基于 Matlab 和 Truetime 的网络控制系统仿真

姓名：谢敬鱼　学号：516021910125

一. 实验目的与要求

1.1 实验目的

1. 了解网络控制系统的组成和运行以及控制原理
2. 熟悉利用 MATLAB 和 TrueTime 进行网络控制系统的仿真的方法
3. 研究丢包率对无线网络传输的网络控制系统稳定性的影响

1.2 实验要求

在本次实验中，我们尝试使用网络化控制系统（NCS）的仿真来研究混杂系统。网络化控制系统具有连续动态的控制对象，离散动态的控制器以及离散时间。NCS 仿真常用于系统分析和性能评估。网络控制系统与一般控制系统的区别有一部分在于需要考虑网络的信号和数据流，它们会受通信网络的影响，即会发生时延和丢包。在 NCS 仿真过程中，我们就需要考虑系统网络的丢包和时延对系统稳定性造成的影响。本次实验中，我们试图使用 NCS 仿真的方式模拟对直流电机进行网络控制。判断我们的控制策略在面对复杂网络环境下的丢包和时延时能否控制系统达到稳定状态，并讨论不同的丢包率情况下系统的稳定性情况。再将结果与理论计算的答案相比较。

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：

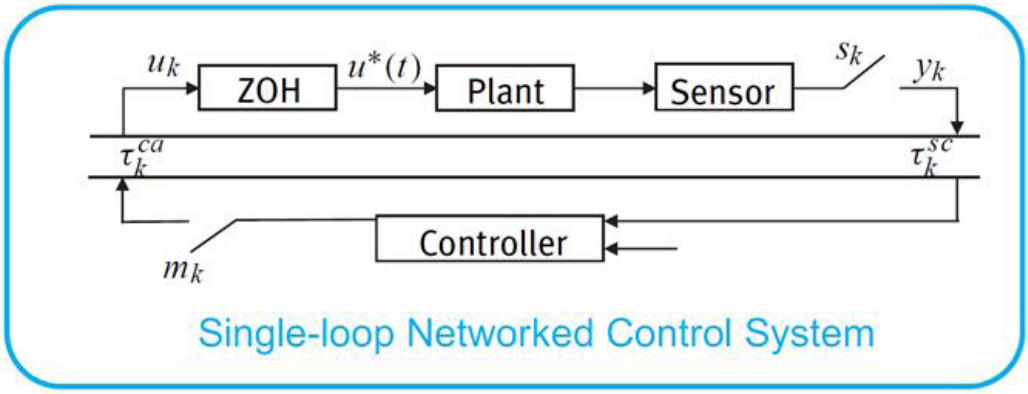


图 1 实验网络结构图

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 $u(t)=Kx(t)=[-0.005 \quad -0.005]$。

基于 Truetime Toolbox 对以上系统进行仿真，并分别给出丢包率分别为 P{p=1}=0.3，P{p=1}=0.6 时的系统状态曲线、输出曲线和控制曲线，并说明在不同丢包率下系统的稳定性。

二．仿真平台构建

TureTime 具有以下几个用途：1. 探索时间不确定性对于控制系统的影响 2. 设计能动态调整控制器的补偿机制 3. 为动态调度提供灵活的实验手段 4. 仿真事件驱动的控制系统。

Turetime 仿真软件主要包括两个基本模块：内核模块（TrueTime Kernel）和网络模块（TrueTime Network），如图 2 所示。Truetime2.0 工具箱主要包括六大模块：TrueTime Kernel（内核模块）、TrueTime Network（网络模块）、TrueTime Wireless Network（无线网络模块）、TrueTime Battery （电池模块）、TrueTimeSend（发送模块）、TrueTime Receive（接受模块）。在 Matlab 命令端输入：truetime 即可调出 Truetime2.0 工具箱。

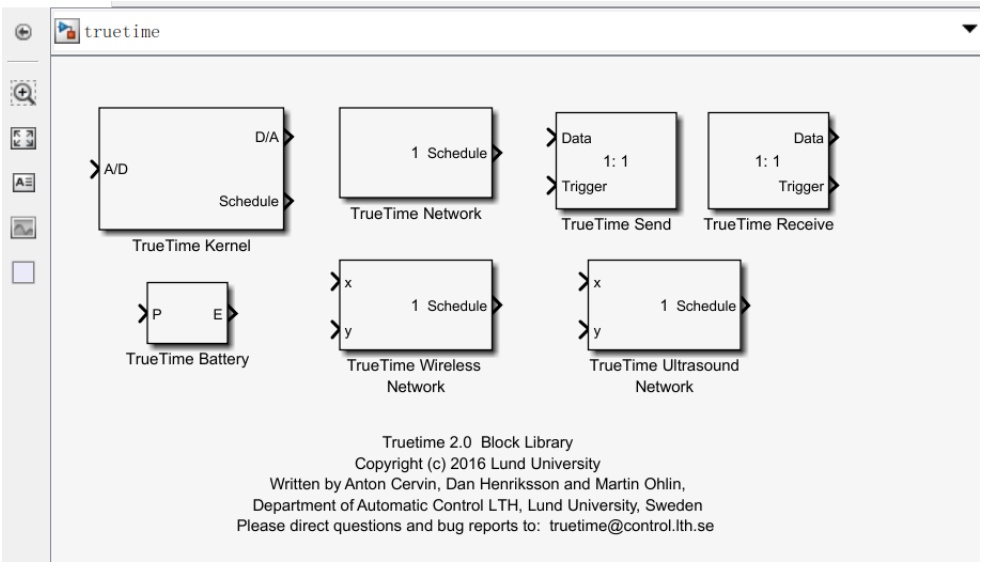


图 2 TrueTime 基本模块介绍

本实验中主要利用到的模块是：truetime-kernel、truetime-wireless network. 它用于模拟网络化控制系统的网络节点（控制器、传感器和执行器），我们可以通过编程的方式令它执行我们定义的 task

