基于 Matlab 和 Truetime 的网络控制系统仿真

电子信息与电气工程学院自动化系　F1703203　517021910825　岳博

一、实验目的

通过动手实践了解网络控制系统的架构及运行控制原理
熟悉使用基于 MATLAB 平台、Simulink 仿真工具和 TrueTime 工具箱的网络控制系统的仿真方法
考察不同丢包率下网络控制系统状态、输出和控制曲线的影响
研究丢包率对无线网络传输的网络控制系统的稳定性的影响

二、仿真平台构建

Truetime 是瑞典隆德（Lund）大学自动化系 MartinOhin，Dan Henriksson 和 Anton Cervin 于 2002 年推出的基于 MATLAB/Simulink 的网络控制系统仿真工具箱。针对每一特定的网络协议，该工具箱可以实现控制系统与实时调度的综合仿真研究，是目前网络控制系统理想的虚拟仿真工具之一。

本次实验使用 Truetime2.0 工具箱的七大模块中的 TrueTime Kernel 模块和 TrueTime Wireless Network 模块。其中，系统的传感器、执行器和控制器由 TrueTime Kernel 表示，无线网络由 TrueTime Wireless Network 表示。传感器采用时钟驱动的方式进行周期性采样，控制器和执行器采用事件驱动方式。

在确定各个功能模块之后，需要对各模块进行初始化，即初始化模块内核，定义消息函数并设置消息调度策略和初始化网络端口并设置节点对应的网络端口代号。

三、实验步骤

对于本次仿真实验，按照 Truetime 仿真流程，我将实验步骤分为三步：建立数学模型，功能模块图仿真和编写相关代码。

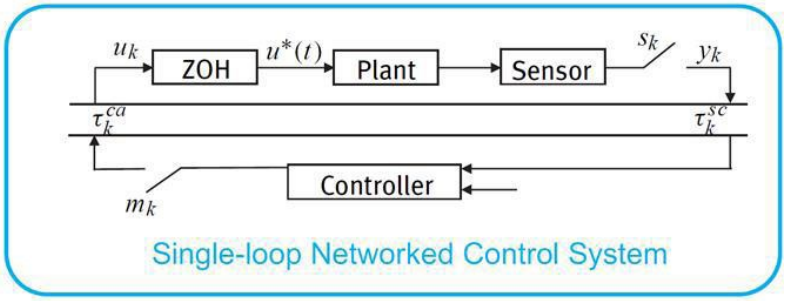
3.1 建立数学模型

这一部分简要重述该网络控制系统的数学模型。
已知直流电机的传递函数为：

$$G(s)=\frac{1000}{s^2+s}$$

该电机通过网络 IEEE 802.11b/g（WLAN）的方式进行远程控制，系统的控制结构如下图所示：

图 1 单闭环网络控制系统流程图



假设传感器采用时钟驱动的方式进行周期性采样，控制器和执行器采用事件驱动方式。存在网络丢包现象，其状态空间模型可以描述成

$$\begin{cases} \dot{x}(t) = \begin{bmatrix} 0 & 1 \\ 0 & -1 \end{bmatrix} x(t) + (1-p)\begin{bmatrix} 0 \\ 1000 \end{bmatrix} u(t) + p\begin{bmatrix} 0 \\ 1000 \end{bmatrix} u(t-1) \\ y = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} x(t) \end{cases}$$

其中初始状态为 x(0)= $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$，p 为丢包概率，p=0 表示无数据丢包，p=1 表示有数据丢包，假设不会发生连续丢包。给定控制器为$u(t) = Kx(t) = [-0.005 \ -0.005]$。

3.2 功能模块图仿真

这一部分主要介绍基于 Simulink 及 Truetime 工具箱的模块图绘制。通过理解 wireless 和 network 两个示例文件，搭建本次实验的网络控制系统的功能模块图，如图 2 所示。

