1 Introduction

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 the concept of sliding mode control and illustrates the attendant features of robustness and performance specification using a straightforward example, several typical sliding mode controllers for continuous system are given, a concrete stability analysis, simulation examples and Matlab programs are given too.

Keywords sliding mode control, sliding surface, Reaching Law, quasi-sliding mode, equivalent control

One of the methods used to solve control problems are the sliding mode techniques. These techniques are generating greater interest.

This book provides the reader with an introduction to classical sliding mode control design examples. Fully worked design examples, which can be used as tutorial material, are included. Industrial case studies, which present the results of sliding mode controller implementations, are used to illustrate successful practical applications of the theory.

Typically, discrepancies may occur between the actual plant and the mathematical model developed for the controller design. These mismatches may be due to various factors. The engineer's role is to ensure required performance levels despite such mismatches. A set of robust control methods have been developed to eliminate any discrepancy. One such approach to the robust control controller design is called the sliding mode control (SMC) methodology. This is a specific type of variable structure control system (VSCS).

In the early 1950s, Emelyanov and several co-researchers such as Utkins and Itkis[1] from the Soviet Union, proposed and elaborated the variable structure control

(VSC) with sliding mode control. During the past decades, VSC and SMC have generated significant interest in the control research community.

SMC has been applied into general design method being examined for wide spectrum of system types including nonlinear system, multi-input multi-output (MIMO) systems, discrete-time models, large-scale and infinite-dimension systems, and stochastic systems. The most eminent feature of SMC is it is completely insensitive to parametric uncertainty and external disturbances during sliding mode[2].

VSC utilizes a high-speed switching control law to achieve two objectives. Firstly, it drives the nonlinear plant's state trajectory onto a specified and user-chosen surface in the state space which is called the sliding or switching surface. This surface is called the switching surface because a control path has one gain if the state trajectory of the plant is "above" the surface and a different gain if the trajectory drops "below" the surface. Secondly, it maintains the plant's state trajectory on this surface for all subsequent times. During the process, the control system's structure varies from one to another and thereby earning the name variable structure control. The control is also called as the sliding mode control[3] to emphasize the importance of the sliding mode.

Under sliding mode control, the system is designed to drive and then constrain the system state to lie within a neighborhood of the switching function. Its two main advantages are (1) the dynamic behavior of the system may be tailored by the particular choice of switching function, and (2) the closed-loop response becomes totally insensitive to a particular class of uncertainty. Also, the ability to specify performance directly makes sliding mode control attractive from the design perspective.

Trajectory of a system can be stabilized by a sliding mode controller. The system states "slides" along the line $s = 0$ after the initial reaching phase. The particular $s = 0$ surface is chosen because it has desirable reduced-order dynamics when constrained to it. In this case, the $s = cx_1 + \dot{x}_1,\ c > 0$ surface corresponds to the first-order LTI system $\dot{x}_1 = -cx_1,$ which has an exponentially stable origin. Now, we consider a simple example of the sliding mode controller design as under.

Consider a plant as

$$J\ddot{\theta}(t) = u(t) \tag{1.1}$$

where J is the inertia moment, $\ddot{\theta}(t)$ is the angle signal, and $u(t)$ is the control input.

Firstly, we design the sliding mode function as

$$s(t) = ce(t) + \dot{e}(t) \tag{1.2}$$

where c must satisfy the Hurwitz condition, $c > 0.$

The tracking error and its derivative value are

$$e(t) = \theta(t) - \theta_{\mathrm{d}}(t),\ \ \dot{e}(t) = \dot{\theta}(t) - \dot{\theta}_{\mathrm{d}}(t)$$

where $\theta(t)$ is the practical position signal, and $\theta_{\mathrm{d}}(t)$ is the ideal position signal.

Therefore, we have

$$\dot{s}(t) = c\dot{e}(t) + \ddot{e}(t) = c\dot{e}(t) + \ddot{\theta}(t) - \ddot{\theta}_{\mathrm{d}}(t) = c\dot{e}(t) + \frac{1}{J}u - \ddot{\theta}_{\mathrm{d}}(t) \tag{1.3}$$

and

$$s\dot{s} = s\left(c\dot{e} + \frac{1}{J}u - \ddot{\theta}_{\mathrm{d}}\right)$$

Secondly, to satisfy the condition $s\dot{s} < 0$, we design the sliding mode controller as

$$u(t) = J(-c\dot{e} + \ddot{\theta}_{\mathrm{d}} - \eta\,\mathrm{sgn}(s)),\ \ \mathrm{sgn}(s) = \begin{cases} 1, & s > 0 \\ 0, & s = 0 \\ -1, & s < 0 \end{cases} \tag{1.4}$$

Then, we get

$$s\dot{s} = -\eta\,|\,s\,| < 0$$

A simulation example is presented for explanation. Consider the plant as

$$J\ddot{\theta}(t) = u(t)$$

where $J = 10$.

The initial state is set as [0.5 1.0] after choosing the position ideal signal $\theta_{\mathrm{d}}(t) = \sin t$. Using controller Eq. (1.4) wherein $c = 0.5$, $\eta = 0.5$ the results are derived as shown in Fig. 1.1 – Fig. 1.3.

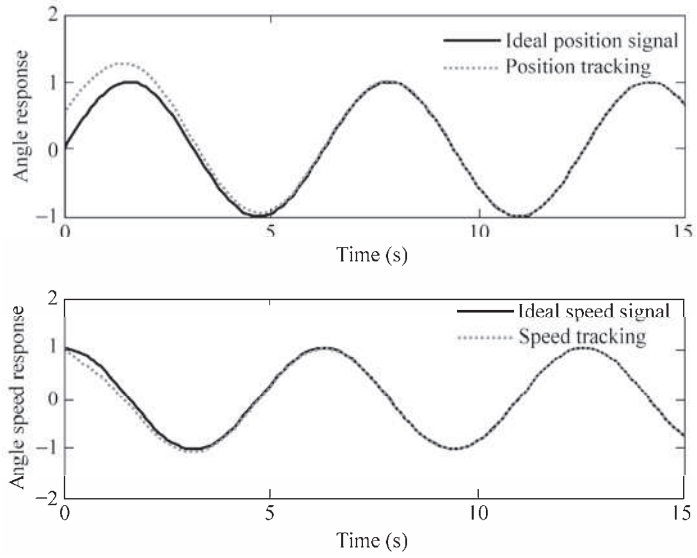


Figure 1.1 Position and speed tracking

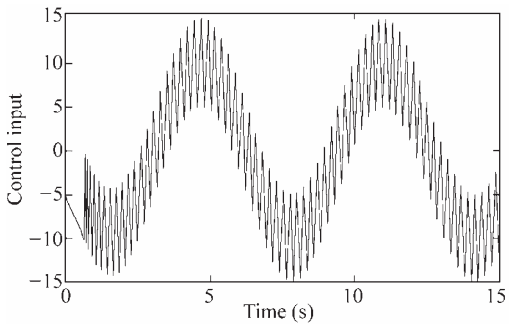


Figure 1.2 Control input

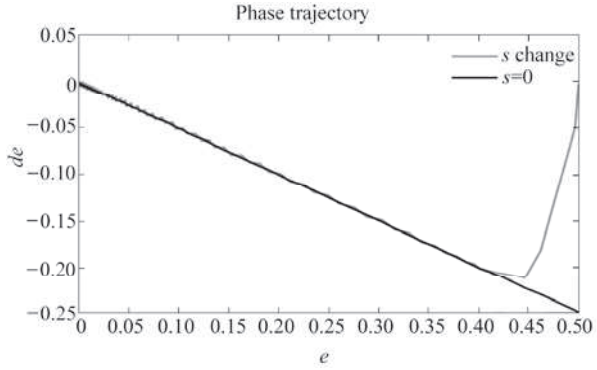
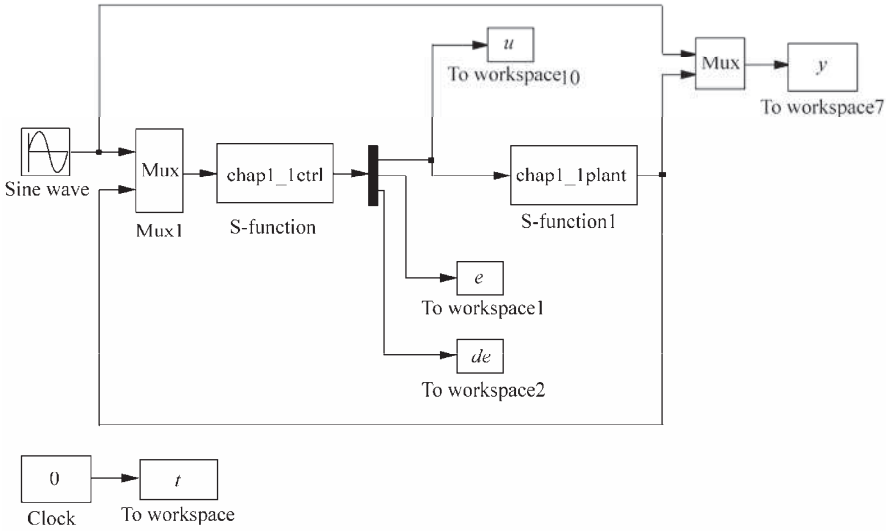


