $\boldsymbol{h}(\boldsymbol{x})$ is the Gaussian function of neural network.

We know that Gaussian function and the neural network weights are difficult to select.

The control input Eq. (10.4) is written as

$$u=\frac{1}{g}(-\hat{f}(\boldsymbol{x})+\ddot{\theta}_{\mathrm{d}}+c\dot{e}+\eta\,\mathrm{sgn}(s)) \tag{10.6}$$

Submitting Eq. (10.6) to Eq. (10.3), we have

$$\begin{aligned}\dot{s}&=\ddot{\theta}_{\mathrm{d}}-f(\boldsymbol{x})-gu-d(t)+c\dot{e}=\ddot{\theta}_{\mathrm{d}}-f(\boldsymbol{x})-(-\hat{f}(\boldsymbol{x})+\ddot{\theta}_{\mathrm{d}}+c\dot{e}+\eta\,\mathrm{sgn}(t))-d(t)+c\dot{e}\\&=-f(\boldsymbol{x})+\hat{f}(\boldsymbol{x})-\eta\,\mathrm{sgn}(s)-d(t)=-\tilde{f}(\boldsymbol{x})-d(t)-\eta\,\mathrm{sgn}(s)\end{aligned} \tag{10.7}$$

where

$$\tilde{f}(\boldsymbol{x})=f(\boldsymbol{x})-\hat{f}(\boldsymbol{x})=\boldsymbol{W}^{\mathrm{T}}\boldsymbol{h}(\boldsymbol{x})+\varepsilon-\hat{\boldsymbol{W}}^{\mathrm{T}}\boldsymbol{h}(\boldsymbol{x})=\tilde{\boldsymbol{W}}^{\mathrm{T}}\boldsymbol{h}(\boldsymbol{x})+\varepsilon \tag{10.8}$$

Define the Lyapunov function as

$$L=\frac{1}{2}s^2+\frac{1}{2}\gamma\tilde{\boldsymbol{W}}^{\mathrm{T}}\tilde{\boldsymbol{W}}$$

where γ is a positive coefficient.

Derivative L, and from Eqs. (10.6) and (10.7), we have

$$\begin{aligned}\dot{L}&=s\dot{s}+\gamma\tilde{\boldsymbol{W}}^{\mathrm{T}}\dot{\tilde{\boldsymbol{W}}}=s(-\tilde{f}(\boldsymbol{x})-d(t)-\eta\,\mathrm{sgn}(s))-\gamma\tilde{\boldsymbol{W}}^{\mathrm{T}}\dot{\hat{\boldsymbol{W}}}\\&=s(-\tilde{\boldsymbol{W}}^{\mathrm{T}}\boldsymbol{h}(\boldsymbol{x})-\varepsilon-d(t)-\eta\,\mathrm{sgn}(s))-\gamma\tilde{\boldsymbol{W}}^{\mathrm{T}}\dot{\hat{\boldsymbol{W}}}\\&=-\tilde{\boldsymbol{W}}^{\mathrm{T}}(s\boldsymbol{h}(\boldsymbol{x})+\gamma\dot{\hat{\boldsymbol{W}}})-s\big(\varepsilon+d(t)+\eta\,\mathrm{sgn}(s)\big)\end{aligned}$$

Let the adaptive rule be

$$\dot{\hat{\boldsymbol{W}}}=-\frac{1}{\gamma}s\boldsymbol{h}(\boldsymbol{x}) \tag{10.9}$$

Then

$$\dot{L}=-s(\varepsilon+d(t)+\eta\,\mathrm{sgn}(s))=-s(\varepsilon+d(t))-\eta\,|\,s\,|$$

We get $\dot{L}\leqslant 0$ approximately as the approximation error ε is sufficiently small in the design $\eta\geqslant\varepsilon_{\mathrm{N}}+D$.

10.1.3 Simulation Example

Consider the following inverted pendulum:

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = \dfrac{g\sin x_1 - mlx_2^2\cos x_1 \sin x_1/(m_c+m)}{l(4/3 - m\cos^2 x_1/(m_c+m))} + \dfrac{\cos x_1/(m_c+m)}{l(4/3 - m\cos^2 x_1/(m_c+m))}u \end{cases}$$

where x_1 and x_2 are the swing angle and swing rate respectively. $g = 9.8\ \mathrm{m/s^2}$, $m_c = 1\ \mathrm{kg}$ is the vehicle mass, m is the mass of the pendulum. l is one half of the pendulum length, and u is the control input.

Choosing $x_1 = \theta$, the desired trajectory is $\theta_d(t) = 0.1\sin t$. The initial state of the plant is $[\pi/60, 0]$. We adapt control law as Eq. (10.6) and adaptive law as Eq. (10.9), choose $c = 15$, $\eta = 0.1$ and adaptive parameter $\gamma = 0.05$.

The structure of RBF is chosen with two input-five hidden-one output, $c_{ij} = 0.20$, $b_j = 0.50$, the initial value of RBF weight value is set as 0.10. The curves of position tracking and uncertainty approximation are shown in Fig. 10.1 – Fig.10.3.