三．实验步骤

3.1 搭建实验仿真平台

实验连接图如图 3 所示：

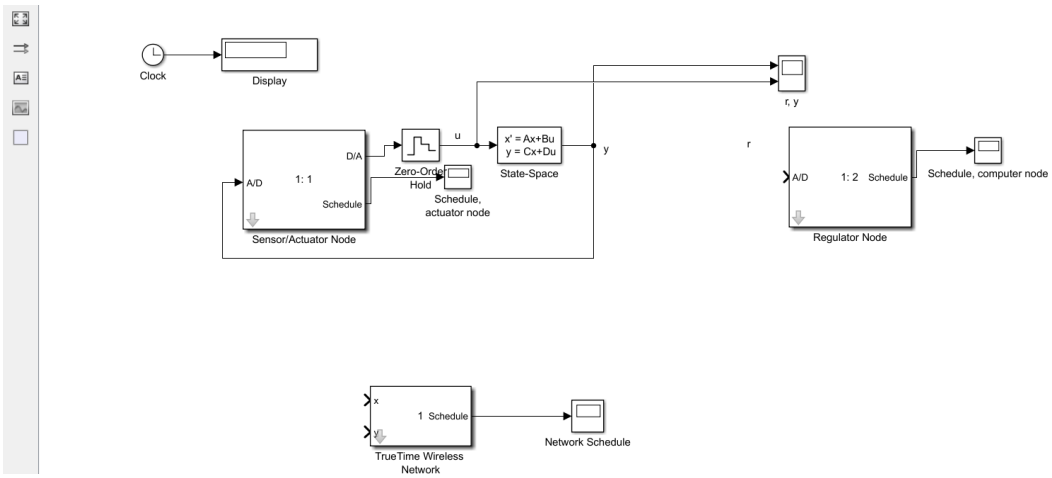


图 3 实验仿真结构图

控制结构图，从上往下主要分为 1.执行器、传感器层 2.无线网络层 3.控制器层。三层网络间的关系为：传感器在周期性采样后通过网络层向控制器传输系统的输出信号；而控制器在接收到传来的系统输出信号后进行控制信号的生成，再通过无线网络层将控制信号传送给执行器；执行器在接收到控制信号后进行直流电机的控制。

3.1.1 执行器、传感器的搭建

执行器、传感器层主要由执行器、传感器节点、零阶保持器、系统状态空间方程、显示器组成。

执行器、传感器节点有两个输入(系统状态空间的输出为 1*2 矩阵)以及一个输出（控制信号），需要将系统状态空间方程的输出连接至执行器、传感器节点的输入并将控制信号输出连接至零阶保持器的输入。零阶保持器主要用于是将离散信号转换为连续信号。

3.1.2 无线网络层的搭建

将 Wireless Network 的 Network type 改为 802.1b(WLAN)，并根据丢包率的需要设置 Loss probability 即可.

3.1.3 控制器层的搭建

在此处我设置 regulator 节点的输入数量为 1，用以接受设定的参考

输入，可用于跟随问题的实

3.2 编写对应的.m 文件

部分代码如下：

Senscode.m 文件：

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

switch seg,
 case 1,
  data.msg.msg = [ttAnalogIn(1);ttAnalogIn(2)];
  exectime = 0.0005;
 case 2,
  data.msg.type = 'sensor_signal';
  ttSendMsg(2, data.msg, 80); % Send message (80 bits) to node 2 (controller)
  exectime = 0.0004;
 case 3,
  exectime = -1; % finished
end
```

Ctrcode.m 文件：

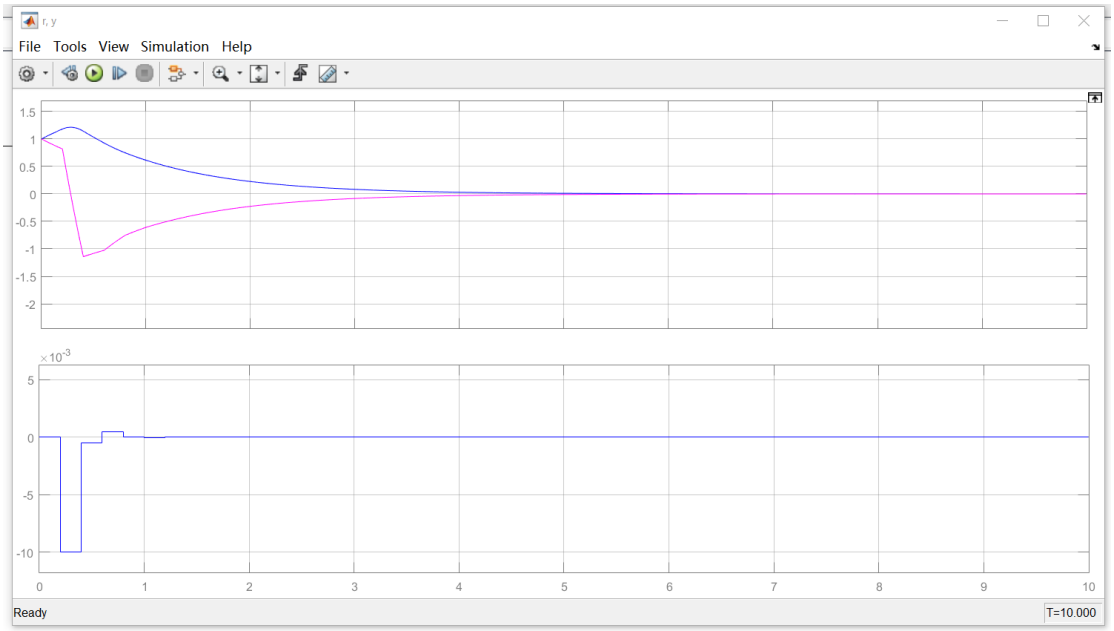
```
function [exectime, data] = ctrlcode(seg, data)