Figure 1.3 Phase trajectory

Simulation programs:

(1) Simulink main program: chap1_1sim.mdl



(2) Controller: chap1_1ctrl.m

```
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)
switch flag,
case 0,
   [sys,x0,str,ts]=mdlInitializeSizes;
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     = 3;
sizes.NumInputs      = 3;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys = simsizes(sizes);
x0  = [];
str = [];
ts  = [];
function sys=mdlOutputs(t,x,u)
thd=u(1);
dthd=cos(t);
ddthd=-sin(t);

th=u(2);
dth=u(3);

c=0.5;
e=th-thd;
de=dth-dthd;
s=c*e+de;

J=10;
xite=0.50;
ut=J*(-c*de+ddthd-xite*sign(s));

sys(1)=ut;
sys(2)=e;
sys(3)=de;
```

(3) Plant: chap1_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=[0.5 1.0];
str=[];
ts=[];
function sys=mdlDerivatives(t,x,u)
J=10;
sys(1)=x(2);
sys(2)=1/J*u;
function sys=mdlOutputs(t,x,u)
sys(1)=x(1);
sys(2)=x(2);
```

(4) Plot program: chap1_1plot.m

```
close all;

figure(1);
subplot(211);
plot(t,y(:,1),'k',t,y(:,2),'r:','linewidth',2);
legend('Ideal position signal','Position tracking');
xlabel('time(s)');ylabel('Angle response');
subplot(212);
plot(t,cos(t),'k',t,y(:,3),'r:','linewidth',2);
legend('Ideal speed signal','Speed tracking');
xlabel('time(s)');ylabel('Angle speed response');

figure(2);
plot(t,u(:,1),'k','linewidth',0.01);
xlabel('time(s)');ylabel('Control input');

c=0.5;
figure(3);
plot(e,de,'r',e,-c'.*e,'k','linewidth',2);
xlabel('e');ylabel('de');
legend('s change','s=0');
title('phase trajectory');
```

Sliding mode control is a nonlinear control method that alters the dynamics of a nonlinear system by the multiple control structures are designed so as to ensure that trajectories always move towards a switching condition. Therefore, the ultimate trajectory will not exist entirely within one control structure. The state-feedback control law is not a continuous function of time. Instead, it switches from one continuous structure to another based on the current position in the state space. Hence, sliding mode control is a variable structure control method. The multiple control structures are designed so as to ensure that trajectories always move towards a switching condition. Therefore, the ultimate trajectory will not exist entirely within one control structure. Instead, the ultimate trajectory will slide along the boundaries of the control structures. The motion of the system as it slides along these boundaries is called a sliding mode[3] and the geometrical locus consisting of the boundaries is called the sliding (hyper) surface. Figure 1.3 shows an example of the trajectory of a system under sliding mode control. The sliding surface is described by $s=0$, and the sliding mode along the surface commences after the finite time when system trajectories have reached the surface. In the context of modern control theory, any variable structure system like a system under SMC, may be viewed as a special case of a hybrid dynamical system.

Intuitively, sliding mode control uses practically infinite gain to force the trajectories of a dynamic system to slide along the restricted sliding mode subspace. Trajectories from this reduced-order sliding mode have desirable properties (e.g., the system naturally slides along it until it comes to rest at a desired equilibrium). The main strength of sliding mode control is its robustness. Because the control can be as simple as a switching between two states, it need not be precise and will not be sensitive to parameter variations that enter into the control channel. Additionally, because the control law is not a continuous function, the sliding mode can be reached in finite time (i.e., better than asymptotic behavior).

There are two steps in the SMC design. The first step is designing a sliding surface so that the plant restricted to the sliding surface has a desired system response. This means the state variables of the plant dynamics are constrained to satisfy another set of equations which define the so-called switching surface. The second step is constructing a switched feedback gains necessary to drive the plant's state trajectory to the sliding surface. These constructions are built on the generalized Lyapunov stability theory.

1.1 Parameters of Sliding Surface Design

For linear system

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}u,\ \ \boldsymbol{x} \in \mathbf{R}^n,\ \ u \in \mathbf{R} \tag{1.5}$$

where $\boldsymbol{x}$ is system state, $\boldsymbol{A}$ is an $n \times n$ matrix, $\boldsymbol{b}$ is an $n \times 1$ vector, and u is control input. A sliding variable can be designed as

$$\boldsymbol{s}(\boldsymbol{x}) = \boldsymbol{C}^{\mathrm{T}}\boldsymbol{x} = \sum_{i=1}^{n} c_i x_i = \sum_{i=1}^{n-1} c_i x_i + x_n \tag{1.6}$$

where $\boldsymbol{x}$ is state vector, $\boldsymbol{C} = [c_1 \quad \cdots \quad c_{n-1} \quad 1]^{\mathrm{T}}$.

In sliding mode control, parameters $c_1, c_2, \cdots, c_{n-1}$ should be selected so that the polynomial $p^{n-1} + c_{n-1}p^{n-2} + \cdots + c_2 p + c_1$ is a Hurwitz polynomial, where p is a Laplace operator.

For example, $n = 2$, $\boldsymbol{s}(\boldsymbol{x}) = c_1 x_1 + x_2$, to satisfy the condition that the polynomial $p + c_1$ is Hurwitz, the eigenvalue of $p + c_1 = 0$ should have a negative real part, i.e. $c_1 > 0$.

Consider another example, $n = 3$, $\boldsymbol{s}(\boldsymbol{x}) = c_1 x_1 + c_2 x_2 + x_3$, to satisfy the condition that the polynomial $p^2 + c_2 p + c_1$ is Hurwitz, the eigenvalue of $p^2 + c_2 p + c_1 = 0$ should have a negative real part. For a positive constant λ in $(p + \lambda)^2 = 0$, we can get $p^2 + 2\lambda p + \lambda^2 = 0$. Therefore, we have $c_2 = 2\lambda$, $c_1 = \lambda^2$.

1.2 Sliding Mode Control Based on Reaching Law

Sliding mode based on reaching law includes reaching phase and sliding phase. The reaching phase drive system is to maintain a stable manifold and the sliding phase drive system ensures slide to equilibrium. The idea of sliding mode can be described as Fig. 1.4.

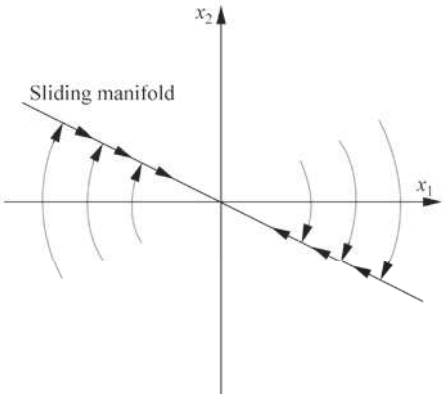


Figure 1.4 The idea of sliding mode

1.2.1 Classical Reaching Laws

(1) Constant Rate Reaching Law

$$\dot{s} = -\varepsilon \operatorname{sgn}(s), \quad \varepsilon > 0 \tag{1.7}$$

where ε represents a constant rate.

This law constrains the switching variable to reach the switching manifold s at a constant rate ε. The merit of this reaching law is its simplicity. But, as will be shown later, if ε is too small, the reaching time will be too long. On the other hand, too large a ε will cause severe chattering.

(2) Exponential Reaching Law

$$\dot{s} = -\varepsilon \operatorname{sgn}(s) - ks, \quad \varepsilon > 0, \quad k > 0 \tag{1.8}$$

where $\dot{s} = -ks$ is exponential term, and its solution is $s = s(0)\mathrm{e}^{-kt}$.

Clearly, by adding the proportional rate term $-ks$, the state is forced to approach the switching manifolds faster when s is large.

(3) Power Rate Reaching Law

$$\dot{s} = -k \mid s \mid^{\alpha} \operatorname{sgn}(s), \quad k > 0, \quad 1 > \alpha > 0 \tag{1.9}$$

This reaching law increases the reaching speed when the state is far away from the switching manifold. However, it reduces the rate when the state is near the manifold. The result is a fast and low chattering reaching mode.

(4) General Reaching Law

$$\dot{s} = -\varepsilon \operatorname{sgn}(s) - f(s), \quad \varepsilon > 0 \tag{1.10}$$

where $f(0) = 0$ and $sf(s) > 0$ when $s \neq 0$.

It is evident that the above four reaching laws can satisfy the sliding mode arrived condition $s\dot{s} < 0$.

1.2.2 Controller Design

1.2.2.1 System Description

The plant is

$$\ddot{\theta}(t) = -f(\theta, t) + bu(t) \tag{1.11}$$

where $f(\theta, t)$ and b are known and $b > 0$.

The sliding mode function is

$$s(t) = ce(t) + \dot{e}(t) \tag{1.12}$$

where c must satisfy Hurwitz condition $c > 0$.

The tracking error and its derivative value is

$$e(t) = r - \theta(t), \quad \dot{e}(t) = \dot{r} - \dot{\theta}(t)$$

where r is the ideal position signal.

Therefore, we have

$$\begin{aligned}\dot{s}(t) &= c\dot{e}(t) + \ddot{e}(t) = c(\dot{r} - \dot{\theta}(t)) + (\ddot{r} - \ddot{\theta}(t)) \\ &= c(\dot{r} - \dot{\theta}(t)) + (\ddot{r} + f(\theta, t) - bu(t))\end{aligned} \tag{1.13}$$

According to the exponential reaching law, we have

$$\dot{s} = -\varepsilon \operatorname{sgn} s - ks, \quad \varepsilon > 0, \quad k > 0 \tag{1.14}$$

From Eqs. (1.13) and (1.14), we have

$$c(\dot{r} - \dot{\theta}(t)) + (\ddot{r} + f(\theta, t) - bu(t)) = -\varepsilon \operatorname{sgn} s - ks$$

Then we can get the sliding mode controller as

$$u(t) = \frac{1}{b}(\varepsilon \operatorname{sgn}(s) + ks + c(\dot{r} - \dot{\theta}(t)) + \ddot{r} + f(\theta, t)) \tag{1.15}$$

1.2.2.2 Simulation Example

Consider the plant as

$$\ddot{\theta}(t) = -f(\theta, t) + bu(t)$$

where $\ddot{\theta}(t) = -25\dot{\theta}$, $b = 133$.

Choosing position ideal signal $r(t) = \sin t$, the initial state is set as [–0.15 –0.15], using controller Eq. (1.15), $c = 15$, $\varepsilon = 5$, $k = 10$, the results can be seen in Fig. 1.5 – Fig. 1.7.

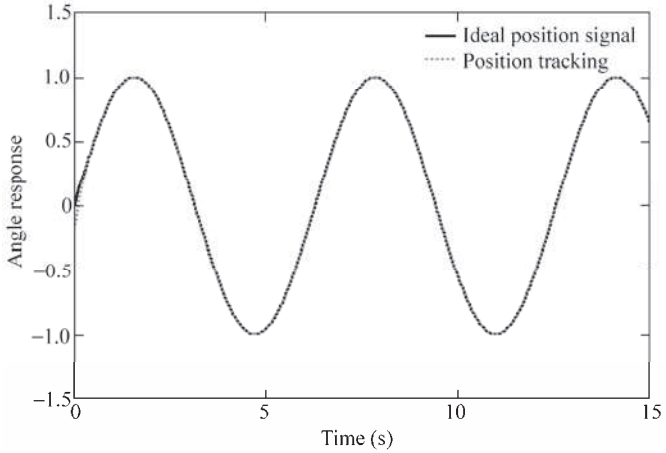


Figure 1.5 Position tracking

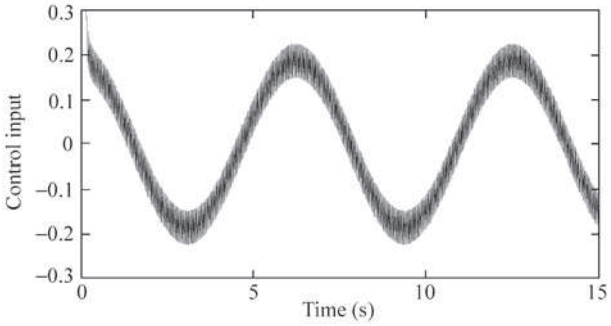


Figure 1.6 Control input

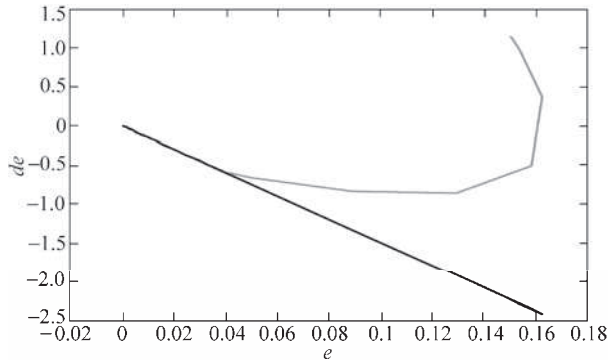
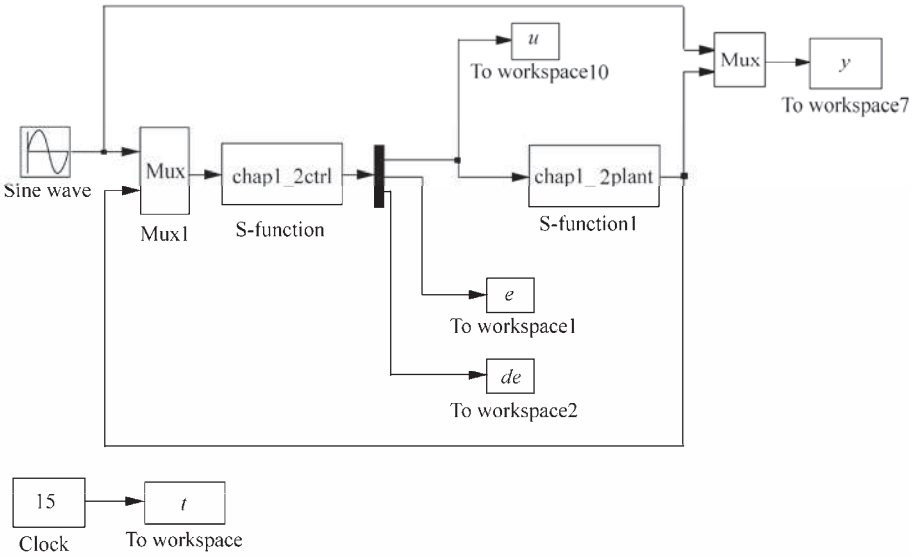


Figure 1.7 Phase trajectory

Simulation programs:

(1) Simulink main program: chap1_2sim.mdl



(2) Controller: chap1_2ctrl.m