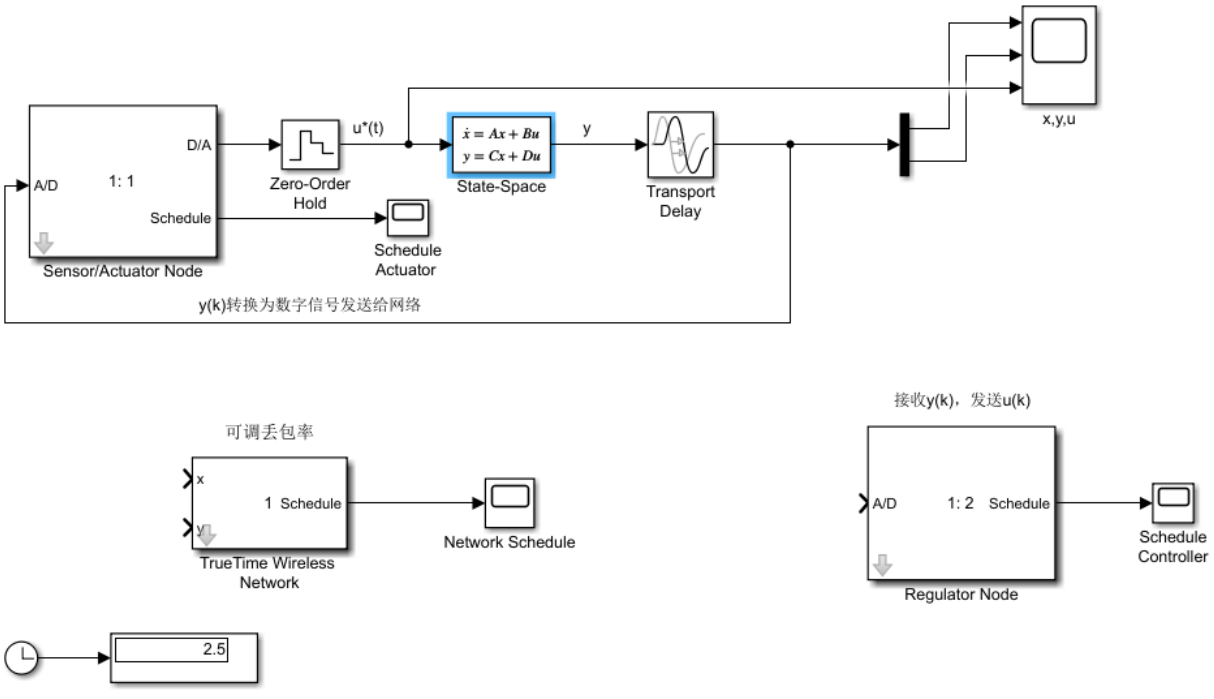


图 2 仿真系统功能模块图

其中，TrueTime Wireless Network 模块模拟了网络部分，可以更改丢包率（Loss Probability）这一参数。在 Regulator Node 中的 TrueTime Kernel 中，模拟了控制器从网络中接收 y(k)，并生成 u(k)，将 u(k)发送到网络中的过程。在 Sensor/Actuator Node 的 TrueTime Kernel 中，模拟了传感器到控制器再到执行器节点的过程，执行器从网络中接收控制信号 u(k)；进行零阶保持；若丢包贼改成 u(k-1)，若不丢包则改成 u(k)；再通过状态转移及传输延迟生成输出信号 y(k)；再讲数字信号 y(k)发送至网络。如此形成闭环网络控制系统。仿真过程可参考图 3。