switch seg,
 case 1,
  % Read all buffered packets
  temp = ttTryFetch('sensor_signal');
  while ~isempty(temp),
    y = temp;
    temp = ttTryFetch('sensor_signal');
  end
  K = [-0.005 -0.005];
  data.u = K*y;
  exectime = 0.0005;
 case 2,
%   p=0.95;
%   x=unifrnd (0,1);
%   if(x<p)
%       data.u=data.uold;
%   end
%   data.uold=data.u;
  msg.msg = data.u;
  msg.type = 'control_signal';
  ttSendMsg(1, msg, 80);    % Send 80 bits to node 1 (actuator)
  exectime = -1; % finished
end
```

3.3 搭建好仿真平台，然后分别设丢包率为 0、0.3、0.6 进行实验仿真，观察系统的状态曲线、输出曲线和控制曲线。

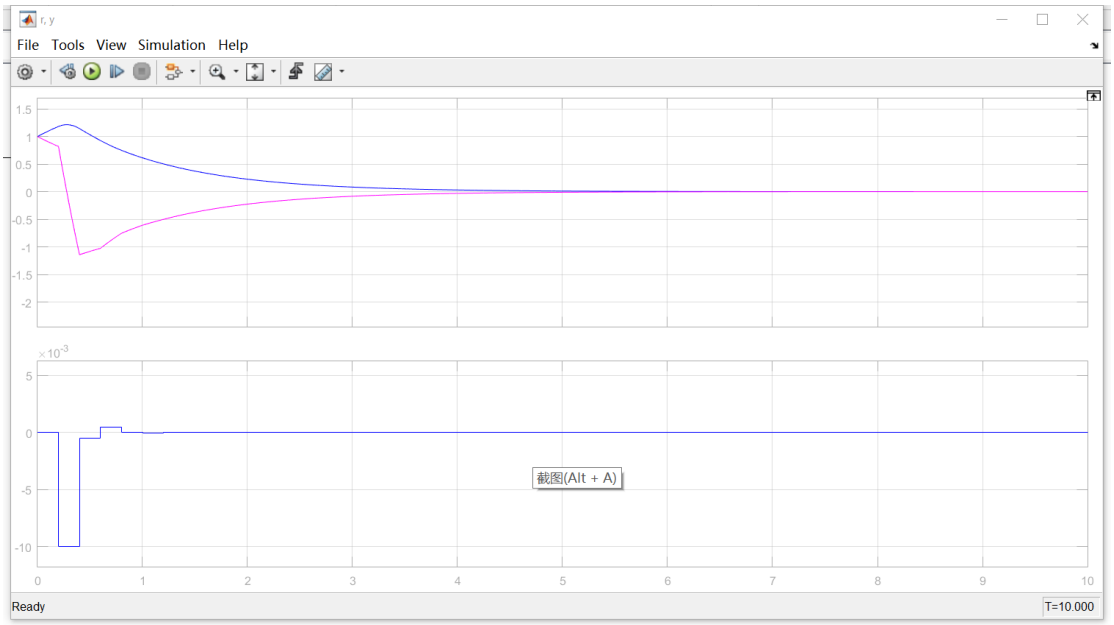
3.3.1 设置 h=0.2s，考察丢包率对于系统稳定性的影响

（1） 丢包率为 P=0 时的仿真结果



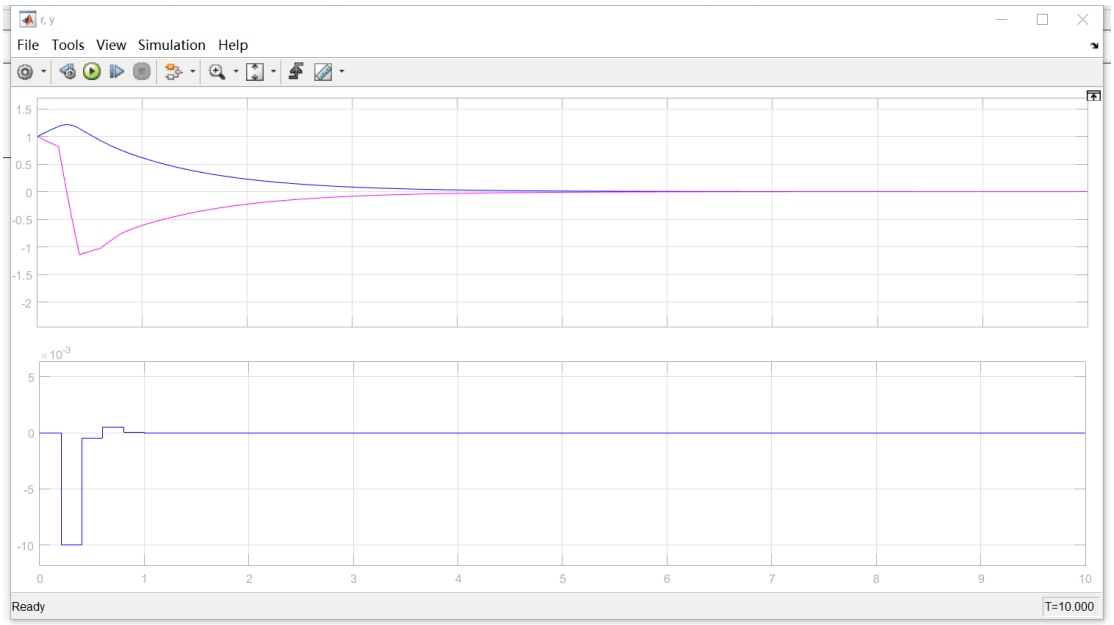
系统稳定！

（2） 丢包率为 P=0.3 时的仿真结果



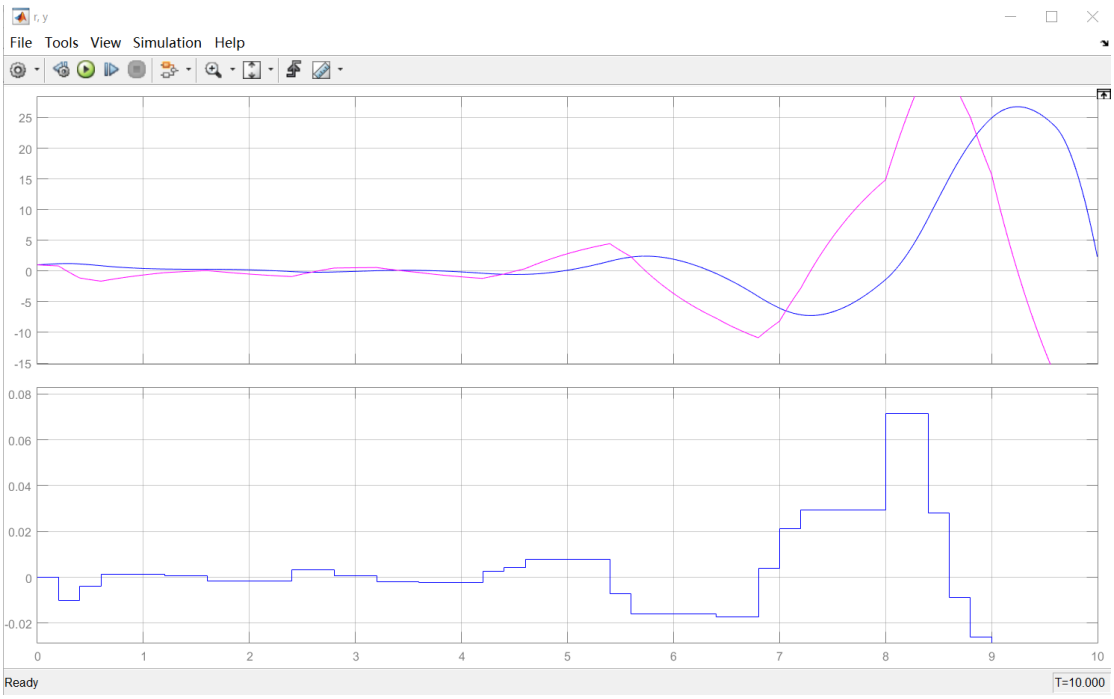
系统稳定！

（3） 丢包率为 P=0.6 时的仿真结果



系统稳定！因为在试验要求的数据中，没有观察到系统不稳定的情况，所以我继续增加丢包率。

（4） 丢包率 P=0.95

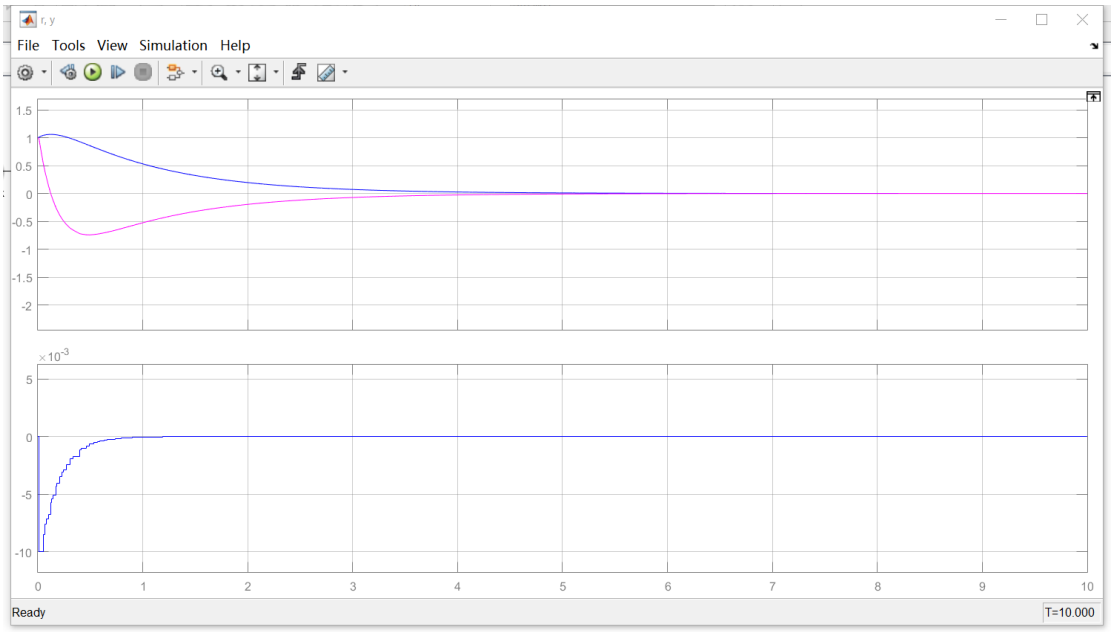


系统变的不稳定。

从上述三组参数下的系统响应效果来看，我们可以得到系统 h 一定时，随着丢包率的增大，系统逐渐变得不稳定。