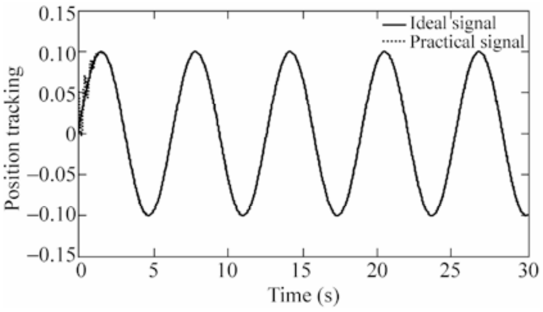


Figure 10.1 Position tracking

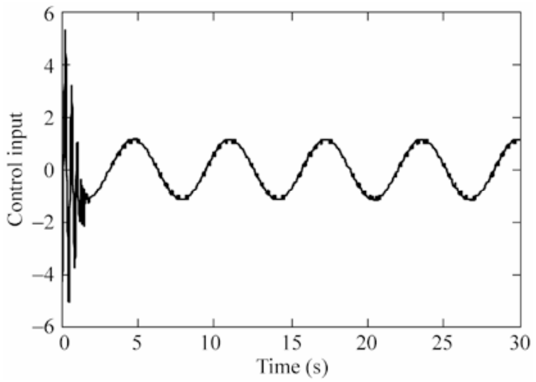


Figure 10.2 Control input

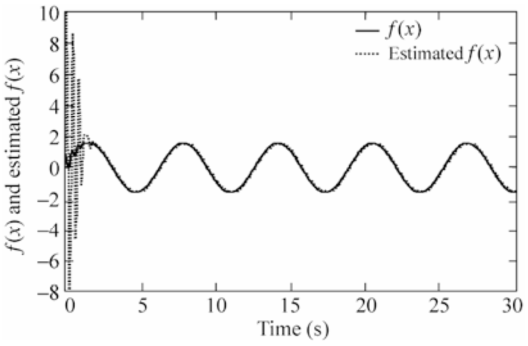
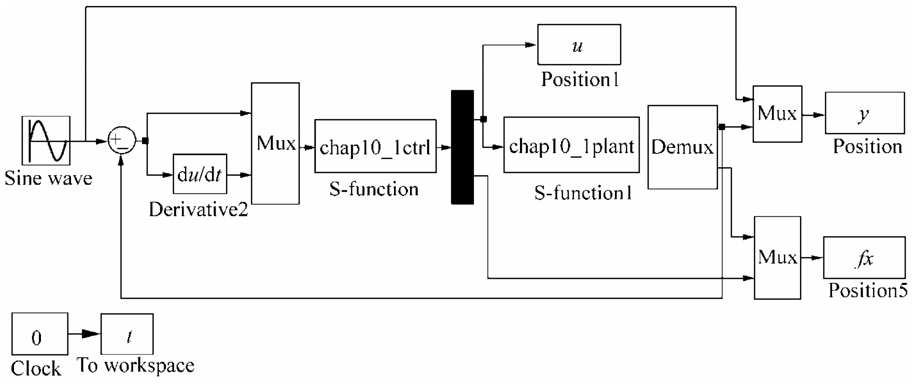


Figure 10.3 $f(\boldsymbol{x})$ and $\hat{f}(\boldsymbol{x})$

Simulation programs:

(1) Main Simulink program: chap10_1sim.mdl



(2) Control law program: chap10_1ctrl.m

```
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)
switch flag,
case 0,
   [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
   sys=mdlDerivatives(t,x,u);
case 3,
   sys=mdlOutputs(t,x,u);
case {2,4,9}
   sys=[];
otherwise
   error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
global c b n
sizes = simsizes;
```

```
sizes.NumContStates  = 5;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 2;
sizes.NumInputs      = 2;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys = simsizes(sizes);
x0  = 0*ones(1,5);
str = [];
ts  = [];
c=0.10*ones(2,5);
b=0.50*ones(5,1);
n=15;
function sys=mdlDerivatives(t,x,u)
global c b n
e=u(1);
de=u(2);
s=n*e+de;

xi=[e;de];
h=zeros(5,1);
for j=1:1:5
   h(j)=exp(-norm(xi-c(:,j))^2/(2*b(j)*b(j)));
end
gama=0.015;
W=[x(1) x(2) x(3) x(4) x(5)]';
for i=1:1:5
   sys(i)=-1/gama*s*h(i);
end
function sys=mdlOutputs(t,x,u)
global c b n
e=u(1);
de=u(2);
thd=0.1*sin(t);
dthd=0.1*cos(t);
ddthd=-0.1*sin(t);
x1=thd-e;

s=n*e+de;
W=[x(1) x(2) x(3) x(4) x(5)]';
xi=[e;de];
h=zeros(5,1);
for j=1:1:5
   h(j)=exp(-norm(xi-c(:,j))^2/(2*b(j)*b(j)));
end
fn=W'*h;

g=9.8;mc=1.0;m=0.1;l=0.5;
S=l*(4/3-m*(cos(x1))^2/(mc+m));
gx=cos(x1)/(mc+m);
gx=gx/S;
```

```
if t<=1.5
   xite=1.0;
else
   xite=0.10;
end
ut=1/gx*(-fn+ddthd+n*de+xite*sign(s));
sys(1)=ut;
sys(2)=fn;
```

(3) Plant program: chap10_1plant.m

```
function [sys,x0,str,ts]=s_function(t,x,u,flag)
switch flag,
case 0,
   [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
   sys=mdlDerivatives(t,x,u);
case 3,
   sys=mdlOutputs(t,x,u);
case {2, 4, 9 }
   sys = [];
otherwise
   error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates  = 2;
sizes.NumDiscStates  = 0;
sizes.NumOutputs    = 2;
sizes.NumInputs     = 1;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[pi/60 0];
str=[];
ts=[];
function sys=mdlDerivatives(t,x,u)
g=9.8;mc=1.0;m=0.1;l=0.5;
S=l*(4/3-m*(cos(x(1)))^2/(mc+m));
fx=g*sin(x(1))-m*l*x(2)^2*cos(x(1))*sin(x(1))/(mc+m);
fx=fx/S;
gx=cos(x(1))/(mc+m);
gx=gx/S;
%%%%%%%%%
dt=0*10*sin(t);
%%%%%%%%%

sys(1)=x(2);
sys(2)=fx+gx*u+dt;
function sys=mdlOutputs(t,x,u)
g=9.8;
mc=1.0;
m=0.1;
```

```
l=0.5;

S=l*(4/3-m*(cos(x(1)))^2/(mc+m));
fx=g*sin(x(1))-m*l*x(2)^2*cos(x(1))*sin(x(1))/(mc+m);
fx=fx/S;

sys(1)=x(1);
sys(2)=fx;
```

