

1 实验目的

本次实验主要是对网络化系统的仿真方法进行探究，基于matlab中的TrueTime工具箱进行仿真，针对随机丢包问题的系统模型，对网络控制系统的稳定性和性能进行评估。实验目的为：

- 了解网络化控制系统的仿真方法
- 学习使用TrueTime仿真工具，进行仿真分析
- 实现具有随机数据丢包的网络控制系统设计

对于通信网络来说，由于网络传输的特点，在每一次传输过程中数据包的丢失状况是随机出现的，数据丢包过程是一类随机过程，本项目重点实现随机丢包模型的控制器设计方法。首先将随机丢包过程建模为一个随机的Markov链，系统模型则可表示为一类Markov跳变系统，利用Lyapunov稳定性和Markov跳变系统理论可以分析闭环系统稳定性，进而给出控制器的设计方法。

2 仿真平台搭建

本实验使用的控制系统仿真工具为Matlab/Simulink，利用TrueTime工具箱进行仿真，Truetime是由瑞典Lund工学院Henriksson等人开发的一个基于Matlab/Simulink的实时网络控制系统的仿真工具箱，为NCS理论的仿真研究提供了简易可行、功能齐全的手段，拜托了软件编程实现特定的网络通讯协议、通信延迟所带来的困难，支持控制与实时调度同时仿真可以方便地仿真实时系统中的资源调度问题。TrueTime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）。

内核模块可以是时间驱动也可以是事件驱动的，它主要包含了一个实时内核，A/D，D/A转换端口，与网络模块连接的信号端口（信号接收（Rcv），信号发送（Snd）），实时调度（schedule）显示端口等，调度器与监视器的输出用于显示仿真过程中公共资源（CPU、监控器、网络）的分配，此外，它还有一个外部中断通道（Interrupts）可以处理外部中断。任务和中断处理器的执行需要通过用户自定义函数来实现。调度策略使用一个优先权函数来决定任务的属性。

网络模块是事件驱动的，当有消息进入或离开网络时它便执行。一条消息包含的信息有发送和接收节点号，用户数据（如测量信号和控制信号），消息的长度和其他可选的实时属性（如优先级或最终时限等）。网络模块包含两个信号端口（信号接收（Rcv），信号发送（Snd）），以及一个实时调度（schedule）显示端口。其中收发信号端口可以通过Matlab模块扩充至多个接口，TrueTime中预定义了多种调度策略，包括固定优先级（Fixed Priority），单调速率（RM，Rate Monotonic），截止期单调（DM，Dead line Monotonic），最小截止期优先（EDF，Earliest Dead line First），同时，它还有多种介质访问控制协议（CSMA/CD,CSMA/CA,Round Robin,FDMA或TDMA）和相应的参数可以选择。

利用TrueTime仿真软件，网络控制系统中的各个处理单元（包括传感器、控制器和执行器）都可以由计算机模块构建，而网络控制系统的实时网络可以由所需协议的网络模

块来构建，另外，再结合Matlab/Simulink 的其他控制模块，就可以简便而又快速的构建一个实时的网络控制系统。利用TrueTime仿真软件包的优点在于：

1. 由于该仿真软件中两个基本模块具有通用性，在构建各个处理单元时只需选用其相应的接口功能进行连接即可，因此大大加快模型构建的速度。
2. 该仿真软件可以比较方便模拟各种实时调度策略，并通过Scope可以很方便地观察各个任务的调度情况和对象的输出情况。
3. 在网络模块中，可以很方便的模拟数据传输率、数据包的大小和丢包率等网络参数，有利于分析各类参数对网络控制系统的性能影响。

使用TrueTime进行仿真时，首先要对网络控制系统中的内核模块TrueTime Kernel和网络模块TrueTime Network以及各个节点进行初始化，在初始化中需要完成以下工作：

1. 初始化功能块内核，设置功能块输入、输出端口的数目和调度策略。
2. 定义消息函数，并根据节点采用的驱动方式，设置不同的消息调度策略。对于时钟驱动节点，调用ttCreatPeriodicTask函数，设置周期性的任务调度策略，以实现定时采样功能。对于事件驱动节点，调用ttCreateInterruptHandler 函数，设置中断式消息调度策略，使节点在接受到网络数据后触发相应的消息。
3. 初始化网络端口，设置节点对应的网络端口代号。控制网络功能由TrueTime Network功能块实现。网络类型、节点数、传输速率以及丢包率等参数可以通过TrueTime Network功能块的设置窗口进行设置。具体的参数选项根据网络类型的不同而不同。