在搭建的过程中，依照数学模型的参数对部分模块的参数设置做出调整。

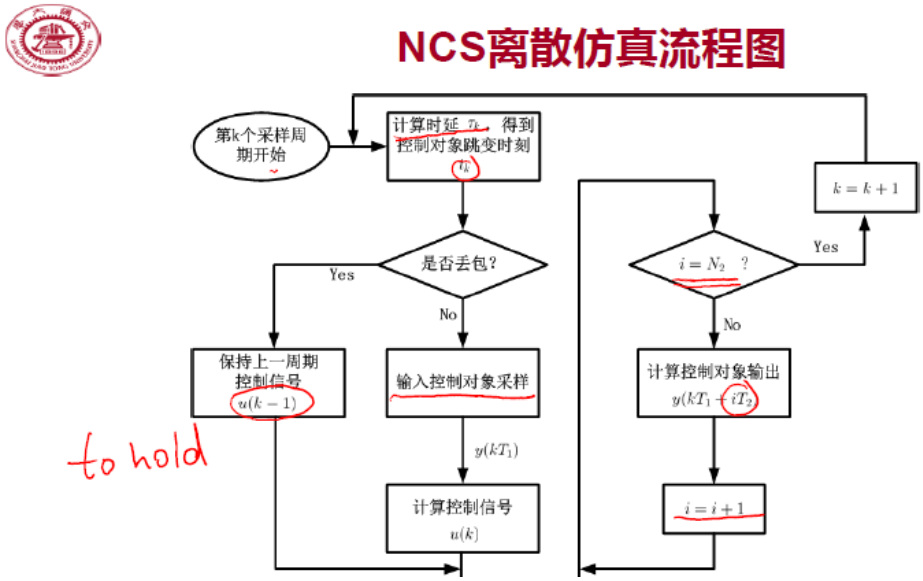


图 3 NCS 离散仿真流程图

3.3 编写相关代码

通过研究 TrueTime 2.0-reference manual 及阅读 wireless 文件夹中的相关代码后，可将代码编写分为两个步骤依次进行，模块初始化和编写任务代码。模块初始化针对控制器（Regulator）和执行器/传感器（Sensor/Actuator），任务代码仅针对控制节点（Regulator Node）。

3.3.1 模块初始化

对控制器（Regulator）和执行器/传感器（Sensor/Actuator）的初始化可直接参照 wireless 中的 regulator.m 文件和 actuator_init.m 文件进行，无需改动。代码如下：

```
function regulator_init

% Distributed control system: regulator node
%
% Receives messages from the sensor node, computes control signal
% and sends it back to the actuator node.

% Initialize TrueTime kernel
ttInitKernel('prioFP'); % fixed priority
ttSetKernelParameter('energyconsumption', 0.010); % 10 mW
```

```
11.
12. % Create mailboxes
13. ttCreateMailbox('sensor_signal', 10)
14. ttCreateMailbox('power_ping', 10)
15. ttCreateMailbox('power_response', 10)
16.
17. % Controller parameters
18. h = 0.010;
19. N = 100000;
20. Td = 0.035;
21. K = 1.5;
22.
23. % Create task data (local memory)
24. data.u = 0.0;
25. data.K = K;
26. data.ad = Td/(N*h+Td);
27. data.bd = N*K*Td/(N*h+Td);
28. data.Dold = 0.0;
29. data.yold = 0.0;
30.
31. % Create controller task
32. deadline = h;
33. prio = 1;
34. ttCreateTask('pid_task', deadline, 'ctrlcode', data);
35. ttSetPriority(prio, 'pid_task');
36.
37. % Create power controller task
38. offset = 2;
39. period = 0.025;
40. prio = 2;
41. power_data.transmitPower = 20;
42. power_data.name = 2;     % We are node number 2 in the network
43. power_data.receiver = 1; % We are communicating with node 1
44. power_data.haverun = 0;  % We have not run yet
45. ttCreatePeriodicTask('power_controller_task', offset, period, 'powctrlcode', power_data);
46. ttSetPriority(prio, 'power_controller_task');
47.
48. % Create power response task
49. deadline = 100;
50. prio = 3;
51. ttCreateTask('power_response_task', deadline, 'powrespcode');
52. ttSetPriority(prio, 'power_response_task');
53.
54. % Initialize network
55. ttCreateHandler('nw_handler', 1, 'msgRcvCtrl');
```

```
ttAttachNetworkHandler('nw_handler');
```

```
function actuator_init

% Distributed control system: actuator node
%
% Receives messages from the controller and actuates
% the plant.

% Initialize TrueTime kernel
ttInitKernel('prioFP'); % fixed priority
ttSetKernelParameter('energyconsumption', 0.010); % 10 mW

% Create mailboxes
ttCreateMailbox('control_signal', 10)
ttCreateMailbox('power_ping', 10)
ttCreateMailbox('power_response', 10)

% Create sensor task
data.y = 0;
offset = 0.0;
period = 0.010;
prio = 1;
ttCreatePeriodicTask('sens_task', offset, period, 'senscode', data);
ttSetPriority(prio, 'sens_task');

% Create actuator task
deadline = 100;
prio = 2;
ttCreateTask('act_task', deadline, 'actcode');
ttSetPriority(prio, 'act_task');

% Create power controller task
offset = 2.07;
period = 0.025;
prio = 3;
power_data.transmitPower = 20;
power_data.name = 1;     % We are node number 1 in the network
power_data.receiver = 2; % We are communicating with node 2
power_data.haverun = 0;  % We have not run yet
ttCreatePeriodicTask('power_controller_task', offset, period, 'powctrlcode', power_data);
ttSetPriority(prio, 'power_controller_task');