(4) Plot program: chap10_1plot.m

```
close all;

figure(1);
plot(t,y(:,1),'k',t,y(:,2),'r:','linewidth',2);
xlabel('time(s)');ylabel('Position tracking');
legend('ideal signal','practical signal');

figure(2);
plot(t,u(:,1),'k','linewidth',2);
xlabel('time(s)');ylabel('Control input');

figure(3);
plot(t,fx(:,1),'k',t,fx(:,2),'r:','linewidth',2);
xlabel('time(s)');ylabel('fx and estiamted fx');
legend('fx','estiamted fx');
```

10.2 RBF Network Adaptive Sliding Mode Control for Manipulator

10.2.1 Problem Statement

Consider the dynamic equation of an n-joint manipulator as follows:

$$\boldsymbol{H}(\boldsymbol{q})\ddot{\boldsymbol{q}}+\boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}}+\boldsymbol{G}(\boldsymbol{q})=\boldsymbol{\tau}-\boldsymbol{F}(\dot{\boldsymbol{q}})-\boldsymbol{\tau}_{\mathrm{d}} \tag{10.10}$$

where $\boldsymbol{H}(\boldsymbol{q})$ is an $n\times n$ positive definite inertial matrix, $\boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})$ is an $n\times n$ inertial matrix, $\boldsymbol{G}(\boldsymbol{q})$ is an $n\times 1$ inertial vector, $\boldsymbol{F}(\dot{\boldsymbol{q}})$ is friction force, $\boldsymbol{\tau}_{\mathrm{d}}$ is the unknown disturbance, and $\boldsymbol{\tau}$ is the control input.

Denote the tracking error as:

$$\boldsymbol{e}(t)=\boldsymbol{q}_{\mathrm{d}}(t)-\boldsymbol{q}(t)$$

Select the sliding variable as:

$$\boldsymbol{s}=\dot{\boldsymbol{e}}+\boldsymbol{\Lambda}\boldsymbol{e} \tag{10.11}$$

where $\boldsymbol{\Lambda}$ is a symmetric positive definite constant matrix and $\boldsymbol{\Lambda}=\boldsymbol{\Lambda}^{\mathrm{T}}>0,$ therefore, we have

$$\dot{\boldsymbol{q}} = -\boldsymbol{s} + \dot{\boldsymbol{q}}_{\mathrm{d}} + \boldsymbol{\Lambda}\boldsymbol{e}$$

$$\begin{aligned}\boldsymbol{H}\dot{\boldsymbol{s}} &= \boldsymbol{H}(\ddot{\boldsymbol{q}}_{\mathrm{d}} - \ddot{\boldsymbol{q}} + \boldsymbol{\Lambda}\dot{\boldsymbol{e}}) = \boldsymbol{H}(\ddot{\boldsymbol{q}}_{\mathrm{d}} + \boldsymbol{\Lambda}\dot{\boldsymbol{e}}) - \boldsymbol{H}\ddot{\boldsymbol{q}} \\ &= \boldsymbol{H}(\ddot{\boldsymbol{q}}_{\mathrm{d}} + \boldsymbol{\Lambda}\dot{\boldsymbol{e}}) + \boldsymbol{C}\dot{\boldsymbol{q}} + \boldsymbol{G} + \boldsymbol{F} + \boldsymbol{\tau}_{\mathrm{d}} - \boldsymbol{\tau} \\ &= \boldsymbol{H}(\ddot{\boldsymbol{q}}_{\mathrm{d}} + \boldsymbol{\Lambda}\dot{\boldsymbol{e}}) - \boldsymbol{C}\boldsymbol{s} + \boldsymbol{C}(\dot{\boldsymbol{q}}_{\mathrm{d}} + \boldsymbol{\Lambda}\boldsymbol{e}) + \boldsymbol{G} + \boldsymbol{F} + \boldsymbol{\tau}_{\mathrm{d}} - \boldsymbol{\tau} \\ &= -\boldsymbol{C}\boldsymbol{s} - \boldsymbol{\tau} + \boldsymbol{f} + \boldsymbol{\tau}_{\mathrm{d}} \end{aligned} \tag{10.12}$$

where $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{H}(\ddot{\boldsymbol{q}}_{\mathrm{d}} + \boldsymbol{\Lambda}\dot{\boldsymbol{e}}) + \boldsymbol{C}(\boldsymbol{q}_{\mathrm{d}} + \boldsymbol{\Lambda}\boldsymbol{e}) + \boldsymbol{G} + \boldsymbol{F}$.

In engineering, $\boldsymbol{f}(\boldsymbol{x})$ is unknown and, therefore, it is required to approximate $\boldsymbol{f}(\boldsymbol{x})$. The RBF network is adopted to approximate $\boldsymbol{f}(\boldsymbol{x})$. The network input is selected based on the expression of $\boldsymbol{f}(\boldsymbol{x})$ [4]:

$$\boldsymbol{x} = [\boldsymbol{e}^{\mathrm{T}} \quad \dot{\boldsymbol{e}}^{\mathrm{T}} \quad \boldsymbol{q}_{\mathrm{d}}^{\mathrm{T}} \quad \dot{\boldsymbol{q}}_{\mathrm{d}}^{\mathrm{T}} \quad \ddot{\boldsymbol{q}}_{\mathrm{d}}^{\mathrm{T}}]$$

The controller is designed as:

$$\boldsymbol{\tau} = \hat{\boldsymbol{f}}(\boldsymbol{x}) + \boldsymbol{K}_{\mathrm{v}}\boldsymbol{s} \tag{10.13}$$

where $\boldsymbol{K}_{\mathrm{v}}$ is a symmetric positive definite constant matrix, $\hat{\boldsymbol{f}}(\boldsymbol{x})$ is the output of RBF network. $\hat{\boldsymbol{f}}(\boldsymbol{x})$ approximates $\boldsymbol{f}(\boldsymbol{x})$.