3 实验步骤

考虑如下的网络化控制系统：

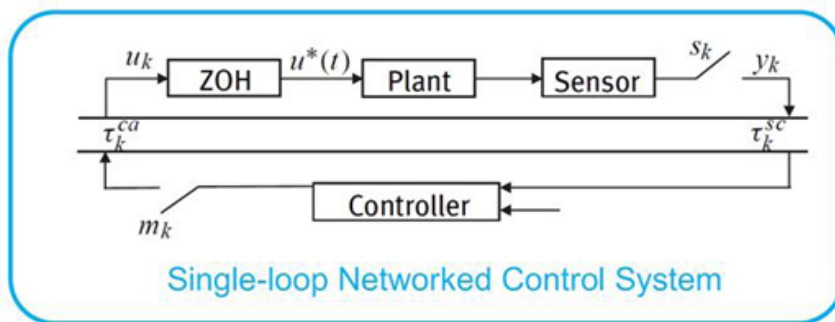


Figure 1: 系统的控制结构图

假设系统存在随机丢包，丢包过程是一个Bernoulli 过程，记为 s_k ，且 $s_k = 1$ 表示输出变量 y_k 存在丢包， $s_k = 0$ 表示输出变量 y_k 成功传送至控制器。图中的传输时延假设为零，并假设控制器至执行器的传输不存在丢包，即 $m_k = 0$

系统为离散时间系统，不需要从连续系统采样，可直接用discrete-time模块构建被控对象，其状态空间模型可以描述成：

$$x(k+1) = \begin{bmatrix} -0.2656 & -2.2023 \\ -1.1878 & 0.9863 \end{bmatrix} x(k) + \begin{bmatrix} -0.5186 \\ 0.3274 \end{bmatrix} u(k) \quad (1)$$

$$y(k) = \begin{bmatrix} 0.2341 & 0.0215 \end{bmatrix} x(k) \quad (2)$$

假设丢包的概率为 p ,可定义：

$$y_c(k) = \begin{cases} y(k), & s_k = 0 \\ y(k-1), & s_k = 1 \end{cases} \quad (3)$$

由以上网络化控制系统，我们要设计相应的动态输出反馈控制器：

$$\begin{cases} x_c(k+1) = A_c x_c(k) + B_c y_c(k) \\ u(k) = C_c x_c(k) \end{cases} \quad (4)$$

对于本仿真实验，按照TrueTime的仿真流程，可以把它分为三步，即仿真系统模块图、模块初始化、编写任务代码，接下来我们按照实验流程介绍具体的步骤。

3.1 仿真系统模块图

3.1.1 仿真系统设计方案一

方案一我们采用将动态反馈输出控制器利用离散时间状态方程在simulink模块中进行了实现，基于系统的流程图，通过阅读network和wireless两个示例文件，我们在wireless的基础上搭建了系统模块图如图 2：

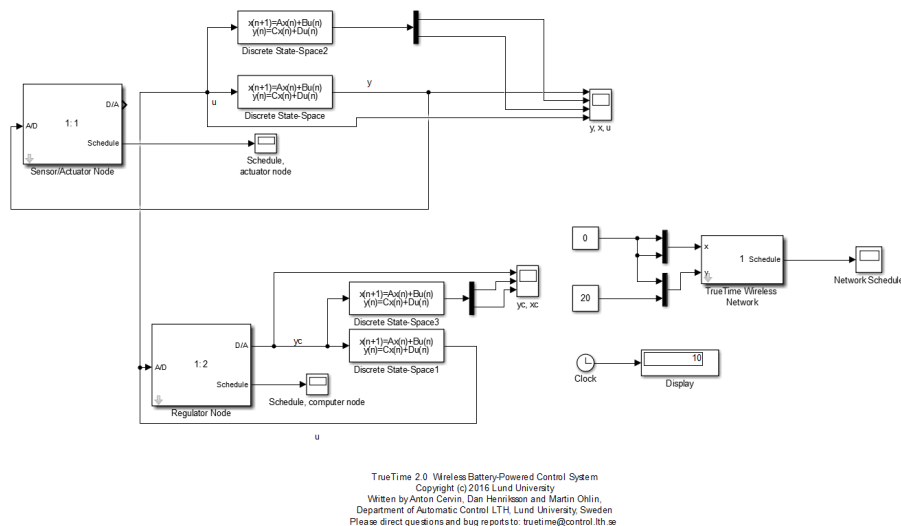
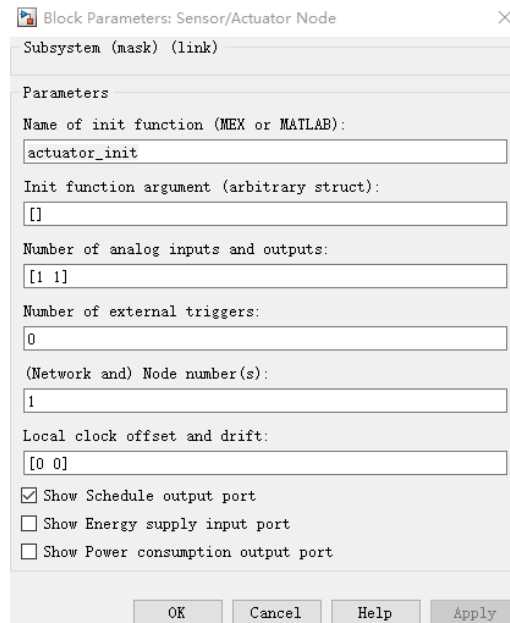