```
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)
switch flag,
case 0,
   [sys,x0,str,ts]=mdlInitializeSizes;
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     = 3;
sizes.NumInputs      = 3;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys = simsizes(sizes);
x0  = [];
str = [];
ts  = [];
function sys=mdlOutputs(t,x,u)
r=u(1);
dr=cos(t);
ddr=-sin(t);

th=u(2);
dth=u(3);

c=15;
e=r-th;
de=dr-dth;
s=c*e+de;

fx=25*dth;
b=133;

epc=5;k=10;
ut=1/b*(epc*sign(s)+k*s+c*de+ddr+fx);

sys(1)=ut;
sys(2)=e;
sys(3)=de;
```

(3) Plant: chap1_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
sizes = simsizes;
sizes.NumContStates  = 2;
sizes.NumDiscStates  = 0;
sizes.NumOutputs    = 2;
sizes.NumInputs     = 1;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[-0.15 -0.15];
str=[];
ts=[];
function sys=mdlDerivatives(t,x,u)
sys(1)=x(2);
sys(2)=-25*x(2)+133*u;
function sys=mdlOutputs(t,x,u)
sys(1)=x(1);
sys(2)=x(2);
```

(4) Plot program: chap1_2plot.m

```
close all;

figure(1);
plot(t,y(:,1),'k',t,y(:,2),'r:','linewidth',2);
legend('Ideal position signal','Position tracking');
xlabel('time(s)');ylabel('Angle response');

figure(2);
plot(t,u(:,1),'k','linewidth',0.01);
xlabel('time(s)');ylabel('Control input');

c=15;
figure(3);
plot(e,de,'r',e,-c'.*e,'k','linewidth',2);
xlabel('e');ylabel('de');
```

1.3 Robust Sliding Mode Control Based on Reaching Law

1.3.1 System Description

The plant is

$$\ddot{\theta}(t) = -f(\theta,t) + bu(t) + d(t) \tag{1.16}$$

where $f(\theta,t)$ and b are known and $b > 0$, $d(t)$ is the disturbance.

The sliding mode function is

$$s(t) = ce(t) + \dot{e}(t) \tag{1.17}$$

where c must satisfy Hurwitz condition $c > 0$.

The tracking error and its derivative value is

$$e(t) = r - \theta(t), \quad \dot{e}(t) = \dot{r} - \dot{\theta}(t)$$

where r is the ideal position signal.

Therefore, we have

$$\begin{aligned} \dot{s}(t) &= c\dot{e}(t) + \ddot{e}(t) = c(\dot{r} - \dot{\theta}(t)) + (\ddot{r} - \ddot{\theta}(t)) \\ &= c(\dot{r} - \dot{\theta}(t)) + (\ddot{r} + f - bu - d) \end{aligned} \tag{1.18}$$

Using the exponential reaching law, we have

$$\dot{s} = -\varepsilon \operatorname{sgn}(s) - ks, \quad \varepsilon > 0, \quad k > 0 \tag{1.19}$$

From Eqs. (1.18) and (1.19), we have

$$c(\dot{r} - \dot{\theta}) + (\ddot{r} + f - bu - d) = -\varepsilon \operatorname{sgn}(s) - ks$$

If we design the sliding mode controller as

$$u(t) = \frac{1}{b}(\varepsilon \operatorname{sgn}(s) + ks + c(\dot{r} - \dot{\theta}) + \ddot{r} + f - d) \tag{1.20}$$

Obviously, all quantities on the right-hand side of Eq. (1.20) are known except the disturbance d, which is unknown. Thus the control law Eq. (1.20) is incomplete. To solve this problem, d in Eq. (1.20) is replaced by a conservative known quantity d_c.