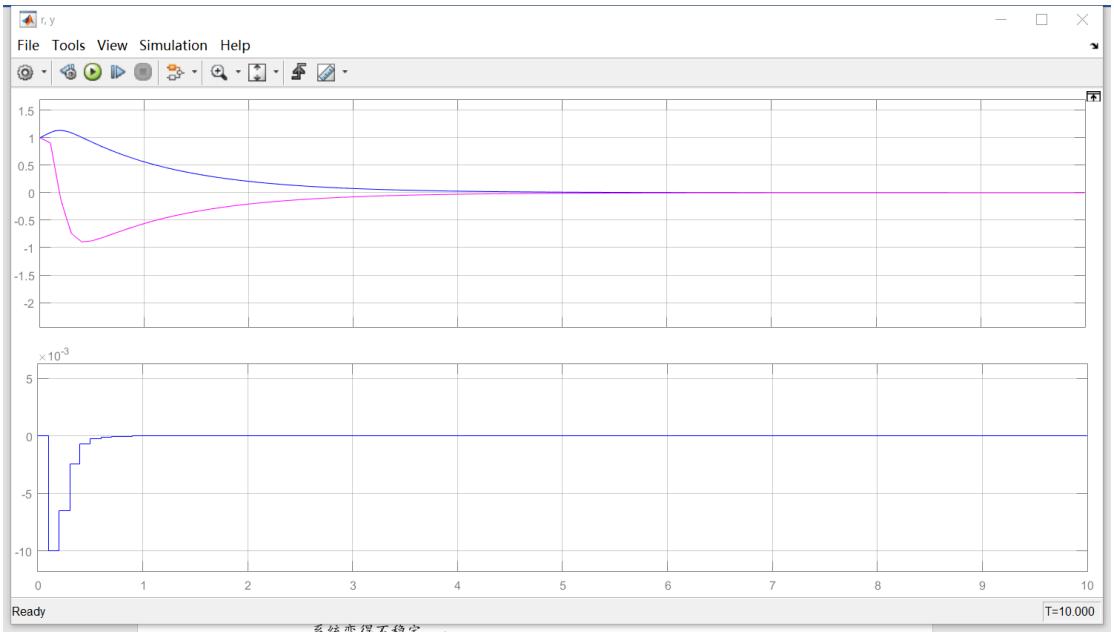
3.3.2 固定丢包率，考察 h 对与系统的影响（丢包率 P=0.8）

（1）设置 h=0.01s



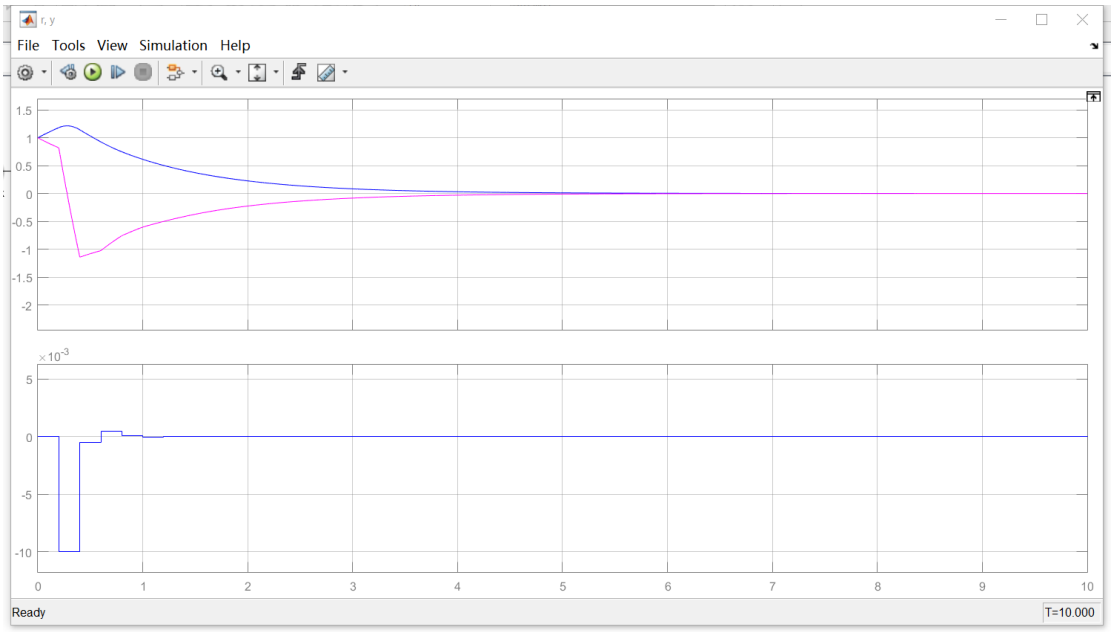
系统稳定！

（2）设置 h =0.1s



系统稳定！

（3）设置 h=0.2s



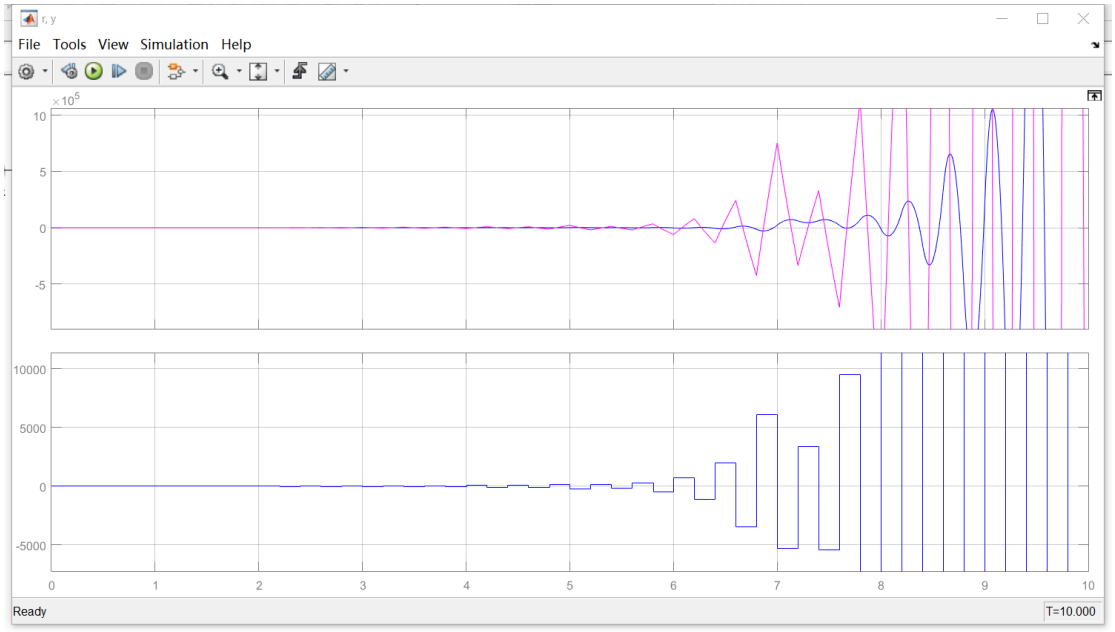
系统依旧稳定！

理论上，采样周期越大(获取真实信息越不及时)，系统越不容易稳定，将 h 设为 0.01，0.1，0.2 时系统的输出信号曲线和控制信号曲线分别如上所示，可以看到随着采样周期的增大，在丢包率一定的情况下系统的波动更加明显甚至会不稳定。但是实验效果并不是特别明显。

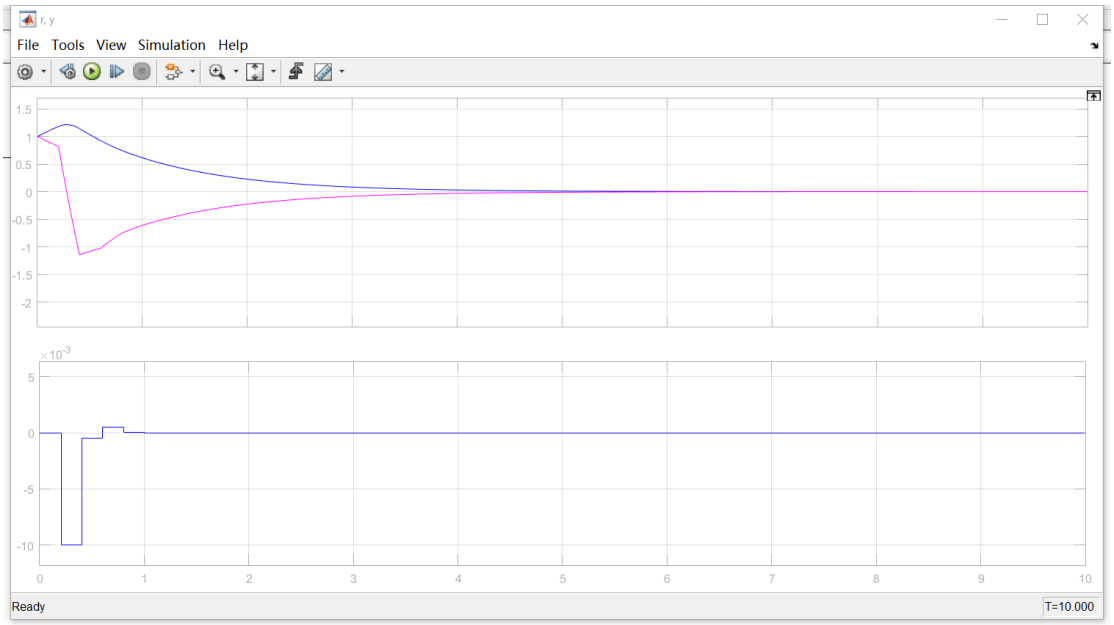
3.3.3 K 的取值的影响

试验过程中，发现改变实验参数，效果并不是特别明显，我认为可能是 K=[-0.005 -0.005]的取值过小，以至于改变实验参数的影响被削弱。所以我尝试将 K 值调大再对实验现象进行了观测。

（1）h=0.2，p=0.6，K=[-0.015 -0.015]



（2）h=0.2，p=0.6，K=[-0.005 -0.005]



两张图进行对比，可以看出增大 K 的取值会让实验效果更加明显。

3.3.4 实验结论

通过上述三组对比试验，我们可以得到如下结论：

1）在 h 一定，K 值固定的情况下，增大丢包率会让系统变得不稳定

2）适当增大 K 值，会使得改变丢包率以及 h 值的效果更加明显

3）h 的取值会影响系统的稳定性，因为其影响了信息的及时性，故 h 越大系统的不稳定性增加。

四. 实验结果分析（结合稳定性判据及其仿真结果进行分析）

公式推导：

$$\dot{x}(t) = Ax(t) + Bu^*(t)$$

$$u^*(t) = \begin{cases} u(t) = Kx(t) & \text{不丢包} \\ u(t-1) = Kx(t-1) & \text{丢包} \end{cases}$$