Figure 2: 方案一系统模块图

从图中我们可以得到，系统的反馈回路为控制器节点输出 $u(k)$ 直接传入控制对象，即本题中的状态空间方程，输出 $y(k)$ ，进入传感器节点，在Sensor/Actuator

Node的TrueTime Kernel中，模拟了传感器节点到控制器再到执行器节点的过程。在其中，TrueTime Wireless Network模块模拟了网络的部分，在其中我们可以设置丢包率的问题。本模拟中，我们假设控制器至执行器的传输不存在丢包，即 $m_k = 0$ ，只存在输出变量 $y(k)$ 的丢包问题。并且传输时延为0，故我们直接置空Sensor/Actuator Node的输出口，将控制器输出直接连接到被控对象，这样就能较为巧妙的屏蔽了 $m_k = 0$ 的丢包问题。Sensor/Actuator Node参数如图 3:



Block Parameters: Sensor/Actuator Node

Subsystem (mask) (link)

Parameters

Name of init function (MEX or MATLAB):
actuator_init

Init function argument (arbitrary struct):
[]

Number of analog inputs and outputs:
[1 1]

Number of external triggers:
0

(Network and) Node number(s):
1

Local clock offset and drift:
[0 0]

☒ Show Schedule output port
☐ Show Energy supply input port
☐ Show Power consumption output port

OK Cancel Help Apply

Figure 3: 传感/执行器参数

控制器模块，输出状态 y_c 传入动态输出反馈控制器的状态空间方程，得到对应的输出 u_k ，被直接连接到系统状态方程输入，控制被控对象。控制器参数如图4:

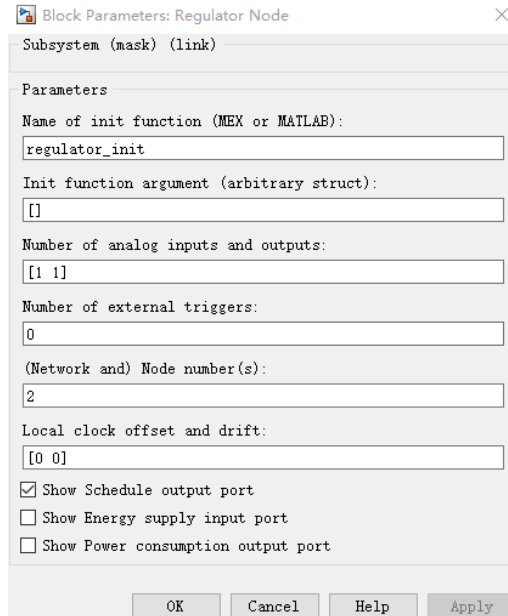


Figure 4: 控制器参数

网络类型我们采用IEEE 802.11b/g (WLAN) 的方式进行远程控制，具体参数如图 5:

系统状态变量为状态空间方程内部变量，为了能够得到其图像，我们又用了两个Discrete State-Space模块，将 D 矩阵置为0，将 C 矩阵置为单位矩阵,得到了 $x(k)$ 和 $x_c(k)$ 。

3.1.2 仿真系统设计方案二

方案二我们采用在Control Node内部实现控制，这样的好处是我们可以内部自行模拟丢包，使用多套控制器进行控制，此时我们可以作出系统的simulink仿真图如图 6:

在内部，我们通过随机函数实现丢包率控制，其实现代码如下:

```
1 function [exectime, data] = ctrlcode(seg, data)
2
3 Ac0=[-0.2126   -0.0640   -0.0249;
4       -1.2209   -0.3637   -0.0026;
5       -0.0001   -0.0000   -0.0725];
6 Ac1=[-0.9689   -0.1132   -0.0023;
7        0.4628   -0.2217    0.0015;
8        0.0026    0.0002    0.0000];
9 Bc0=[-0.0265 -0.0015 -1.0723] ;
10 Bc1=[-3.2039 7.1770 -0.9890] ;
11 Cc0=[ 0.1033    4.1218    0.0033];
12 Cc1=[0.0904    4.1609   -0.0044];
```



Block Parameters: TrueTime Wireless Network

Wireless Network (mask) (link)

Parameters

Network type: 802.11b (WLAN)

Network Number: 1

Number of nodes: 2

Data rate (bits/s): 800000

Minimum frame size (bits): 272

Transmit power (dbm): 30

Receiver signal threshold (dbm): -48

Pathloss function: default

Pathloss exponent (1/distance^x): 3.5

ACK timeout (s): 0.00004

Retry limit: 5

Error coding threshold: 0.03

Loss probability (0-1): 0.05

Initial seed: 0

☒ Show Schedule output port

☐ Show Power consumption output port

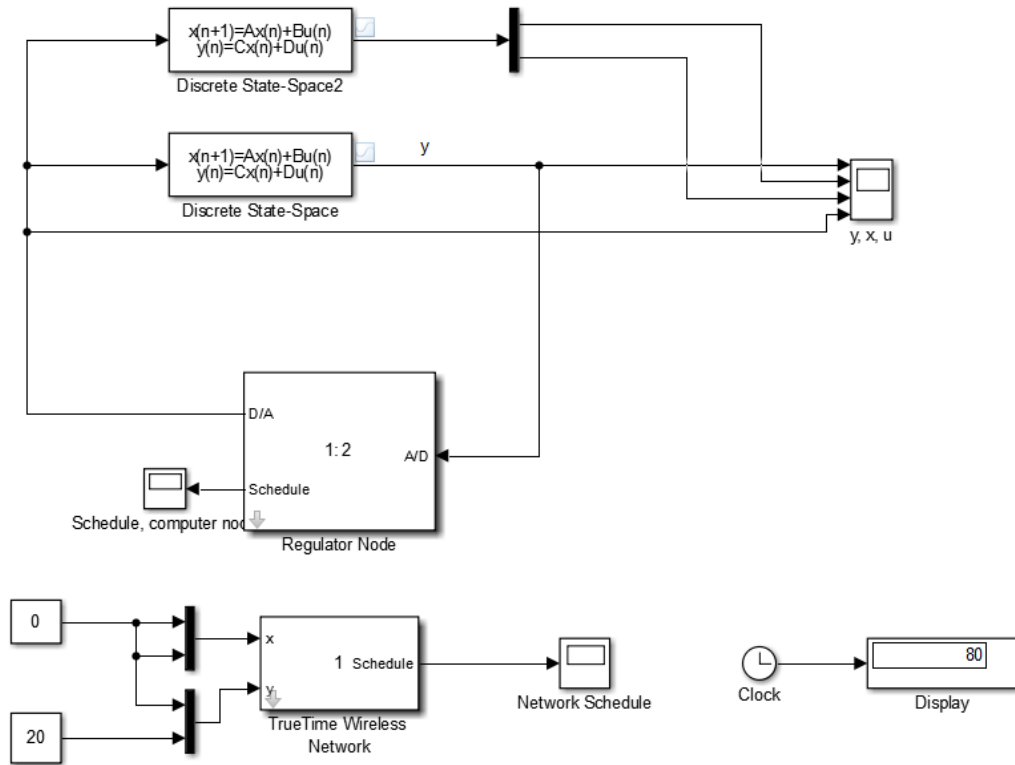
OK Cancel Help Apply

Figure 5: 网络节点参数

```

13
14 p = 0.05;
15
16 switch seg,
17 case 1,
18   % Read all buffered packets
19   temp = ttTryFetch( sensor_signal );
20   while ~isempty(temp),
21     y = temp;
22     temp = ttTryFetch( sensor_signal );
23   end
24
25
26   if (unifrnd(0,1)>p)
27     data.yc = ttAnalogIn(1);
28     data.xcnew=Acl*data.xc+Bcl*data.yc;
29     data.u=Ccl*data.xc;

```



TrueTime 2.0 Wireless Battery-Powered Control System
Copyright (c) 2016 Lund University
Written by Anton Cervin, Dan Henriksson and Martin Ohlin,
Department of Automatic Control LTH, Lund University, Sweden
Please direct questions and bug reports to: truetime@control.lth.se