```

```
% Create power response task
deadline = 100;
prio = 4;
ttCreateTask('power_response_task', deadline, 'powrespcode');
ttSetPriority(prio, 'power_response_task');

% Initialize network
ttCreateHandler('nw_handler', 1, 'msgRcvActuator');
ttAttachNetworkHandler('nw_handler');
```

3.3.2 编写任务代码

```
function [exectime, data] = ctrlcode(seg, data)

switch seg
 case 1
  % Read all buffered packets
  temp = ttTryFetch('sensor_signal');
  while ~isempty(temp)
    y = temp;
    temp = ttTryFetch('sensor_signal');
  end
  if isempty(y)
    data.u = data.old;  % 丢包，保持 u(k-1)
  else
    data.u = -0.005 * (y(1) + y(2));  % 不丢包，保持 u(k)
  end
  exectime = 0.0005;

 case 2
  msg.msg = data.u;
  msg.type = 'control_signal';
  ttSendMsg(1, msg, 80);    % Send 80 bits to node 1 (actuator)
  exectime = -1; % finished
end
```

四、实验结果分析

4.1 不同丢包率下三条结果曲线

当丢包率分别为 0.3 和 0.6 时系统状态曲线、输出曲线和控制曲线如图 4 和图 5 所示。显然，在丢包率为 0.3 和 0.6 时，系统是稳定的。