From Eqs. (10.13) and (10.12), we have

$$\begin{aligned}\boldsymbol{H}\dot{\boldsymbol{s}} &= -\boldsymbol{C}\boldsymbol{s} - \hat{\boldsymbol{f}}(\boldsymbol{x}) - \boldsymbol{K}_{\mathrm{v}}\boldsymbol{s} + \boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{\tau}_{\mathrm{d}} \\ &= -(\boldsymbol{K}_{\mathrm{v}} + \boldsymbol{C})\boldsymbol{s} + \tilde{\boldsymbol{f}}(\boldsymbol{x}) + \boldsymbol{\tau}_{\mathrm{d}} = -(\boldsymbol{K}_{\mathrm{v}} + \boldsymbol{C})\boldsymbol{s} + \varsigma_0 \end{aligned} \tag{10.14}$$

where $\tilde{\boldsymbol{f}}(\boldsymbol{x}) = \boldsymbol{f}(\boldsymbol{x}) - \hat{\boldsymbol{f}}(\boldsymbol{x})$, $\varsigma_0 = \tilde{\boldsymbol{f}}(\boldsymbol{x}) + \boldsymbol{\tau}_{\mathrm{d}}$.

Select the Lyapunov function as:

$$L = \frac{1}{2}\boldsymbol{s}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{s}$$

Therefore,

$$\dot{L} = \boldsymbol{s}^{\mathrm{T}}\boldsymbol{H}\dot{\boldsymbol{s}} + \frac{1}{2}\boldsymbol{s}^{\mathrm{T}}\dot{\boldsymbol{H}}\boldsymbol{s} = -\boldsymbol{s}^{\mathrm{T}}\boldsymbol{K}_{\mathrm{v}}\boldsymbol{s} + \frac{1}{2}\boldsymbol{s}^{\mathrm{T}}(\dot{\boldsymbol{H}} - 2\boldsymbol{C})\boldsymbol{s} + \boldsymbol{s}^{\mathrm{T}}\varsigma_0$$

$$\dot{L} = \boldsymbol{s}^{\mathrm{T}}\varsigma_0 - \boldsymbol{s}^{\mathrm{T}}\boldsymbol{K}_{\mathrm{v}}\boldsymbol{s}$$

It indicates that, with $\boldsymbol{K}_{\mathrm{v}}$, the stability of control system depends on ς_0, i.e. the approximation precision and the magnitude of $\boldsymbol{\tau}_{\mathrm{d}}$.

RBF network can be adopted to approximate $\boldsymbol{f}(\boldsymbol{x})$. The desired algorithm of RBF network is:

$$\phi_i = g(\|\boldsymbol{x} - \boldsymbol{c}_i\|^2/\sigma_i^2), \quad i = 1,2,\cdots,n$$

$$\boldsymbol{y} = \boldsymbol{W}^{*\mathrm{T}}\boldsymbol{\varphi}(\boldsymbol{x}), \quad \boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{W}^{*\mathrm{T}}\boldsymbol{\varphi}(\boldsymbol{x}) + \boldsymbol{\varepsilon}$$

where $\boldsymbol{x}$ is the input state of network, $\boldsymbol{\varphi}(\boldsymbol{x})=[\phi_1 \quad \phi_2 \quad \cdots \quad \phi_n]^{\mathrm{T}}$, $\boldsymbol{\varepsilon}$ is the approximation error of neural network, $\boldsymbol{W}^*$ is the weight vector of desired RBF network.

10.2.2 Sliding Mode Control with Respect to the Approximation of $\boldsymbol{f}(\boldsymbol{x})$

10.2.2.1 Design of Controller

RBF network is adopted to approximate $\boldsymbol{f}(\boldsymbol{x})$, therefore, the output of RBF network is:

$$\hat{\boldsymbol{f}}(\boldsymbol{x})=\hat{\boldsymbol{W}}^{\mathrm{T}}\boldsymbol{\varphi}(\boldsymbol{x}) \tag{10.15}$$

Select

$$\tilde{\boldsymbol{W}}=\boldsymbol{W}^*-\hat{\boldsymbol{W}}, \quad \|\boldsymbol{W}^*\|_{\mathrm{F}}\leqslant W_{\max}$$

Therefore, we have

$$\varsigma_0=\tilde{\boldsymbol{f}}(\boldsymbol{x})+\boldsymbol{\tau}_{\mathrm{d}}=\tilde{\boldsymbol{W}}^{\mathrm{T}}\boldsymbol{\varphi}(\boldsymbol{x})+\boldsymbol{\varepsilon}+\boldsymbol{\tau}_{\mathrm{d}}$$

Controller is designed as[4]:

$$\boldsymbol{\tau}=\hat{\boldsymbol{f}}(\boldsymbol{x})+\boldsymbol{K}_{\mathrm{v}}\boldsymbol{s}-\boldsymbol{v} \tag{10.16}$$

where $\boldsymbol{v}$ is the robust element required to overcome the network approximation error $\boldsymbol{\varepsilon}$ and the disturbance $\boldsymbol{\tau}_{\mathrm{d}}$.