其中，$A = \begin{bmatrix} 0 & 1 \\ 0 & -1 \end{bmatrix}, B = \begin{bmatrix} 0 \\ 1000 \end{bmatrix}$

Bernoulli过程

$$x_{k+1} = A_{mk}x_k = \begin{cases} \underbrace{\left[e^{Ah} - \int_0^h e^{As}B\bar{K}ds\right]}_{:=A_0} x_k & \text{没有丢包的概率: } 1\text{-}p \\ e^{Ah}x_k \quad :=A_1 & \text{丢包的概率：} p \end{cases}$$

当网络未出现丢包时：

$$x(t) = Ax(t) + BKx(t)$$

$$\mathrm{x_{k+1}} = \left[e^{Ah} + \int_0^h e^{As}dsBK\right]x_k$$

令$\mathrm{A_0} = \left[e^{Ah} + \int_0^h e^{As}dsBK\right]$

当网络存在丢包现象时：

$$x(t) = Ax(t) + BKx(t-1)$$

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

又因为：$x_k = A_0 x_{k-1}$

$$x_{k+1} = \left[e^{Ah} + \int_0^h e^{As}dsBK\, A_0^{-1}\right]x_k$$

令$\mathrm{A_1} = \left[e^{Ah} + \int_0^h e^{As}dsBK\, A_0^{-1}\right]$

这里我们假设系统没有连续丢包，故可以得到上述关系式。若系统发生连续丢包则对应的公式应该为：

定义新的状态空间，令$\zeta_{\mathrm{k}} = \begin{bmatrix} x_k \\ u_{k-1} \end{bmatrix}$

当网络未出现丢包时：