Then we can get the sliding mode controller as

$$u(t) = \frac{1}{b}(\varepsilon \operatorname{sgn}(s) + ks + c(\dot{r} - \dot{\theta}) + \ddot{r} + f - d_c) \tag{1.21}$$

where, d_c is chosen to guarantee the reaching condition.

Substituting Eq. (1.21) into Eq. (1.18) and simplifying the result, we get

$$\dot{s}(t) = -\varepsilon \operatorname{sgn}(s) - ks + d_c - d \tag{1.22}$$

The term d_c can be chosen to ensure the reaching condition. It is reasonable to assume that d is bounded, therefore, so is d_c. That is

$$d_L \leqslant d(t) \leqslant d_U \tag{1.23}$$

where the bounds d_L and d_U are known.

Referring to Eq. (1.22), d_c is chosen according to the following logic;

When $s(t) > 0,\ \dot{s}(t) = -\varepsilon - ks + d_c - d,$ we want $\dot{s}(t) < 0,$ so let $d_c = d_L$

When $s(t) < 0,\ \dot{s}(t) = \varepsilon - ks + d_c - d,$ we want $\dot{s}(t) > 0,$ so let $d_c = d_U$

Therefore, if we define $d_1 = \dfrac{d_U - d_L}{2},\ d_2 = \dfrac{d_U + d_L}{2},$ then we can get

$$d_c = d_2 - d_1 \operatorname{sgn}(s) \tag{1.24}$$

1.3.2 Simulation Example

Consider the plant as

$$\ddot{\theta}(t) = -f(\theta, t) + bu(t) + d(t)$$

where $\ddot{\theta}(t) = -25\dot{\theta},\ b = 133,\ d(t) = 10 \sin(\pi t).$

Choosing position ideal signal $r(t) = \sin t,$ the initial state is set as [–0.15 –0.15], using controller Eq. (1.21), $c = 15,\ \varepsilon = 0.5,\ k = 10,$ the results can be seen in Fig. 1.8 – Fig. 1.10.

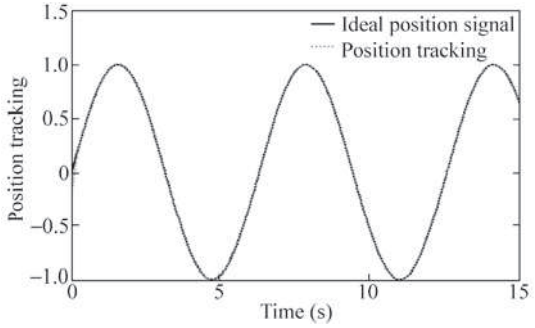


Figure 1.8 Position tracking

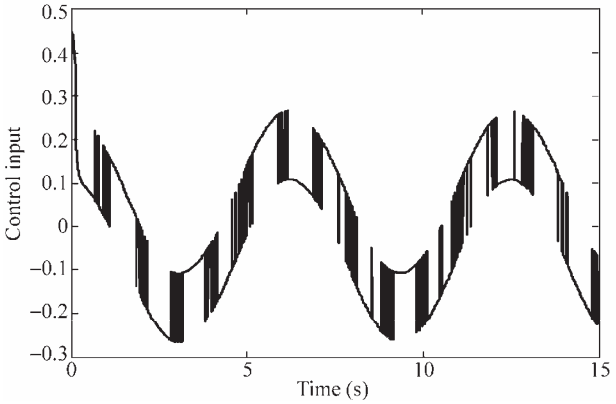


Figure 1.9 Control input

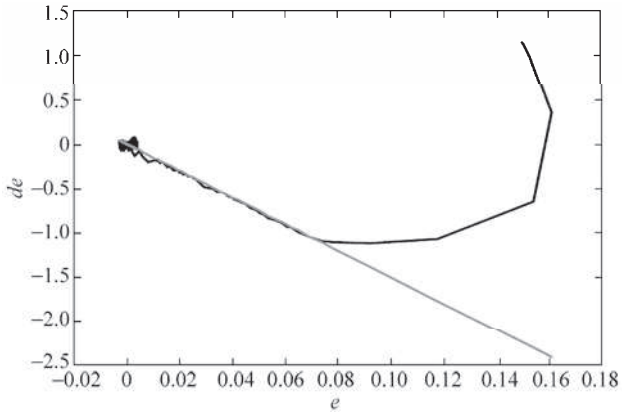
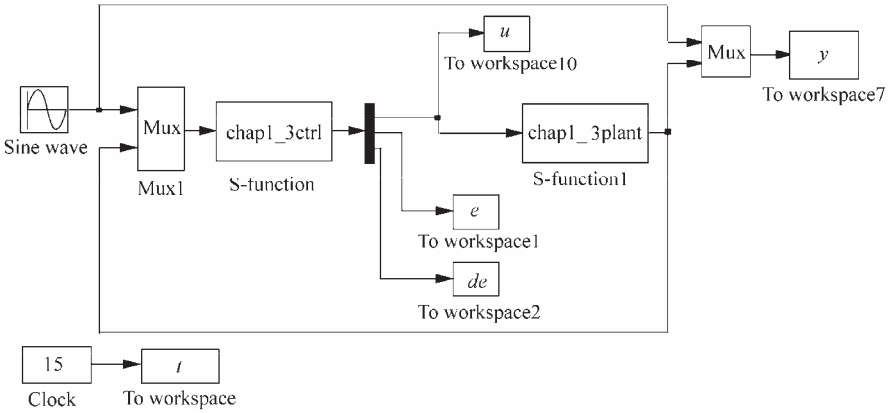


Figure 1.10 Phase trajectory

Simulation programs:

(1) Simulink main program: chap1_3sim.mdl



(2) Controller: chap1_3ctrl.m

```
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)
switch flag,
case 0,
   [sys,x0,str,ts]=mdlInitializeSizes;
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    = 3;
sizes.NumInputs     = 3;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys = simsizes(sizes);
x0  = [];
str = [];
ts  = [];
function sys=mdlOutputs(t,x,u)
r=u(1);
dr=cos(t);
ddr=-sin(t);

th=u(2);
dth=u(3);

c=15;
e=r-th;
de=dr-dth;
s=c*e+de;

fx=25*dth;
b=133;
dL=-10;dU=10;
d1=(dU-dL)/2;
d2=(dU+dL)/2;
dc=d2-d1*sign(s);

epc=0.5;k=10;
ut=1/b*(epc*sign(s)+k*s+c*de+ddr+fx-dc);

sys(1)=ut;
sys(2)=e;
sys(3)=de;
```

(3) Plant: chap1_3plant.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=[-0.15 -0.15];
str=[];
ts=[];
function sys=mdlDerivatives(t,x,u)
sys(1)=x(2);
sys(2)=-25*x(2)+133*u+10*sin(pi*t);
function sys=mdlOutputs(t,x,u)
sys(1)=x(1);
sys(2)=x(2);
```

(4) Plot program: chap1_3plot.m

```
close all;