Figure 6: 方案二系统模块图

```

30     data.xc=data.xcnew;
31     ttAnalogOut(1,data.u);
32     else
33         data.xcnew=Ac0*data.xc+Bc0*data.yc;
34         data.u=Cc0*data.xc;
35         data.xc=data.xcnew;
36         ttAnalogOut(1,data.u);
37     end
38     msg.msg = data.xc;
39     msg.type = control_signal ;
40     ttSendMsg(1, msg, 80);    % Send 80 bits to node 1 (actuator)
41     exectime = -1; % finished
42 end

```

3.2 模块初始化

在搭建好仿真系统模块图后，我们就要对其中的执行器控制器编写相应的初始代码。对于actuator_init和regulator_init初始化，直接可以参照系统给出的wirelss示例，代码段如下：

```
1 function actuator_init
2
3 % Distributed control system: actuator node
4 %
5 % Receives messages from the controller and actuates
6 % the plant.
7
8 % Initialize TrueTime kernel
9 ttInitKernel( prioFP ); % fixed priority
10 ttSetKernelParameter( energyconsumption , 0.010); % 10 mW
11
12 % Create mailboxes
13 ttCreateMailbox( control_signal , 10)
14 ttCreateMailbox( power_ping , 10)
15 ttCreateMailbox( power_response , 10)
16
17 % Create sensor task
18 data.y = 0;
19 offset = 0.0;
20 period = 0.010;
21 prio = 1;
22 ttCreatePeriodicTask( sens_task , offset, period, senscode , data);
23 ttSetPriority(prio, sens_task );
24
25 % Create actuator task
26 deadline = 100;
27 prio = 2;
28 ttCreateTask( act_task , deadline, actcode );
29 ttSetPriority(prio, act_task );
30
31 % Create power controller task
32 offset = 2.07;
33 period = 0.025;
34 prio = 3;
35 power_data.transmitPower = 20;
36 power_data.name = 1; % We are node number 1 in the network
37 power_data.receiver = 2; % We are communicating with node 2
38 power_data.haverun = 0; % We have not run yet
39 ttCreatePeriodicTask( power_controller_task , offset, period, ...
40     powctrlcode , power_data);
41 ttSetPriority(prio, power_controller_task );
```



```
41
42 % Create power response task
43 deadline = 100;
44 prio = 4;
45 ttCreateTask( power_response_task , deadline, powrespcode );
46 ttSetPriority(prio, power_response_task );
47
48 % Initialize network
49 ttCreateHandler( nw_handler , 1, msgRcvActuator );
50 ttAttachNetworkHandler( nw_handler );
```

```
1 function regulator_init
2
3 % Distributed control system: regulator node
4 %
5 % Receives messages from the sensor node, computes control signal
6 % and sends it back to the actuator node.
7
8 % Initialize TrueTime kernel
9 ttInitKernel( prioFP ); % fixed priority
10 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 );
56 ttAttachNetworkHandler( nw_handler );
```

3.3 编写任务代码

最后一节中，将控制器任务(task)代码的编写，任务(task)用来模拟用户代码的执行任务，分为两类：

1. 任务的周期性时钟触发：ttCreatePeriodicTask(name,starttime, period, codeFcn, data)
2. 任务的非周期性时钟、事件、中断触发：ttCreateTask(name, deadline, codeFcn, data), ttCreateJob (taskname,time), ttKillJob(taskname)

参照示例wirelss中的代码，我们可以对任务代码进行编写。实现控制任务分成了两个case，在case 1中我们使用ttAnalogOut向动态输出反馈控制器输出 y_c ，在case 2中我们获得系统的控制信号并将其发送，实际上由于我们直接将动态输出反馈控制器的输出 u 连接到了状态空间方程，case 2的部分也可以省略。

```
1 function [exectime, data] = ctrlcode(seg, data)
2
3 switch seg,
4 case 1,
5     % Read all buffered packets
6     temp = ttTryFetch( sensor_signal );
7     while ~isempty(temp),
```

```

8     y = temp;
9     temp = ttTryFetch( sensor_signal );
10  end
11
12  data.yc = y;
13  ttAnalogOut(1, data.yc);
14  exectime = 0.0005;
15  case 2,
16  data.u = ttAnalogIn(1);
17  msg.msg = data.u;
18  msg.type = control_signal ;
19  ttSendMsg(1, msg, 80);    % Send 80 bits to node 1 (actuator)
20  exectime = -1; % finished
21 end