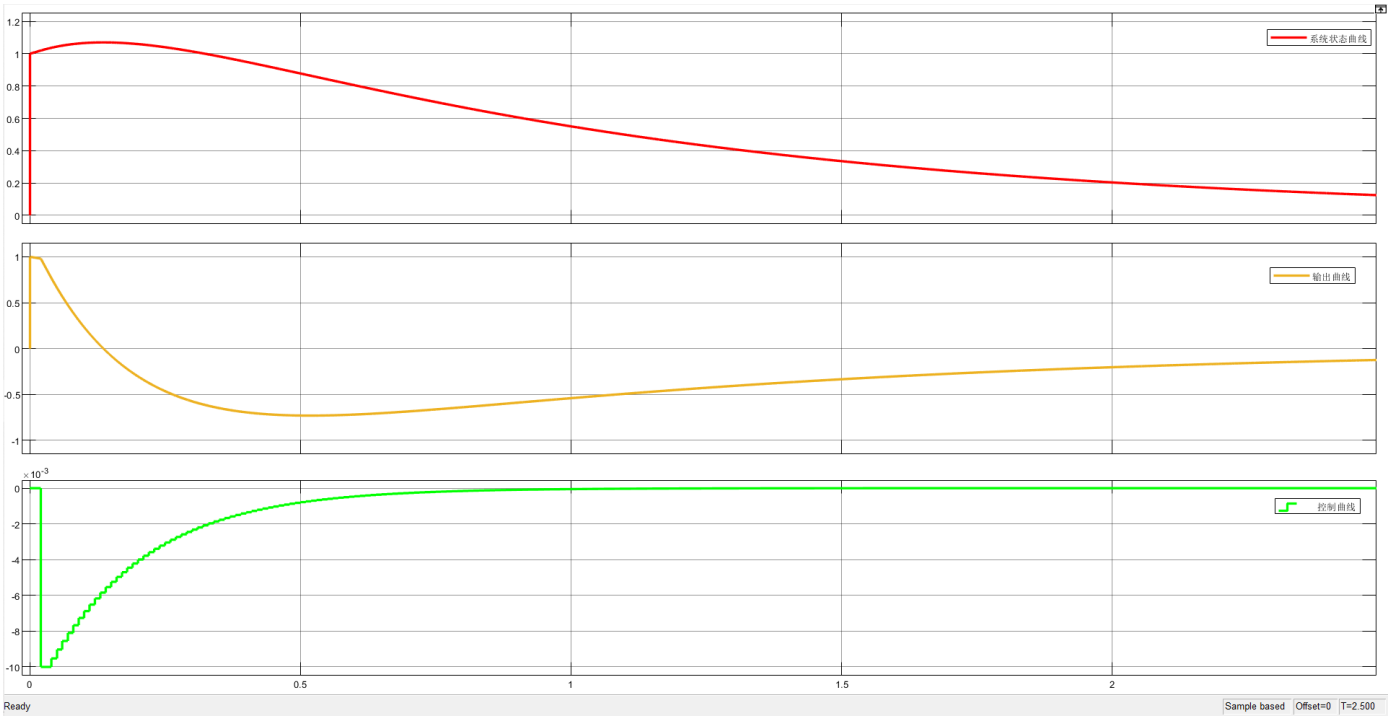


图 4 丢包率为 0.3 时，系统状态曲线、输出曲线和控制曲线

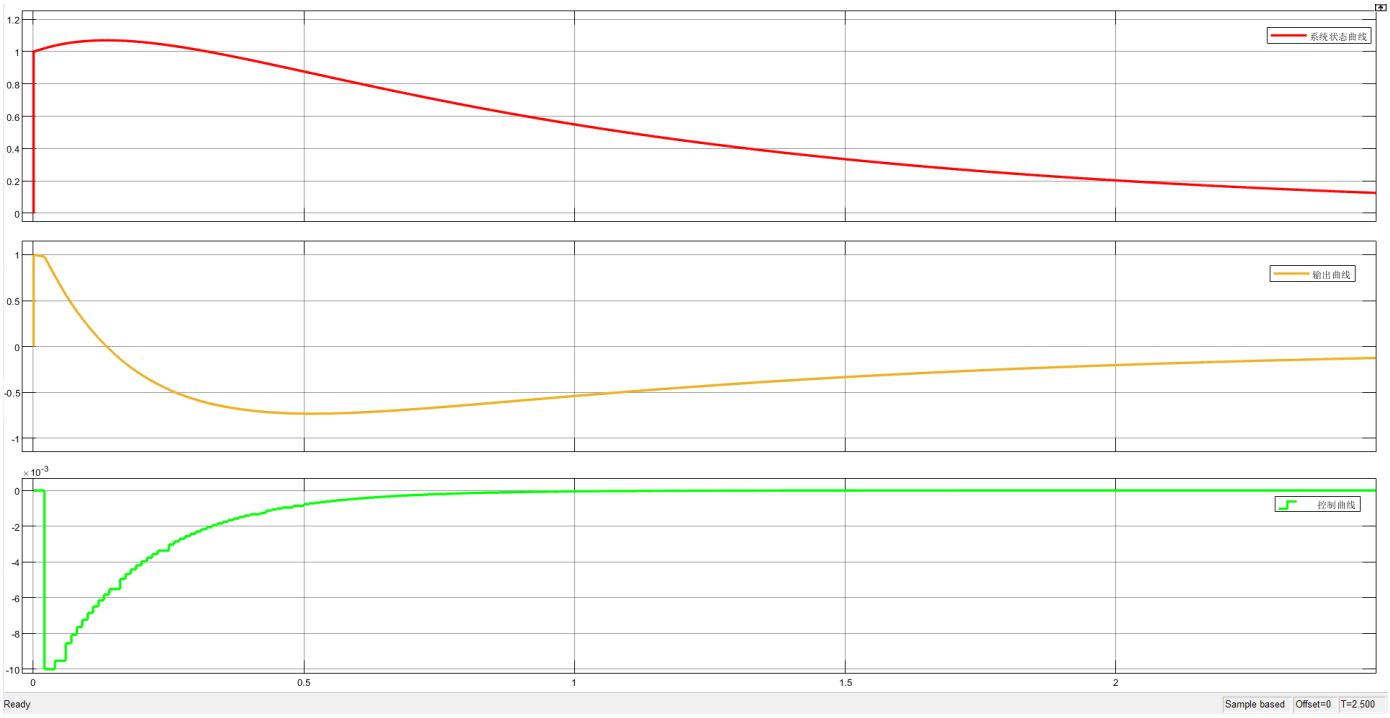


图 5 丢包率为 0.6 时，系统状态曲线、输出曲线和控制曲线

4.2 不同丢包率下稳定性分析

根据上课讲解的 Bernoulli 过程的线性跳变系统，建立新的系统状态变量$\xi_k = \begin{bmatrix} x_k \\ u_{k-1} \end{bmatrix}$。

由是否丢包情况下两种不同的控制率，即：

$$u_k = \begin{cases} -\bar{K}x_k & ,p = 0 \\ -\bar{K}x_{k-1} & ,p = 1 \end{cases}$$

结合系统离散状态方程：

$$x_{k+1} = e^{Ah}x_k + \int_0^h e^{As}dsBu_k$$

其中，$A = \begin{bmatrix} 0 & 1 \\ 0 & -1 \end{bmatrix}$，$B = \begin{bmatrix} 0 \\ 1000 \end{bmatrix}$，采样周期为 0.01s。

将u_k表达式代入x_{k+1}，可得

$$\xi_{k+1} = \begin{bmatrix} x_{k+1} \\ u_k \end{bmatrix} = \begin{cases} \begin{bmatrix} (e^{Ah} - \int_0^h e^{As}dsB\bar{K}) & 0 \\ -\bar{K} & 0 \end{bmatrix} \begin{bmatrix} x_k \\ u_{k-1} \end{bmatrix}, p = 0 \\ \begin{bmatrix} e^{Ah} & \int_0^h e^{As}dsB) \\ 0 & I \end{bmatrix} \begin{bmatrix} x_k \\ u_{k-1} \end{bmatrix} \quad ,p = 1 \end{cases}$$