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

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

c=15;
figure(3);
plot(e,de,'k',e,-c'.*e,'r','linewidth',2);
xlabel('e');ylabel('de');
```

1.4 Sliding Mode Robust Control Based on Upper Bound

1.4.1 System Description

Consider a second-order nonlinear inverted pendulum as follows:

$$\ddot{\theta} = f(\theta,\dot{\theta}) + \Delta f(\theta,\dot{\theta}) + g(\theta,\dot{\theta})u + \Delta g(\theta,\dot{\theta})u + d_0(t) \tag{1.25}$$

where f and g are known nonlinear functions, $u \in R$ and $y = \theta \in R$ are the control input and measurement output respectively. $d_0(t)$ is the disturbance, and $|\, d(\theta,\dot{\theta},t)\,| \leqslant D,\ D$ is a positive constant.

Let $d(\theta,\dot{\theta},t) = \Delta f(\theta,\dot{\theta}) + \Delta g(\theta,\dot{\theta})u + d_0(t),$ therefore, Eq. (1.25) can be written as

$$\ddot{\theta} = f(\theta,\dot{\theta}) + g(\theta,\dot{\theta})u + d(\theta,\dot{\theta},t) \tag{1.26}$$

1.4.2 Controller Design

Let desired position input be $\theta_{\mathrm{d}},$ and $e = \theta_{\mathrm{d}} - \theta.$ The sliding variable is selected as

$$s = \dot{e} + ce \tag{1.27}$$

where $c > 0.$ Therefore,

$$\dot{s} = \ddot{e} + c\dot{e} = \ddot{\theta}_{\mathrm{d}} - \ddot{\theta} + c\dot{e} = \ddot{\theta}_{\mathrm{d}} - f - gu - d + c\dot{e}$$

The controller is adopted as

$$u = \frac{1}{g}[-f + \ddot{\theta}_{\mathrm{d}} + c\dot{e} + \eta\,\mathrm{sgn}(s)] \tag{1.28}$$

Select the Lyapunov function as

$$L = \frac{1}{2}s^2$$

Therefore, we have

$$\begin{aligned}\dot{L} &= s\dot{s} = s(\ddot{\theta}_{\mathrm{d}} - f - gu - d + c\dot{e}) \\ &= s(\ddot{\theta}_{\mathrm{d}} - f - (-f + \ddot{\theta}_{\mathrm{d}} + c\dot{e} + \eta\,\mathrm{sgn}\,(s)) - d + c\dot{e}) \\ &= s(-d - \eta\,\mathrm{sgn}\,(s)) \\ &= -sd - \eta\,|\,s\,|\end{aligned}$$

If $\eta \geqslant D,$ then

$$\dot{L} = -sd - \eta\,|\,s\,| \leqslant 0$$

In order to restrain the chattering phenomenon, the saturated function $\text{sat}(s)$ is adopted instead of $\text{sgn}(s)$ in Eq. (1.29) as

$$\text{sat}(s)=\begin{cases}1, & s>\Delta\\ ks, & |s|\leqslant\Delta,\ k=1/\Delta\\ -1, & s<-\Delta\end{cases} \tag{1.29}$$

where Δ is the "boundary layer".

The nature of saturated function is: out of the boundary layer, switch control is selected, in the boundary layer, the usual feedback control is adopted. Therefore, the chattering phenomenon can be restrained thoroughly.

1.4.3 Simulation Example

The dynamic equation of inverted pendulum is

$$\begin{cases}\dot{x}_1=x_2\\ \dot{x}_2=f(\boldsymbol{x})+g(\boldsymbol{x})\cdot u\end{cases}$$

where
$$f(\boldsymbol{x})=\frac{g\sin x_1-mlx_2^2\cos x_1\sin x_1/(m_c+m)}{l(4/3-m\cos^2 x_1/(m_c+m))}$$

$$g(\boldsymbol{x})=\frac{\cos x_1/(m_c+m)}{l(4/3-m\cos^2 x_1/(m_c+m))}$$

where $\boldsymbol{x}=[x_1\quad x_2]$, x_1 and x_2 are the oscillation angle and oscillation rate respectively, $g=9.8\ \text{m/s}^2$, m_c is the vehicle mass, $m_c=1\ \text{kg}$, m is the mass of pendulum bar, $m=0.1\ \text{kg}$, l is one half of pendulum length, $l=0.5\ \text{m}$, u is the control input.

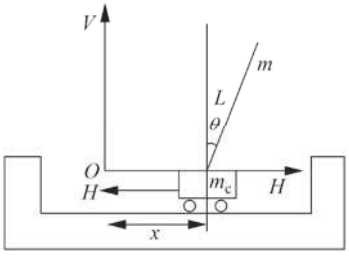


Figure 1.11 Inverted pendulum system

Let $x_1=\theta$, and the desired trajectory is $\theta_d(t)=0.1\sin(t)$. The initial state of the inverted pendulum is $[\pi/60\quad 0]$, and $\eta=0.20$. The controller is Eq. (1.28). M is a variable in the simulation program. $M=1$ indicates the controller with a switch function, and $M=2$ indicates the controller with a saturation function. The

switch function is adopted firstly, and $M=1$. The simulation results are shown in Fig. 1.12 and Fig. 1.13. In order to restrain the chattering phenomenon, the saturated function is adopted instead of switch function, and let $M=2$ in the simulation program, and let $\Delta=0.05$ in Eq. (1.29). The simulation results are shown in Fig. 1.14 and Fig. 1.15.

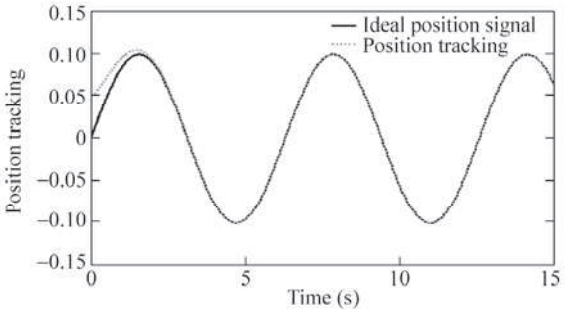


Figure 1.12 Position tracking using a switch function ($M=1$)

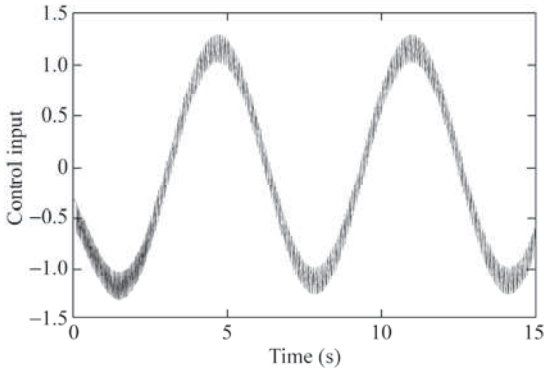


Figure 1.13 Control input using switch function ($M=1$)

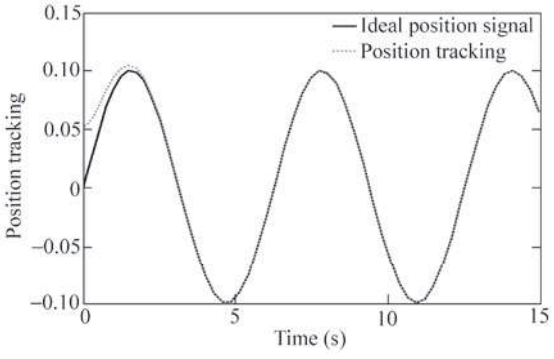


Figure 1.14 Position tracking using saturated function ($M=2$)

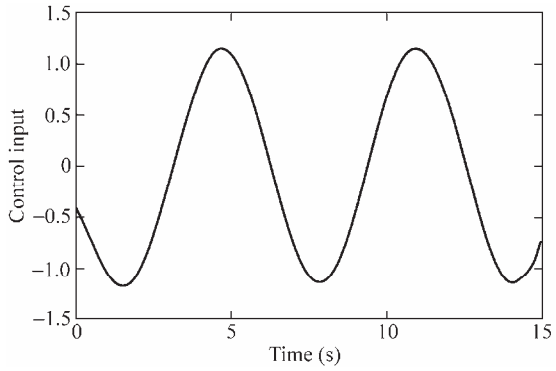
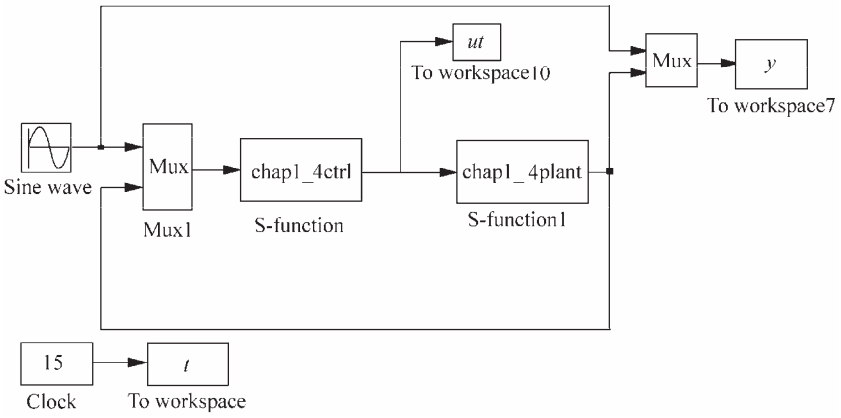


Figure 1.15 Control input using saturated function ($M=2$)

Simulation programs:

(1) Simulink main program: chap1_4sim.mdl



(2) S-function of controller: chap1_4ctrl.m

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

```
sizes.NumOutputs    = 1;
sizes.NumInputs     = 3;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys = simsizes(sizes);
x0  = [];
str = [];
ts  = [];
function sys=mdlOutputs(t,x,u)
r=0.1*sin(t);
dr=0.1*cos(t);
ddr=-0.1*sin(t);

x1=u(2);
x2=u(3);

e=r-x1;
de=dr-x2;

c=1.5;
s=c*e+de;

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

fx=g*sin(x1)-m*l*x2^2*cos(x1)*sin(x1)/(mc+m);
fx=fx/T;

gx=cos(x1)/(mc+m);
gx=gx/T;

xite=0.20;

M=2;
if M==1
   ut=1/gx*(-fx+ddr+c*de+xite*sign(s));
elseif M==2               %Saturated function
       delta=0.05;
        kk=1/delta;
      if abs(s)>delta
        sats=sign(s);
      else
        sats=kk*s;
      end
   ut=1/gx*(-fx+ddr+c*de+xite*sats);
end
sys(1)=ut;
```

(3) S-function of the plant: chap1_4plant.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)
sys(1)=x(1);
sys(2)=x(2);
```