```

4 实验结果分析

4.1 控制器设计

4.1.1 两套控制器设计

已知具有随机丢包问题的系统可以用如下状态空间方程表示

$$\begin{cases} x(k+1) = Ax(k) + Bu(k) \\ y(k) = Cx(k) \end{cases} \quad (5)$$

其中

$$A = \begin{bmatrix} -0.2656 & -2.2023 \\ -1.1878 & 0.9863 \end{bmatrix}, B = \begin{bmatrix} -0.5186 \\ 0.3274 \end{bmatrix}, C = \begin{bmatrix} 0.2341 & 0.0215 \end{bmatrix}$$

根据统计，丢包的概率为 p ，且可定义 $y_c(k) = \begin{cases} y(k), s_k = 0 \\ y(k-1), s_k = 1 \end{cases}$

为以上系统设计动态输出反馈控制器

$$\begin{cases} x_c(k+1) = A_c x_c(k) + B_c y_c(k) \\ u(k) = C_c x_c(k) \end{cases} \quad (6)$$

根据以上式子，可以定义系统新的增广状态向量

$$\xi(k) = \begin{bmatrix} x(k) \\ y_c(k-1) \end{bmatrix} \quad (7)$$

由此我们可以得到如下的系统模型

丢包时

$$\xi(k+1) = \begin{bmatrix} x(k+1) \\ y_c(k) \end{bmatrix} \quad (8)$$

$$= \begin{bmatrix} A & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} x(k) \\ y_c(k-1) \end{bmatrix} + \begin{bmatrix} B \\ 0 \end{bmatrix} u(k) \quad (9)$$

$$= A_0 \xi(k) + B_0 u(k) \quad (10)$$

$$y_c(k) = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \xi(k) = C_0 \xi(k) \quad (11)$$

未丢包时

$$\xi(k+1) = \begin{bmatrix} x(k+1) \\ y_c(k) \end{bmatrix} \quad (12)$$

$$= \begin{bmatrix} A & 0 \\ C & 0 \end{bmatrix} \begin{bmatrix} x(k) \\ y_c(k-1) \end{bmatrix} + \begin{bmatrix} B \\ 0 \end{bmatrix} u(k) \quad (13)$$

$$= A_1 \xi(k) + B_1 u(k) \quad (14)$$

$$y_c(k) = \begin{bmatrix} C & 0 \end{bmatrix} \xi(k) = C_1 \xi(k) \quad (15)$$

此时系统还不能判断稳定性，为了可以判断系统的稳定性，对该状态向量继续增广，令新的状态向量为

$$\varepsilon(k) = \begin{bmatrix} \xi(k) \\ x_c(k) \end{bmatrix} \quad (16)$$

则可以得到闭环系统模型

丢包时

$$\varepsilon(k+1) = \begin{bmatrix} \xi(k+1) \\ x_c(k+1) \end{bmatrix} = \begin{bmatrix} A_0 & B_0 C_c \\ B_c C_0 & A_c \end{bmatrix} \begin{bmatrix} \xi(k) \\ x_c(k) \end{bmatrix} = \bar{A}_0 \varepsilon(k) \quad (17)$$

未丢包时

$$\varepsilon(k+1) = \begin{bmatrix} \xi(k+1) \\ x_c(k+1) \end{bmatrix} = \begin{bmatrix} A_1 & B_1 C_c \\ B_c C_1 & A_c \end{bmatrix} \begin{bmatrix} \xi(k) \\ x_c(k) \end{bmatrix} = \bar{A}_1 \varepsilon(k) \quad (18)$$

控制器的参数选取需要建立在系统均方稳定的前提下，由于该系统存在丢包和未丢包两种状态，可以视为只有两种状态的Markovian线性跳变系统。Markovian线性跳变系统的稳定性条件为：

存在 $P_i > 0, i = 0, \dots, L$ 满足如下任一条件

$$P_i - A_i^T \left(\sum_{j=1}^L p_{ij} P_j \right) A_i > 0, i = 1, \dots, L \quad (19)$$

$$P_j - A_j \left(\sum_{i=1}^L p_{ij} P_i \right) A_j^T > 0, j = 1, \dots, L \quad (20)$$

$$P_i - \sum_{j=1}^L p_{ij} A_j^T P_j A_j > 0, i = 1, \dots, L \quad (21)$$

$$P_j - \sum_{i=1}^L p_{ij} A_i P_i A_i^T > 0, j = 1, \dots, L \quad (22)$$

对闭环系统的Markov跳变模型，设 $Z = P^{-1}$ ，应用以上稳定性条件，可以得到

$$\begin{bmatrix} Z & * & * \\ \sqrt{p} \bar{A}_0 Z & Z & 0 \\ \sqrt{1-p} \bar{A}_1 Z & 0 & Z \end{bmatrix} > 0 \quad (23)$$