令$A_0 = \begin{bmatrix} (e^{Ah} - \int_0^h e^{As}dsB\bar{K}) & 0 \\ -\bar{K} & 0 \end{bmatrix}, A_1 = \begin{bmatrix} e^{Ah} & \int_0^h e^{As}dsB) \\ 0 & I \end{bmatrix}$，由 Lyapunov 稳定性判据可知，若$\exists P > 0$，使得$(1-p){A_0}^T P A_0 + p{A_1}^T P A_1 - P < 0$，则系统是均方稳定的。

运用 LMI 工具箱对 P>0 进行求解，得到当丢包率大于等于 0.96 时，系统不满足均方稳定（MSS），当丢包率小于等于 0.95 时，系统是均方稳定的。

```
h=0.01;
eAh=[1 1-exp(-h);0 exp(-h)];
A01=eAh-[h h+exp(-h)-1;0 1-exp(-h)]*[0;1000]*[0.005 0.005];
A0=[A01(1),A01(3),0;A01(2),A01(4),0;-0.005,-0.005,0];
tmp=[h h+exp(-h)-1;0 1-exp(-h)]*[0;1000];
A1=[eAh(1),eAh(3),tmp(1);eAh(2),eAh(4),tmp(2);0,0,1];
c=zeros(1,120);
i=1;
for p=0:0.01:1
setlmis([]);
P=lmivar(1,[3,1]);
lmiterm([-1,1,1,P],1,1);%P>0
lmiterm([2,1,1,P],(1-p)*A0',A0);
lmiterm([2,1,1,P],p*A1',A1);
lmiterm([2,1,1,P],-1,1);
lmis=getlmis;
[tmin,xfeas]=feasp(lmis);
if tmin>0
c(i)=p;
i=i+1;
```

```
21. end
22. %P=dec2mat(lmis,xfeas,P);
23. end
24. c
```