(4) plot program: chap1_4plot.m

```
close all;

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

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

1.5 Sliding Mode Control Based on Quasi-Sliding Mode

1.5.1 Quasi-Sliding Mode

In practical engineering systems, the chattering of sliding mode control may cause damage to system components such as actuators. One way to alleviate the chattering is to use the quasi-sliding mode method which can make the state stay in a certain range at Δ neighborhood. Often we name Δ as the boundary layer.

In a continuous system, there are two common methods for the quasi-sliding mode design.

(1) Saturation function instead of sgn function

$$\text{sat}(s)=\begin{cases}1, & s>\Delta\\ ks, & |s|\leqslant\Delta\\ -1, & s<-\Delta\end{cases}\qquad k=\frac{1}{\Delta} \tag{1.30}$$

where Δ is called "the boundary layer" which is shown in Fig. 1.16. Outside the boundary layer we use switch control and inside the boundary layer we use linear feedback control.

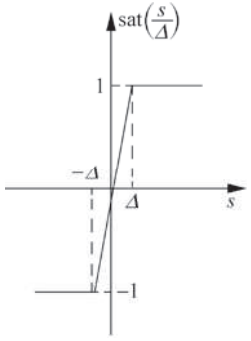


Figure 1.16 Saturation function

(2) Relay function instead of sgn function

$$\theta(s)=\frac{s}{|s|+\delta} \tag{1.31}$$

where δ is a very small positive constant.

1.5.2 Simulation Example

Consider the plant as

$$\ddot{\theta}(t) = -f(\theta, t) + bu(t) + d(t)$$

where, $\ddot{\theta}(t) = -25\dot{\theta}$, $b = 133$, $d(t) = 50\sin t$.

Using the quasi-sliding mode, the chattering can be alleviated.

(1) Use upper bound based sliding mode control law (1.28), $M = 1$ represents using sgn function. The simulation results are shown in Fig. 1.17 – Fig. 1.19.

(2) Use the quasi-sliding mode in the control law (1.28), $M = 2$ represents using saturation function. Let $\Delta = 0.05$, the simulation results are shown in Fig. 1.20 and Fig. 1.21. $M = 3$ represents using relay function, let $\delta = 0.05$, the simulation results are shown in Fig. 1.22 and Fig. 1.23.

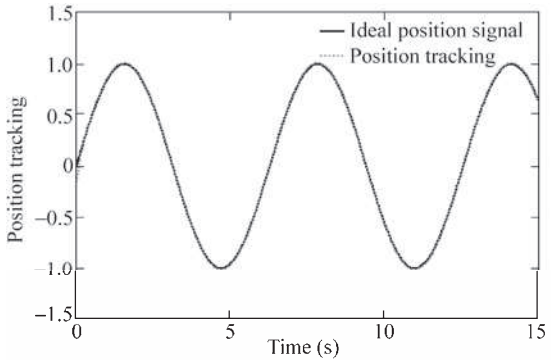


Figure 1.17 Position tracking ($M = 1$)

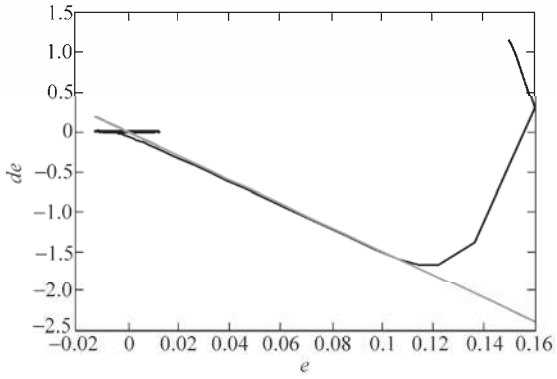


Figure 1.18 Phase trajectory ($M = 1$)

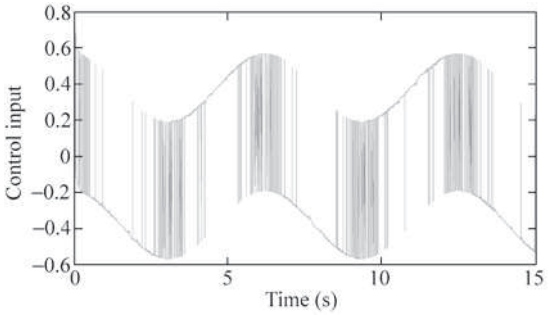


Figure 1.19 Control input ($M=1$)

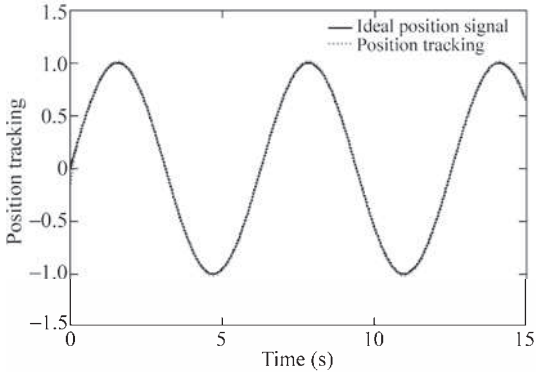


Figure 1.20 Position tracking ($M=2$)

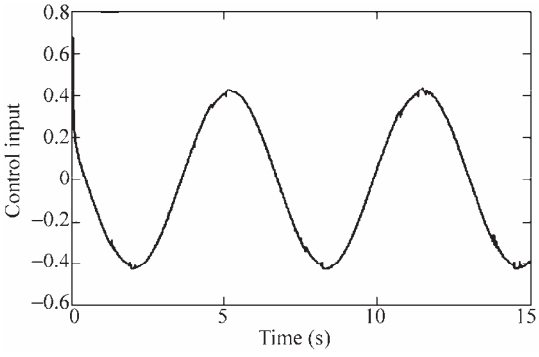


Figure 1.21 Control input ($M=2$)

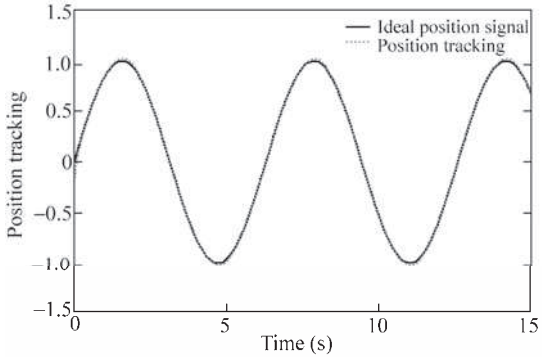


Figure 1.22 Position tracking ($M=3$)

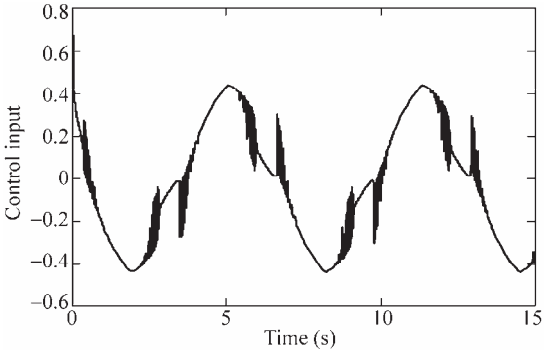
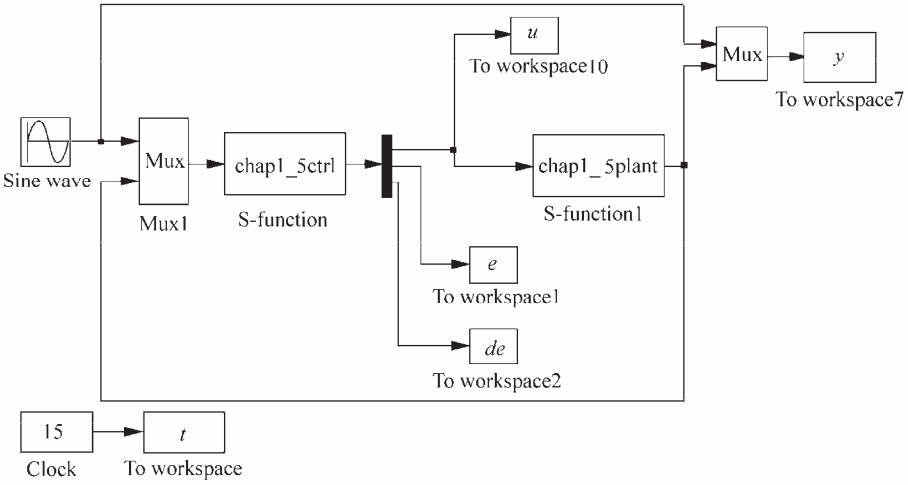


Figure 1.23 Control input ($M=3$)

Simulation programs:

(1) Simulink main program: chap1_5sim.mdl



(2) Controller: chap1_5ctrl.m

```
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)
switch flag,
case 0,
   [sys,x0,str,ts]=mdlInitializeSizes;
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    = 3;
sizes.NumInputs     = 3;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys = simsizes(sizes);
x0  = [];
str = [];
ts  = [];
function sys=mdlOutputs(t,x,u)
r=u(1);
dr=cos(t);
ddr=-sin(t);

th=u(2);
dth=u(3);

c=15;
e=r-th;
de=dr-dth;
s=c*e+de;

D=50;
xite=1.50;

fx=25*dth;
b=133;

M=2;
if M==1          %Switch function
  ut=1/b*(c*(dr-dth)+ddr+fx+(D+xite)*sign(s));
elseif M==2      %Saturated function
  fai=0.20;
  if abs(s)<=fai
```

```
      sat=s/fai;
   else
      sat=sign(s);
   end
   ut=1/b*(c*(dr-dth)+ddr+fx+(D+xite)*sat);
elseif M==3      %Relay function
   delta=0.015;
   rs=s/(abs(s)+delta);
   ut=1/b*(c*(dr-dth)+ddr+fx+(D+xite)*rs);
end

sys(1)=ut;
sys(2)=e;
sys(3)=de;
```

(3) Plant: chap1_5plant.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=[-0.15 -0.15];
str=[];
ts=[];
function sys=mdlDerivatives(t,x,u)
dt=50*sin(t);
sys(1)=x(2);
sys(2)=-25*x(2)+133*u+dt;
function sys=mdlOutputs(t,x,u)
sys(1)=x(1);
sys(2)=x(2);
```

(4) Plot program: chap1_5plot.m

```
close all;

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

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

c=15;
figure(3);
plot(e,de,'k',e,-c'.*e,'r','linewidth',2);
xlabel('e');ylabel('de');
```

1.6 Sliding Mode Control Based on the Equivalent Control

In the sliding mode controller, the control law usually consists of the equivalent control u_{eq} and the switching control u_{sw}. The equivalent control keeps the state of system on the sliding surface, while the switching control forces the system sliding on the sliding surface.