$$\zeta_{\mathrm{k+1}} = \begin{bmatrix} x_{k+1} \\ u_k \end{bmatrix} = \begin{bmatrix} e^{Ah} + \int_0^h e^{As}BKds & 0 \\ K & 0 \end{bmatrix}\begin{bmatrix} x_k \\ u_{k-1} \end{bmatrix} = \begin{bmatrix} e^{Ah} + \int_0^h e^{As}BKds & 0 \\ K & 0 \end{bmatrix}\zeta_{\mathrm{k}}$$

当网络存在丢包时：

$$\zeta_{\mathrm{k+1}} = \begin{bmatrix} x_{k+1} \\ u_k \end{bmatrix} = \begin{bmatrix} e^{Ah} & \int_0^h e^{As}Bds \\ 0 & 1 \end{bmatrix}\begin{bmatrix} x_k \\ u_{k-1} \end{bmatrix} = \begin{bmatrix} e^{Ah} + \int_0^h e^{As}Bds & 0 \\ 0 & 1 \end{bmatrix}\zeta_{\mathrm{k}}$$

相应地：

$$\mathrm{A_0} = \begin{bmatrix} e^{Ah} + \int_0^h e^{As}BKds, & 0 \\ K & 0 \end{bmatrix}$$

$$\mathrm{A_1} = \begin{bmatrix} e^{Ah} & \int_0^h e^{As}Bds \\ 0 & 1 \end{bmatrix}$$

我们可以令状态 0 表示为不丢包，1 表示为丢包，丢包率为 p。因为题设假设不会发生连续的丢包，所以此系统的 Markovian 线性跳变系统可以如下表示：

不丢包状态到不丢包状态：

$$\xi_{k+1} = A_0\xi_k \qquad P(m_{k+1} = 0|m_k = 0) = 1 - p$$

丢包状态到不丢包状态为 100%（不发生连续丢包）：

$$\xi_{k+1} = A_0\xi_k \qquad P(m_{k+1} = 0|m_k = 1) = 1$$

不丢包状态到丢包状态：

$$\xi_{k+1} = A_1\xi_k \qquad P(m_{k+1} = 1|m_k = 0) = p$$