在查阅资料后得到如下定理，可在设计丢包和未丢包两套控制器时使用：

如果存在矩阵 $Y = Y^T$ ， $X = X^T$ ， L_i ， F_i 和 W_i ， $i = \{0, 1\}$ ，满足如下的线性矩阵不等式，则上述闭环系统均方稳定

$$\begin{bmatrix} \begin{bmatrix} Y & I \\ I & X \end{bmatrix} & * & * \\ \sqrt{p} \begin{bmatrix} Y \bar{A}_0 + L_0 \bar{C}_0 & W_0 \\ \bar{A}_0 & \bar{A}_0 X + \bar{B}_0 F_0 \end{bmatrix} & \begin{bmatrix} Y & I \\ I & X \end{bmatrix} & * \\ \sqrt{1-p} \begin{bmatrix} Y \bar{A}_1 + L_1 \bar{C}_1 & W_1 \\ \bar{A}_1 & \bar{A}_1 X + \bar{B}_1 F_1 \end{bmatrix} & \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} Y & I \\ I & X \end{bmatrix} \end{bmatrix} > 0 \quad (24)$$

```

1 A=[-0.2656 -2.2023;-1.1878 0.9863];
2 B=[-0.5186;0.3274];
3 C=[0.2341 0.0215];
4 p=0.05;
5 %p=0.12;
6
7 A1=[A(1,1),A(1,2),0;A(2,1),A(2,2),0;C(1,1),C(1,2),0];
8 B1=[B(1,1);B(2,1);0];
9 C1=[C 0];
10 A0=[A(1,1),A(1,2),0;A(2,1),A(2,2),0;0,0,1];
11 B0=[B(1,1);B(2,1);0];
12 C0=[0 0 1];
13 I=[1,0,0;0,1,0;0,0,1];
14
15 setlmis([]);
16 Y=lmivar(1,[3,1]);
17 X=lmivar(1,[3,1]);
18 L0=lmivar(2,[3,1]);
19 L1=lmivar(2,[3,1]);
20 W0=lmivar(2,[3,3]);
21 W1=lmivar(2,[3,3]);
22 F0=lmivar(2,[1,3]);
23 F1=lmivar(2,[1,3]);
24
25 lmiterm([-1,1,1,Y],1,1);

```

```

26  lmiterm([-1,2,1,0],I);
27  lmiterm([-1,2,2,X],1,1);
28  lmiterm([-1,3,1,Y],sqrt(p),A0);lmiterm([-1,3,1,L0],sqrt(p),C0);
29  lmiterm([-1,3,2,W0],sqrt(p),1);
30  lmiterm([-1,3,3,Y],1,1);
31  lmiterm([-1,4,1,0],sqrt(p)*A0);
32  lmiterm([-1,4,2,X],sqrt(p)*A0,1);lmiterm([-1,4,2,F0],sqrt(p)*B0,1);
33  lmiterm([-1,4,3,0],I);
34  lmiterm([-1,4,4,X],1,1);
35  lmiterm([-1,5,1,Y],sqrt(1-p),A1);lmiterm([-1,5,1,L1],sqrt(1-p),C1);
36  lmiterm([-1,5,2,W1],sqrt(1-p),1);
37  %
38  lmiterm([-1,5,3,0],0);
39  lmiterm([-1,5,4,0],0);
40  lmiterm([-1,6,3,0],0);
41  lmiterm([-1,6,4,0],0);
42  %
43  lmiterm([-1,5,5,Y],1,1);
44  lmiterm([-1,6,1,0],sqrt(1-p)*A1);
45  lmiterm([-1,6,2,X],sqrt(1-p)*A1,1);lmiterm([-1,6,2,F1],sqrt(1-p)*B1,1);
46  lmiterm([-1,6,5,0],I);
47  lmiterm([-1,6,6,X],1,1);
48  lmis=getlmis;
49  [tmin,xfas]=feasp(lmis);
50
51  Y=dec2mat(lmis,xfas,Y);
52  X=dec2mat(lmis,xfas,X);
53  L0=dec2mat(lmis,xfas,L0);
54  L1=dec2mat(lmis,xfas,L1);
55  W0=dec2mat(lmis,xfas,W0);
56  W1=dec2mat(lmis,xfas,W1);
57  F0=dec2mat(lmis,xfas,F0);
58  F1=dec2mat(lmis,xfas,F1);
59
60  Bc1=inv(Y)*L1;
61  Bc0=inv(Y)*L0;
62  Cc1=F1*inv(inv(Y)-X);
63  Cc0=F0*inv(inv(Y)-X);
64  Ac1=-inv(Y)*(Y*A1*X+Y*B1*F1+L1*C1*X-W1)*inv(inv(Y)-X);
65  Ac0=-inv(Y)*(Y*A0*X+Y*B0*F0+L0*C0*X-W0)*inv(inv(Y)-X);
66  p
67  Ac0
68  Bc0
69  Cc0
70  Ac1
71  Bc1
72  Cc1