From Eqs. (10.16) and (10.12), we have

$$\boldsymbol{H}\dot{\boldsymbol{s}}=-(\boldsymbol{K}_{\mathrm{v}}+\boldsymbol{C})\boldsymbol{s}+\tilde{\boldsymbol{W}}^{\mathrm{T}}\boldsymbol{\varphi}(\boldsymbol{x})+(\boldsymbol{\varepsilon}+\boldsymbol{\tau}_{\mathrm{d}})+\boldsymbol{v}=-(\boldsymbol{K}_{\mathrm{v}}+\boldsymbol{C})\boldsymbol{s}+\varsigma_1 \tag{10.17}$$

where $\varsigma_1=\tilde{\boldsymbol{W}}^{\mathrm{T}}\boldsymbol{\varphi}(\boldsymbol{x})+(\boldsymbol{\varepsilon}+\boldsymbol{\tau}_{\mathrm{d}})+\boldsymbol{v}$.

The robust element $\boldsymbol{v}$ is designed as:

$$\boldsymbol{v}=-(\varepsilon_{\mathrm{N}}+b_{\mathrm{d}})\mathrm{sgn}(\boldsymbol{s}) \tag{10.18}$$

where $\|\boldsymbol{\varepsilon}\|\leqslant\varepsilon_{\mathrm{N}}$, $\|\boldsymbol{\tau}_{\mathrm{d}}\|\leqslant b_{\mathrm{d}}$.

10.2.2.2 Stability Analysis

Select the Lyapunov function as

$$L=\frac{1}{2}\boldsymbol{s}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{s}+\frac{1}{2}\mathrm{tr}(\tilde{\boldsymbol{W}}^{\mathrm{T}}\boldsymbol{F}_{\mathrm{W}}^{-1}\tilde{\boldsymbol{W}})$$

where $\boldsymbol{H}$ and $\boldsymbol{F}_{\mathrm{W}}$ are positive matrices. Therefore, we have

$$\dot{L} = \boldsymbol{s}^{\mathrm{T}}\boldsymbol{H}\dot{\boldsymbol{s}} + \frac{1}{2}\boldsymbol{s}^{\mathrm{T}}\dot{\boldsymbol{H}}\boldsymbol{s} + \mathrm{tr}(\tilde{\boldsymbol{W}}^{\mathrm{T}}\boldsymbol{F}_{\mathrm{W}}^{-1}\dot{\tilde{\boldsymbol{W}}})$$

From Eq. (10.17), we have

$$\dot{L} = -\boldsymbol{s}^{\mathrm{T}}\boldsymbol{K}_{\mathrm{v}}\boldsymbol{s} + \frac{1}{2}\boldsymbol{s}^{\mathrm{T}}(\dot{\boldsymbol{H}} - 2\boldsymbol{C})\boldsymbol{s} + \mathrm{tr}\tilde{\boldsymbol{W}}^{\mathrm{T}}(\boldsymbol{F}_{\mathrm{W}}^{-1}\dot{\tilde{\boldsymbol{W}}} + \boldsymbol{\varphi}\boldsymbol{s}^{\mathrm{T}}) + \boldsymbol{s}^{\mathrm{T}}(\boldsymbol{\varepsilon} + \boldsymbol{\tau}_{\mathrm{d}} + \boldsymbol{v})$$

We know that the manipulator has the characteristic of $\boldsymbol{s}^{\mathrm{T}}(\dot{\boldsymbol{H}} - 2\boldsymbol{C})\boldsymbol{s} = 0$. Select $\dot{\tilde{\boldsymbol{W}}} = -\boldsymbol{F}_{\mathrm{W}}\boldsymbol{\varphi}\boldsymbol{s}^{\mathrm{T}}$, i.e., the adaptive rule of the network is

$$\dot{\hat{\boldsymbol{W}}} = \boldsymbol{F}_{\mathrm{W}}\boldsymbol{\varphi}\boldsymbol{s}^{\mathrm{T}} \tag{10.19}$$

Therefore,

$$\dot{L} = -\boldsymbol{s}^{\mathrm{T}}\boldsymbol{K}_{\mathrm{v}}\boldsymbol{s} + \boldsymbol{s}^{\mathrm{T}}(\boldsymbol{\varepsilon} + \boldsymbol{\tau}_{\mathrm{d}} + \boldsymbol{v})$$

Because

$$\boldsymbol{s}^{\mathrm{T}}(\boldsymbol{\varepsilon} + \boldsymbol{\tau}_{\mathrm{d}} + \boldsymbol{v}) = \boldsymbol{s}^{\mathrm{T}}(\boldsymbol{\varepsilon} + \boldsymbol{\tau}_{\mathrm{d}}) + \boldsymbol{s}^{\mathrm{T}}\boldsymbol{v} = \boldsymbol{s}^{\mathrm{T}}(\boldsymbol{\varepsilon} + \boldsymbol{\tau}_{\mathrm{d}}) - \|\boldsymbol{s}\|(\varepsilon_{\mathrm{N}} + \boldsymbol{b}_{\mathrm{d}}) \leqslant 0$$

We have

$$\dot{L} \leqslant 0$$

10.2.3 Simulation Example

The kinetic equation of the two-joint manipulator is:

$$\boldsymbol{H}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q}) = \boldsymbol{\tau} - \boldsymbol{F}(\dot{\boldsymbol{q}}) - \boldsymbol{\tau}_{\mathrm{d}}$$

where

$$\boldsymbol{H}(\boldsymbol{q}) = \begin{bmatrix} p_1 + p_2 + 2p_3\cos q_2 & p_2 + p_3\cos q_2 \\ p_2 + p_3\cos q_2 & p_2 \end{bmatrix}$$

$$\boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}}) = \begin{bmatrix} -p_3\dot{q}_2\sin q_2 & -p_3(\dot{q}_1 + \dot{q}_2)\sin q_2 \\ p_3\dot{q}_1\sin q_2 & 0 \end{bmatrix}$$

$$\boldsymbol{G}(\boldsymbol{q}) = \begin{bmatrix} p_4 g\cos q_1 + p_5 g\cos(q_1 + q_2) \\ p_5 g\cos(q_1 + q_2) \end{bmatrix}$$

$$\boldsymbol{F}(\dot{\boldsymbol{q}}) = 0.02\,\mathrm{sgn}(\dot{\boldsymbol{q}}), \quad \boldsymbol{\tau}_{\mathrm{d}} = [0.2\sin t \quad 0.2\sin t]^{\mathrm{T}}$$