1.6.1 System Description

A n-order SISO nonlinear system can be described as

$$x^{(n)} = f(x,t) + bu(t) + d(t) \tag{1.32}$$

$$\boldsymbol{x} = [x \quad \dot{x} \quad \cdots \quad x^{(n-1)}]^{\mathrm{T}} \tag{1.33}$$

where $b > 0$, $x \in \mathbf{R}^n$, $u \in \mathbf{R}$, $d(t)$ denotes external disturbance and uncertainty while we assume $|d(t)| \leqslant D$.

1.6.2 Sliding Mode Controller Design

1.6.2.1 Equivalent Controller Design

Ignoring external disturbance and uncertainty, the plant can be described as

$$x^{(n)} = f(x,t) + bu(t) \tag{1.34}$$

The tracking error vector is

$$\boldsymbol{e} = \boldsymbol{x}_{\mathrm{d}} - \boldsymbol{x} = [e \quad \dot{e} \quad \cdots \quad e^{(n-1)}]^{\mathrm{T}} \tag{1.35}$$

Then switch function is

$$s(x,t) = \boldsymbol{C}\boldsymbol{e} = c_1 e + c_2 \dot{e} + \cdots + e^{(n-1)} \tag{1.36}$$

where $\boldsymbol{C} = [c_1 \quad c_2 \quad \cdots \quad c_{n-1} \quad 1]$ is a $1 \times n$ vector.
Choose $\dot{s} = 0$, we get

$$\begin{aligned} \dot{s}(x,t) &= c_1 \dot{e} + c_2 \ddot{e} + \cdots + e^{(n)} = c_1 \dot{e} + c_2 \ddot{e} + \cdots + c_{n-1} e^{(n-1)} + x_{\mathrm{d}}^{(n)} - x^{(n)} \\ &= \sum_{i=1}^{n-1} c_i e^{(i)} + x_{\mathrm{d}}^{(n)} - f(x,t) - bu(t) = 0 \end{aligned} \tag{1.37}$$

The control law is designed as

$$u_{\mathrm{eq}} = \frac{1}{b}\left(\sum_{i=1}^{n-1} c_i e^{(i)} + x_{\mathrm{d}}^{(n)} - f(x,t) \right) \tag{1.38}$$

1.6.2.2 Sliding Mode Controller Design

In order to satisfy reaching conditions of sliding mode control $s(x,t) \cdot \dot{s}(x,t) \leqslant -\eta \,|\, s \,|,\ \eta > 0$, we must choose switching control whose control law is

$$u_{\mathrm{sw}} = \frac{1}{b} K \operatorname{sgn}(s) \tag{1.39}$$

where $K = D + \eta$.

The sliding mode controller include the equivalent control and the switching control, then we have

$$u = u_{\mathrm{eq}} + u_{\mathrm{sw}} \tag{1.40}$$

Stability proof:

$$\dot{s}(x,t) = \sum_{i=1}^{n-1} c_i e^{(i)} + x_{\mathrm{d}}^{(n)} - f(x,t) - bu(t) - d(t) \tag{1.41}$$

Submitting Eqs. (1.40) – (1.41), we can get:

$$\begin{aligned} \dot{s}(x,t) &= \sum_{i=1}^{n-1} c_i e^{(i)} + x_{\mathrm{d}}^{(n)} - f(x,t) - b\left(\frac{1}{b}\left(\sum_{i=1}^{n-1} c_i e^{(i)} + x_{\mathrm{d}}^{(n)} - f(x,t) \right) + \frac{1}{b} K \operatorname{sgn}(s) \right) - d(t) \\ &= -K \operatorname{sgn}(s) - d(t) \end{aligned}$$

Therefore, we have

$$s\dot{s} = s(-K\,\mathrm{sgn}(s)) - s \cdot d(t) = -\eta\,|\,s\,| \leqslant 0 \tag{1.42}$$

1.6.3 Simulation Example

We choose a plant as follows:

$$\ddot{x} = -25\dot{x} + 133u(t) + d(t)$$

Therefore $f(x,t) = -25\dot{x}$, $b = 133$.

Let $d(t) = 50\sin(t)$, $\eta = 0.10$, ideal position signal is $r = \sin(2\pi t)$, choose $c = 25$, then we can get $D = 50$. Adapting controller (1.40), the simulation results are shown in Fig. 1.24 and Fig. 1.25.

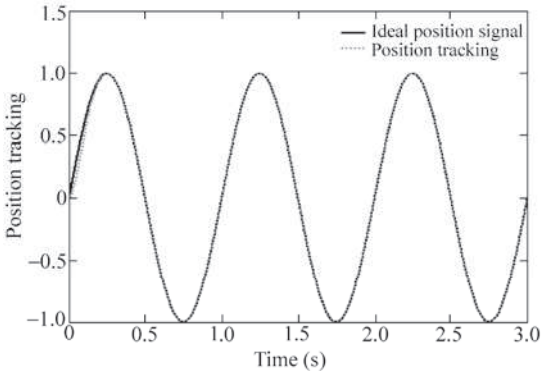


Figure 1.24 Position tracking

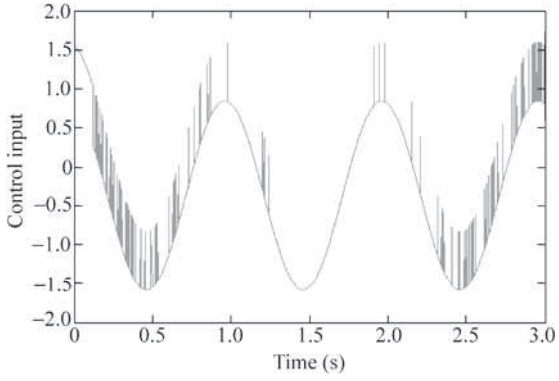
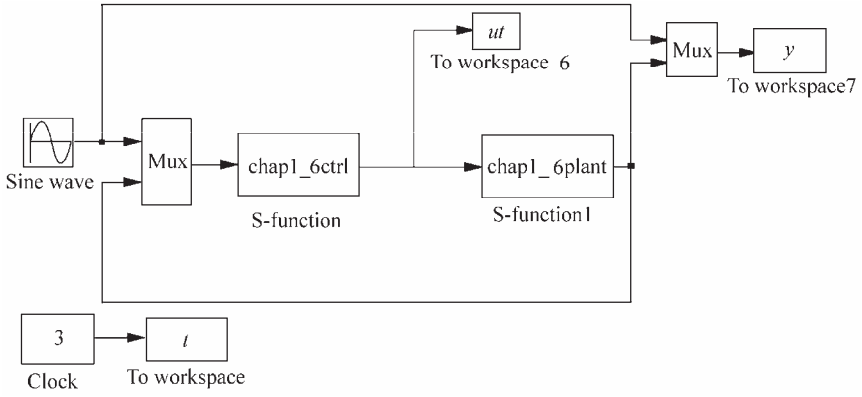


Figure 1.25 Control input

Simulation programs:

(1) Simulink main program: chap1_6sim.mdl



(2) Controller: chap1_6ctrl.m

```
function [sys,x0,str,ts]=s_function(t,x,u,flag)
switch flag,
case 0,
   [sys,x0,str,ts]=mdlInitializeSizes;
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     = 1;
sizes.NumInputs      = 3;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[];
str=[];
ts=[];
function sys=mdlOutputs(t,x,u)
r=u(1);
dr=2*pi*cos(2*pi*t);
ddr=-(2*pi)^2*sin(2*pi*t);
x=u(2);dx=u(3);
e=r-x;
de=dr-dx;
```

```
c=25;
s=c*e+de;

f=-25*dx;
b=133;

ueq=1/b*(c*de+ddr-f);
D=50;
xite=0.10;
K=D+xite;
usw=1/b*K*sign(s);

ut=ueq+usw;

sys(1)=ut;
```

(3) Plant: chap1_6plant.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=[0,0];
str=[];
ts=[];
function sys=mdlDerivatives(t,x,u)
dt=50*sin(t);
sys(1)=x(2);
sys(2)=-25*x(2)+133*u+dt;
function sys=mdlOutputs(t,x,u)
sys(1)=x(1);
sys(2)=x(2);
```

(4) Plot program: chap1_6plot.m