丢包状态到丢包状态为 0%（不发生连续丢包）：

$$\xi_{k+1} = A_1\xi_k \qquad P(m_{k+1} = 1|m_k = 1) = 0$$

线性跳变系统是MSS的，如果

$$P > 0, i = 0, \cdots, N$$

$$(1-p)A_0^T PA_0 + pA_1^T PA_1 - P \prec 0$$

根据稳定性判据，如果存在 P>0，使得$(1-p)A_0^T PA_0 + pA_1^T PA_1 - P < 0$则系统是稳定的。所以我们通过 matlab 的对于 LMI 进行求解可以得到理论的系统稳定性，可与实际效果进行比较。这里，以系统不会连续丢包的计算为例，进行稳定性的分析

$$e^{Ah} = \begin{bmatrix} 1 & 0.1813 \\ 0 & 0.8187 \end{bmatrix}$$

$$H = e^{Ah} + \int_0^h e^{As} dsBK = \begin{bmatrix} 1 & 0.1813 \\ -0.90635 & -0.08765 \end{bmatrix}$$

$$A_0 = \left[e^{Ah} + \int_0^h e^{As} dsBK \right] = \begin{bmatrix} -0.90635 & 0.08765 \\ -0.906345 & -0.08764 \end{bmatrix}$$

$$A_0^{-1} = \begin{bmatrix} 1070 & -1071 \\ 11070 & 11071 \end{bmatrix}$$

$$A_1 = \left[e^{Ah} + \int_0^h e^{As} dsBK\, A_0^{-1} \right] = \begin{bmatrix} -935.5 & -936.3 \\ 9063.4 & 9064.3 \end{bmatrix}$$

利用 matlab 中的求解 LMI 的工具包解 MSS 的线性矩阵不等式，用 feasp 求解 LMI 之后得到 tmin，tmin 小于 0 则不等式有解，而 tmin 大于 0 则说明不等式没有解。利用如下图所示的代码对不等式矩阵进行求解。