Let $\boldsymbol{p} = [p_1 \quad p_2 \quad p_3 \quad p_4 \quad p_5] = [2.9 \quad 0.76 \quad 0.87 \quad 3.04 \quad 0.87]$. The selection of the Gauss function of the RBF network is very important to the control of the neural network. If the parameter is not suitable, then, the available mapping of the Gauss function cannot be obtained. Hence, the RBF network is unavailable. Therefore, c should be selected according to the scope of network input. We select $b = 0.20$. The initial weight matrix of the network is selected as 0 or 0.1, and the network input is selected as $\boldsymbol{z} = [\boldsymbol{e} \quad \dot{\boldsymbol{e}} \quad \boldsymbol{q}_{\mathrm{d}} \quad \dot{\boldsymbol{q}}_{\mathrm{d}} \quad \ddot{\boldsymbol{q}}_{\mathrm{d}}]$.

The initial state vector of the system is $[0.09 \quad 0 \quad -0.09 \quad 0]$. The desired position commands of the two joints are $q_{1\mathrm{d}} = 0.1\sin t$ and $q_{2\mathrm{d}} = 0.1\sin t$ respectively. The controller parameters are: $\boldsymbol{K}_{\mathrm{v}} = \mathrm{diag}\{20, 20\}$, $\boldsymbol{F}_{\mathrm{W}} = \mathrm{diag}\{15, 15\}$, $\boldsymbol{\varLambda} = \mathrm{diag}\{5, 5\}$. In the sliding robust element, select $\varepsilon_{\mathrm{N}} = 0.20$ and $b_{\mathrm{d}} = 0.10$. The controller is given in Eq. (10.16), and the adaptive rule is given in Eq. (10.19). Simulation results are shown in Fig.10.4 – Fig.10.7.