```
close all;

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

figure(2);
plot(t,ut(:,1),'r','linewidth',2);
xlabel('time(s)');ylabel('Control input');
```

1.7 Digital Simulation of Sliding Mode Control

1.7.1 Basic Theory

In practical engineering we often use digital control. The digital control system structure is shown in Fig. 1.26, and the corresponding program diagram of the system is shown in Fig. 1.27.

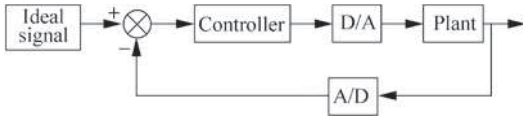


Figure 1.26 Digital control system structure

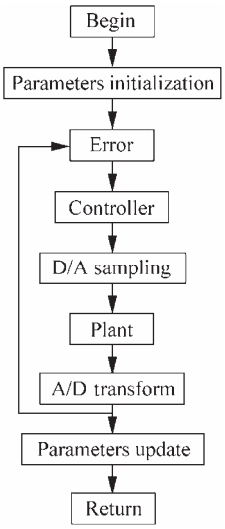


Figure 1.27 Program diagram of digital control algorithm

1.7.2 Simulation Example

We choose a plant as follow:

$$\ddot{x} = -25\dot{x} + 133u(t) + d(t)$$

Therefore $f(x,t) = -25\dot{x}$, $b = 133$. We choose sampling time $T = 0.001$. Let $d(t) = 3\sin(t)$, then $\eta = 3.1$. Choose ideal position signal as $r = \sin(t)$, and $c = 5$. Adapting controller Eq. (1.28) (in program $M = 1$), the simulation results are shown in Fig. 1.28 – Fig. 1.30. Moreover, using saturation function (1.29) instead of switch function (in program $M = 2$), the simulation results are shown in Fig. 1.31 – Fig. 1.33.

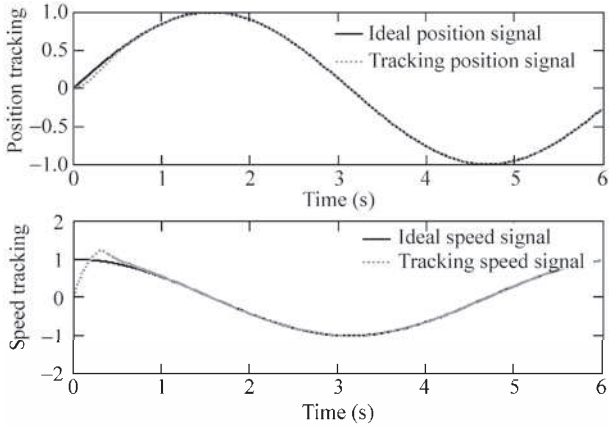


Figure 1.28 Position tracking

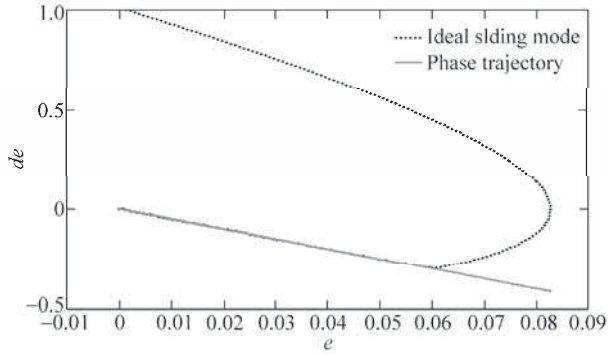


Figure 1.29 Phase trajectory

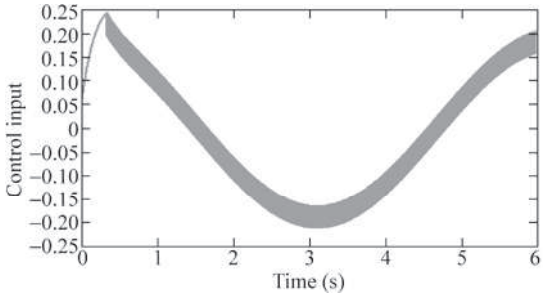


Figure 1.30 Control input

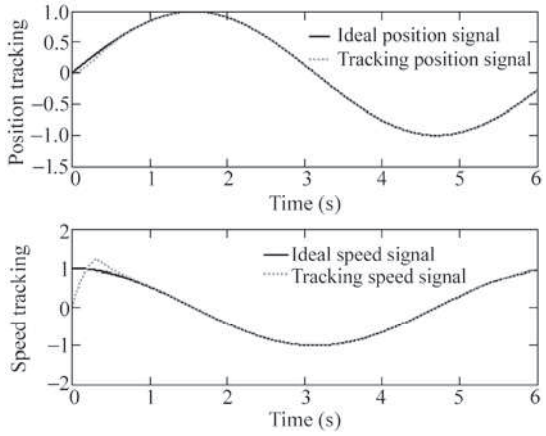


Figure 1.31 Position tracking

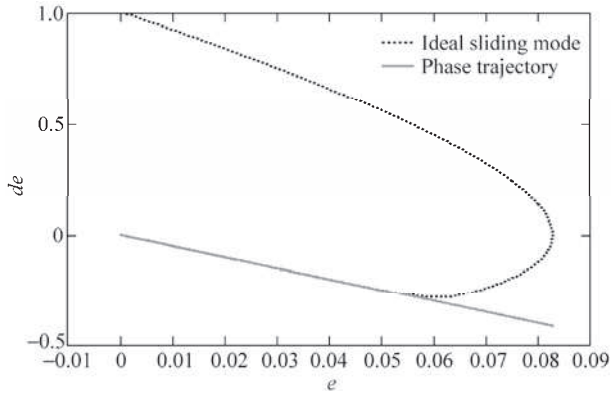


Figure 1.32 Phase trajectory

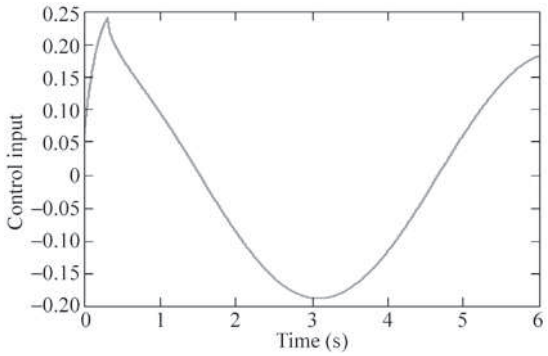


Figure 1.33 Control input

Simulation programs:

(1) Main program: chap1_7.m

```
clear all;
close all;
a=25;b=133;
xk=zeros(2,1);
ut_1=0;
c=5;
T=0.001;
for k=1:1:6000
time(k)=k*T;
thd(k)=sin(k*T);
dthd(k)=cos(k*T);
ddthd(k)=-sin(k*T);

tSpan=[0 T];

para=ut_1;      % D/A
[tt,xx]=ode45('chap1_7plant',tSpan,xk,[],para);
xk=xx(length(xx),:);    % A/D
th(k)=xk(1);
dth(k)=xk(2);

e(k)=thd(k)-th(k);
de(k)=dthd(k)-dth(k);
s(k)=c*e(k)+de(k);

xite=3.1;  % xite>max(dt)
M=1;
if M==1
   ut(k)=1/b*(a*dth(k)+ddthd(k)+c*de(k)+xite*sign(s(k)));
elseif M==2                %Saturated function
       delta=0.05;
        kk=1/delta;
```

```
        if abs(s(k))>delta
          sats=sign(s(k));
        else
          sats=kk*s(k);
        end
    ut(k)=1/b*(a*dth(k)+ddthd(k)+c*de(k)+xite*sats);
  end
  ut_1=ut(k);
  end
  figure(1);
  subplot(211);
  plot(time,thd,'k',time,th,'r:','linewidth',2);
  xlabel('time(s)');ylabel('position tracking');
  legend('ideal position signal','tracking position signal');
  subplot(212);
  plot(time,dthd,'k',time,dth,'r:','linewidth',2);
  xlabel('time(s)');ylabel('speed tracking');
  legend('ideal speed signal','tracking speed signal');
  figure(2);
  plot(thd-th,dthd-dth,'k:',thd-th,-c*(thd-th),'r','linewidth',2);
     %Draw line(s=0)
  xlabel('e');ylabel('de');
  legend('ideal sliding mode','phase trajectory');
  figure(3);
  plot(time,ut,'r','linewidth',2);
  xlabel('time(s)');ylabel('Control input');
```

(2) Plant program: chap1_7plant.m

```
function dx=Plant(t,x,flag,para)
dx=zeros(2,1);
a=25;b=133;
ut=para(1);
dt=3.0*sin(t);
dx(1)=x(2);
dx(2)=-a*x(2)+b*ut+dt;
```

References

[1] Itkis U. Control System of Variable Structure. New York: Wiley, 1976

[2] Hung JY, Gao W, Hung JC. Variable Structure Control: A Survey, IEEE Transaction on Industrial Electronics, 1993,40(1): 2 – 22

[3] Edwards C, Spurgeon S. Sliding Mode Control: Theory and Applications, London: Taylor and Francis, 1998