为了观察不稳定情况下，系统状态曲线、输出曲线和控制曲线的走势，我重新设置丢包率为 0.98，这时三条曲线走势如图 6 所示。

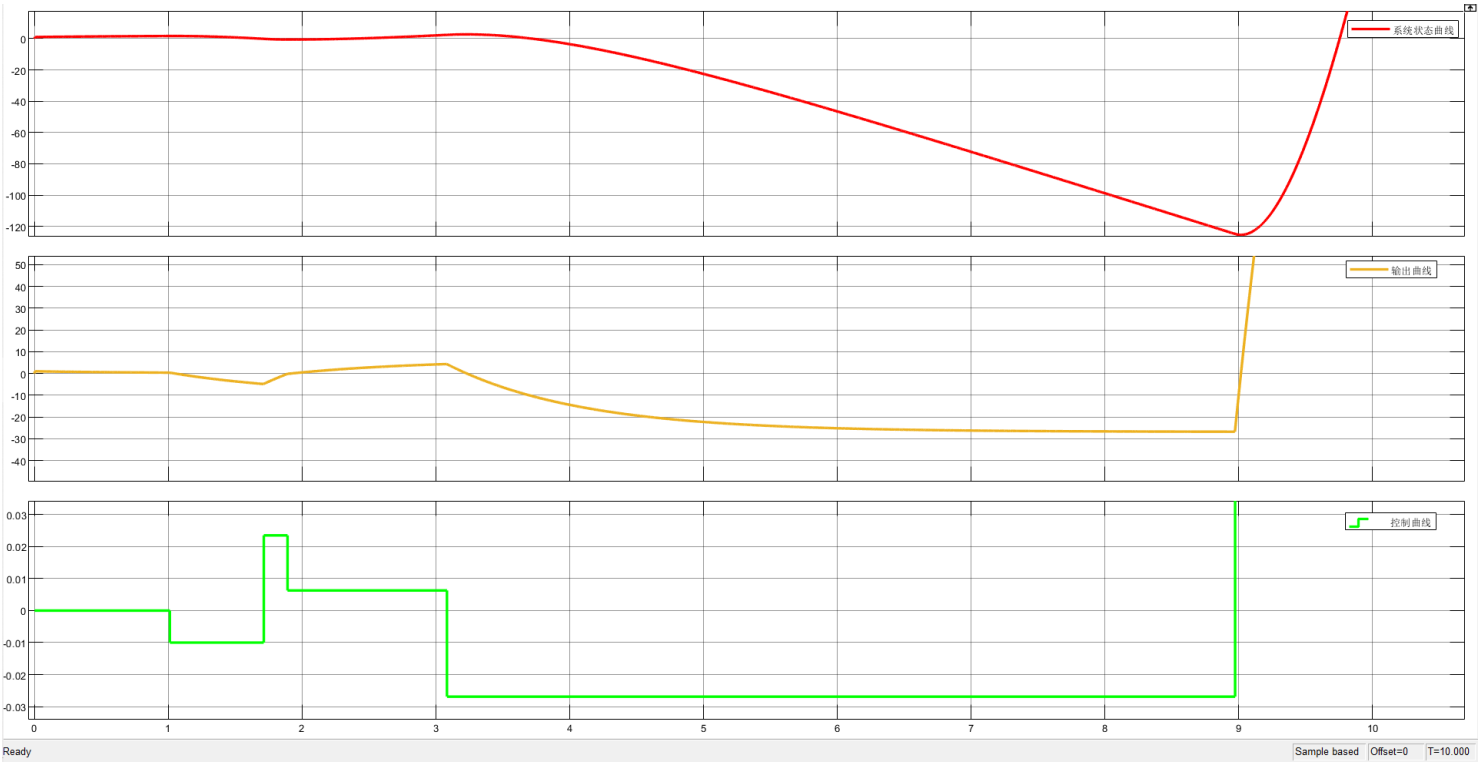


图 6 丢包率为 0.98 时，系统状态曲线、输出曲线和控制曲线

致谢

由衷感谢认真负责的杨博老师和陈彩莲老师对本次实验前期理论知识的细致讲解，对实验的耐心介绍和答疑，也感谢张景龙助教对配置仿真环境及实验过程中种种问题的耐心解答！