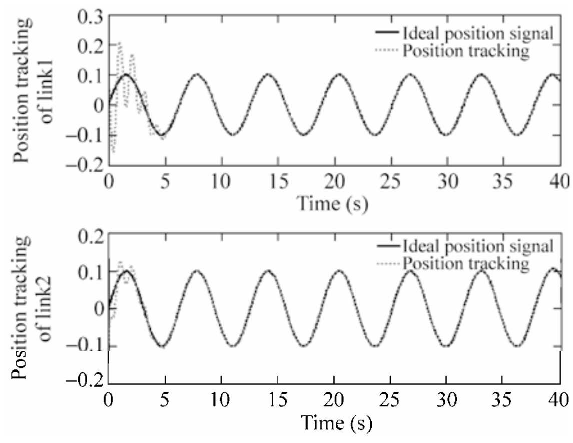


Figure 10.4 Position tracking of joints 1 and 2

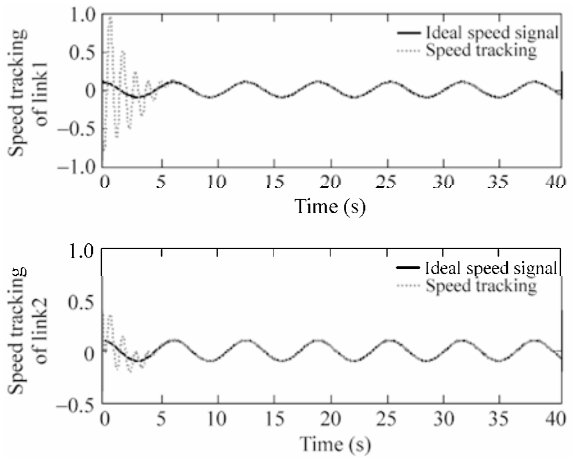


Figure 10.5 Velocity tracking of joints 1 and 2

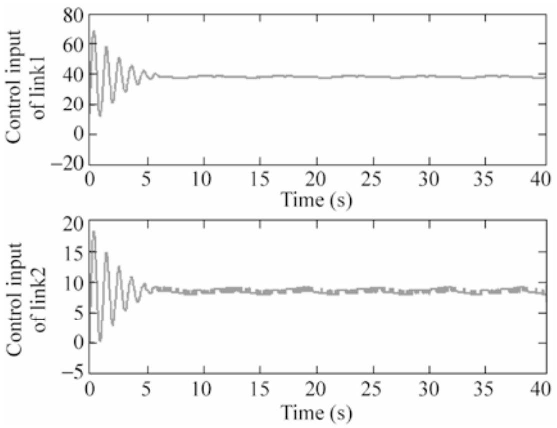


Figure 10.6 Control inputs of joints 1 and 2

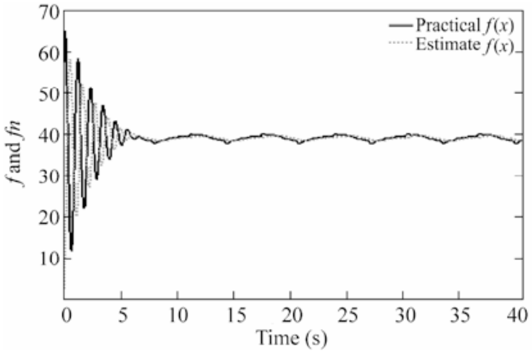
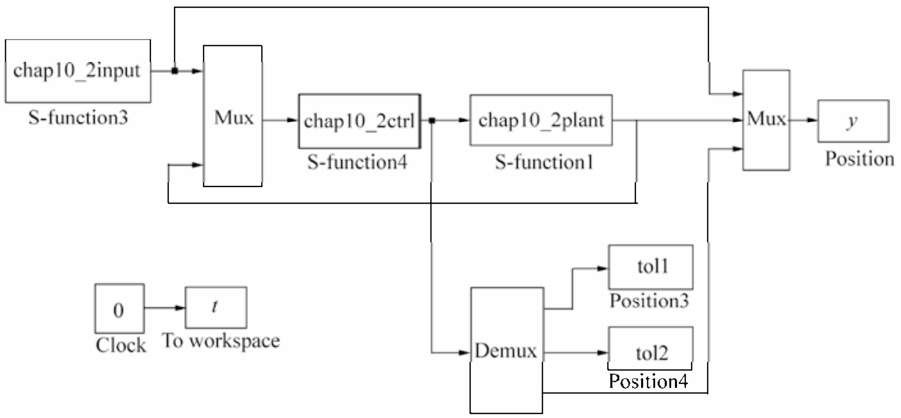


Figure 10.7 $\| \boldsymbol{f}(\boldsymbol{x}) \|$ and $\| \hat{\boldsymbol{f}}(\boldsymbol{x}) \|$ of joints 1 and 2

Simulation programs:

(1) Simulink main program: chap10_2sim.mdl



(2) Program of position commands: chap10_2input.m

```
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)
switch flag,
case 0,
   [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
   sys=mdlDerivatives(t,x,u);
case 3,
   sys=mdlOutputs(t,x,u);
case {2,4,9}
   sys=[];
otherwise
   error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 6;
sizes.NumInputs      = 0;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0  = [];
str = [];
ts  = [0 0];
function sys=mdlOutputs(t,x,u)
qd1=0.1*sin(t);
d_qd1=0.1*cos(t);
dd_qd1=-0.1*sin(t);
qd2=0.1*sin(t);
d_qd2=0.1*cos(t);
dd_qd2=-0.1*sin(t);

sys(1)=qd1;
sys(2)=d_qd1;
sys(3)=dd_qd1;
sys(4)=qd2;
sys(5)=d_qd2;
sys(6)=dd_qd2;
```

(3) Controller S function: chap10_2ctrl.m

```
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)
switch flag,
case 0,
   [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
   sys=mdlDerivatives(t,x,u);
```

```
case 3,
   sys=mdlOutputs(t,x,u);
case {2,4,9}
   sys=[];
otherwise
   error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
global node c b Fai
node=7;
c=0.1*[-1.5 -1 -0.5 0 0.5 1 1.5;
      -1.5 -1 -0.5 0 0.5 1 1.5;
      -1.5 -1 -0.5 0 0.5 1 1.5;
      -1.5 -1 -0.5 0 0.5 1 1.5;
      -1.5 -1 -0.5 0 0.5 1 1.5];
b=10;
Fai=5*eye(2);

sizes = simsizes;
sizes.NumContStates  = 2*node;
sizes.NumDiscStates  = 0;
sizes.NumOutputs    = 3;
sizes.NumInputs     = 11;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys = simsizes(sizes);
x0  = 0.1*ones(1,2*node);
str = [];
ts  = [];
function sys=mdlDerivatives(t,x,u)
global node c b Fai
qd1=u(1);
d_qd1=u(2);
dd_qd1=u(3);
qd2=u(4);
d_qd2=u(5);
dd_qd2=u(6);

q1=u(7);
d_q1=u(8);
q2=u(9);
d_q2=u(10);

q=[q1;q2];

e1=qd1-q1;
e2=qd2-q2;
de1=d_qd1-d_q1;
de2=d_qd2-d_q2;
```

```
e=[e1;e2];
de=[de1;de2];
S=de+Fai*e;

qd=[qd1;qd2];
dqd=[d_qd1;d_qd2];
dqr=dqd+Fai*e;
ddqd=[dd_qd1;dd_qd2];
ddqr=ddqd+Fai*de;

z1=[e(1);de(1);qd(1);dqd(1);ddqd(1)];
z2=[e(2);de(2);qd(2);dqd(2);ddqd(2)];
for j=1:1:node
   h1(j)=exp(-norm(z1-c(:,j))^2/(b*b));
   h2(j)=exp(-norm(z2-c(:,j))^2/(b*b));
end

Fw=15*eye(node);
for i=1:1:node
   sys(i)=15*h1(i)*S(1);
   sys(i+node)=15*h2(i)*S(2);
end
function sys=mdlOutputs(t,x,u)
global node c b Fai
qd1=u(1);
d_qd1=u(2);
dd_qd1=u(3);
qd2=u(4);
d_qd2=u(5);
dd_qd2=u(6);

q1=u(7);
d_q1=u(8);
q2=u(9);
d_q2=u(10);

q=[q1;q2];

e1=qd1-q1;
e2=qd2-q2;
de1=d_qd1-d_q1;
de2=d_qd2-d_q2;
e=[e1;e2];
de=[de1;de2];
S=de+Fai*e;

qd=[qd1;qd2];
dqd=[d_qd1;d_qd2];
dqr=dqd+Fai*e;
```

```
ddqd=[dd_qd1;dd_qd2];
ddqr=ddqd+Fai*de;

z=[e;de;qd;dqd;ddqd];
W_f1=[x(1:node)]';
W_f2=[x(node+1:node*2)]';

z1=[e(1);de(1);qd(1);dqd(1);ddqd(1)];
z2=[e(2);de(2);qd(2);dqd(2);ddqd(2)];
for j=1:1:node
    h1(j)=exp(-norm(z1-c(:,j))^2/(b*b));
    h2(j)=exp(-norm(z2-c(:,j))^2/(b*b));
end

fn=[W_f1*h1';
    W_f2*h2'];
Kv=20*eye(2);

epN=0.20;bd=0.1;
v=-(epN+bd)*sign(S);
tol=fn+Kv*S-v;

fn_norm=norm(fn);
sys(1)=tol(1);
sys(2)=tol(2);
sys(3)=fn_norm;
```