```
A0 = [0.9064 0.08765;-0.9063 -0.08765];
A1 = [-935.5 -936.3;9063.4 9064.3];
%A0 = a0;A1=a1;
p = 0;
setlmis([]);
P = lmivar(1,[2,1]);
lmiterm([1,1,1,P],(1-p)*A0',A0);
lmiterm([1,1,1,P],p*A1',A1);
lmiterm([1,1,1,P],-1,1);
lmisys = getlmis;
[tmin, feas] = feasp(lmisys);
PP = dec2mat(lmisys,feas,P)
tmin
```

（1） Zero-Order-Hold=0.2s 丢包率 P=0.6

```
Result:  best value of t:    -29.016700
         f-radius saturation:  0.000% of R =  1.00e+09


PP =

   1.0e+03 *

    1.4968    0.1547
    0.1547    0.0160


tmin =

  -29.0167
```

tmin = -29.0167<0，则有解，可以找到合适的 P 矩阵，所以系统是稳定的。仿真结果与理论结果相符

（2） Zero-Order-Hold=0.2s 丢包率 P=0.3

```
Result:  best value of t:  -106.032329
         f-radius saturation:  0.000% of R =  1.00e+09


PP =

   1.0e+03 *

    3.3606    0.3475
    0.3475    0.0359


tmin =

 -106.0323
```

tmin = -106.0323<0，有解，可以找到合适的 P 矩阵，所以系统是稳定的。仿真结果与理论结果相符。

（3） Zero-Order-Hold=0.2s 丢包率 P=0

```
Result:  best value of t: -1.603173e+08
         f-radius saturation:  99.005% of R =  1.00e+09


PP =

   1.0e+08 *

    9.5936    0.7727
    0.7727    1.6779


tmin =

  -1.6032e+08
```

tmin = -1.6032e+08<0，有解，可以找到合适的 P 矩阵，所以系统是稳定的。仿真结果与理论结果相符。

五．实验总结与感悟

本实验中主要用到的工具有 truetime 和 matlab。其中 truetime 是首次接触，但是经过对于样例的理解，很容易上手。通过这两种工具依旧 matlab 中关于求解 LMI 的工具，我对于实验要求的系统进行了仿真。将仿真得到的

结果与理论值进行比较，发现契合度比较高，说明实验操作是成功的。

通过本次实验，加强了我对于网络丢包问题造成的系统不稳定现象的理解，同时也加强了我对于稳定性判据的理解与应用。整个过程是收获颇丰。不仅熟悉了 truetime 这个工具包的使用，也重新巩固了 matlab 的使用，通识让我对于系统稳定性判据有了更加深刻的理解。从最开始的不知从何下手，到最后独立完成实验，得益于老师、助教以及同学们的帮助，在此深表谢意。感谢大家的帮助能够让我顺利完成实验，并收获很多。