(4) Program of the plant: chap10_2plant.m

```
function [sys,x0,str,ts]=s_function(t,x,u,flag)

switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2, 4, 9 }
    sys = [];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
global p g
sizes = simsizes;
sizes.NumContStates  = 4;
sizes.NumDiscStates  = 0;
sizes.NumOutputs    = 5;
sizes.NumInputs     =3;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[0.09 0 -0.09 0];
str=[];
ts=[];

p=[2.9 0.76 0.87 3.04 0.87];
g=9.8;
function sys=mdlDerivatives(t,x,u)
global p g

H=[p(1)+p(2)+2*p(3)*cos(x(3)) p(2)+p(3)*cos(x(3));
   p(2)+p(3)*cos(x(3)) p(2)];
C=[-p(3)*x(4)*sin(x(3)) -p(3)*(x(2)+x(4))*sin(x(3));
    p(3)*x(2)*sin(x(3))  0];
G=[p(4)*g*cos(x(1))+p(5)*g*cos(x(1)+x(3));
   p(5)*g*cos(x(1)+x(3))];
dq=[x(2);x(4)];
F=0.2*sign(dq);
told=[0.1*sin(t);0.1*sin(t)];

tol=u(1:2);

S=inv(H)*(tol-C*dq-G-F-told);

sys(1)=x(2);
sys(2)=S(1);
sys(3)=x(4);
sys(4)=S(2);
function sys=mdlOutputs(t,x,u)
global p g
H=[p(1)+p(2)+2*p(3)*cos(x(3)) p(2)+p(3)*cos(x(3));
   p(2)+p(3)*cos(x(3)) p(2)];
C=[-p(3)*x(4)*sin(x(3)) -p(3)*(x(2)+x(4))*sin(x(3));
    p(3)*x(2)*sin(x(3))  0];
G=[p(4)*g*cos(x(1))+p(5)*g*cos(x(1)+x(3));
   p(5)*g*cos(x(1)+x(3))];
dq=[x(2);x(4)];
F=0.2*sign(dq);
told=[0.1*sin(t);0.1*sin(t)];

qd1=0.1*sin(t);
d_qd1=0.1*cos(t);
dd_qd1=-0.1*sin(t);
qd2=0.1*sin(t);
d_qd2=0.1*cos(t);
dd_qd2=-0.1*sin(t);
```

```
q1=x(1);
d_q1=dq(1);
q2=x(3);
d_q2=dq(2);
q=[q1;q2];
e1=qd1-q1;
e2=qd2-q2;
de1=d_qd1-d_q1;
de2=d_qd2-d_q2;
e=[e1;e2];
de=[de1;de2];
Fai=5*eye(2);
dqd=[d_qd1;d_qd2];
dqr=dqd+Fai*e;
ddqd=[dd_qd1;dd_qd2];
ddqr=ddqd+Fai*de;
f=H*ddqr+C*dqr+G+F;
f_norm=norm(f);

sys(1)=x(1);
sys(2)=x(2);
sys(3)=x(3);
sys(4)=x(4);
sys(5)=f_norm;
```

(5) Plot program: chap10_2plot.m

```
close all;

figure(1);
subplot(211);
plot(t,y(:,1),'k',t,y(:,7),'r:','linewidth',2);
xlabel('time(s)');ylabel('Position tracking for link 1');
legend('Ideal position signal','Position tracking');
subplot(212);
plot(t,y(:,4),'k',t,y(:,9),'r:','linewidth',2);
xlabel('time(s)');ylabel('Position tracking for link 2');
legend('Ideal position signal','Position tracking');

figure(2);
subplot(211);
plot(t,y(:,2),'k',t,y(:,8),'r:','linewidth',2);
xlabel('time(s)');ylabel('Speed tracking for link 1');
legend('Ideal speed signal','Speed tracking');
subplot(212);
plot(t,y(:,5),'k',t,y(:,10),'r:','linewidth',2);
xlabel('time(s)');ylabel('Speed tracking for link 2');
legend('Ideal speed signal','Speed tracking');

figure(3);
```

```
subplot(211);
plot(t,tol1(:,1),'r','linewidth',2);
xlabel('time(s)');ylabel('control input of link 1');
subplot(212);
plot(t,tol2(:,1),'r','linewidth',2);
xlabel('time(s)');ylabel('control input of link 2');

figure(4);
plot(t,y(:,11),'k',t,y(:,12),'r:','linewidth',2);
xlabel('time(s)');ylabel('f and fn');
legend('Practical f(x)','Estimate f(x)');
```

References

[1] Hartman EJ, Keeler JD, Kowalski JM. Layered neural networks with Gaussian hidden units as universal approximations. Neural computation, 1990, 2(2): 210 – 215

[2] Park J, Sandberg IW. Universal approximation using radial-basis-function networks. Neural computation, 1991,3: 246 – 257

[3] Ge SS, Lee TH, Harris CJ. Adaptive Neural Network Control of Robotic Manipulators. World Scientific, London, 1998

[4] Lewis FL, Liu K, Yesildirek A. Neural Net Robot Controller with Guaranteed Tracking Performance. IEEE Transactions on Neural Networks, 1995, 6(3): 703 – 715