```

在matlab中使用LMI toolbox求解得到以下结果

当 $p = 0.05$ 时, 丢包时的控制器参数

$$A_{c0} = \begin{bmatrix} -0.2126 & -0.0640 & -0.0249 \\ -1.2209 & -0.3637 & -0.0026 \\ -0.0001 & -0.0000 & -0.0725 \end{bmatrix}, B_{c0} = \begin{bmatrix} -0.0265 \\ -0.0015 \\ -1.0723 \end{bmatrix}, C_{c0} = \begin{bmatrix} 0.1033 & 4.1218 & 0.0033 \end{bmatrix}$$

成功传输时的控制器参数

$$A_{c1} = \begin{bmatrix} -0.9689 & -0.1132 & -0.0023 \\ 0.4628 & -0.2217 & 0.0015 \\ 0.0026 & 0.0002 & 0.0000 \end{bmatrix}, B_{c1} = \begin{bmatrix} -3.2039 \\ 7.1770 \\ -0.9890 \end{bmatrix}, C_{c1} = \begin{bmatrix} 0.0904 & 4.1609 & -0.0044 \end{bmatrix}$$

当 $p = 0.12$ 时, 丢包的控制器参数

$$A_{c0} = \begin{bmatrix} -0.2169 & -0.0510 & 0.0219 \\ -1.2186 & -0.3718 & -0.0056 \\ -0.0000 & -0.0000 & -0.0032 \end{bmatrix}, B_{c0} = \begin{bmatrix} 0.0125 \\ 0.0004 \\ -1.0032 \end{bmatrix}, C_{c0} = \begin{bmatrix} 0.0940 & 4.1483 & 0.0183 \end{bmatrix}$$

成功传输时的控制器参数

$$A_{c1} = \begin{bmatrix} -1.4937 & -0.1797 & 0.0003 \\ 0.7424 & -0.1845 & -0.0002 \\ 0.0002 & 0.0000 & 0.0000 \end{bmatrix}, B_{c1} = \begin{bmatrix} -5.4900 \\ 8.3991 \\ -0.9993 \end{bmatrix}, C_{c1} = \begin{bmatrix} 0.1100 & 4.1277 & 0.0005 \end{bmatrix}$$

4.1.2 同一套参数控制器设计与问题反思

当控制器节点的控制参数只有一套时, 即当传感器节点发送的数据存在丢包和成功传输时使用相同的 A_c, B_c, C_c 时, 在将系统增广为 $\epsilon(k)$ 后, 利用Markovian线性跳变系统的稳定性条件可知, 若存在 $P > 0, P = P^T$, 使得

$$p\bar{A}_0^T P \bar{A}_0 + (1-p)\bar{A}_1^T P \bar{A}_1 - P < 0 \quad (25)$$

那么该系统是均方稳定的。

为了使用shur补引理, 将上式化为如下等价形式

$$\begin{bmatrix} \sqrt{p}\bar{A}_0^T & \sqrt{1-p}\bar{A}_1^T \end{bmatrix} \begin{bmatrix} P & 0 \\ 0 & P \end{bmatrix} \begin{bmatrix} \sqrt{p}\bar{A}_0 \\ \sqrt{1-p}\bar{A}_1 \end{bmatrix} - P < 0 \quad (26)$$

上式运用shur补引理得

$$\begin{bmatrix} -P & \begin{bmatrix} \sqrt{p}\bar{A}_0^T & \sqrt{1-p}\bar{A}_1^T \end{bmatrix} \\ \begin{bmatrix} \sqrt{p}\bar{A}_0 \\ \sqrt{1-p}\bar{A}_1 \end{bmatrix} & -\begin{bmatrix} P & 0 \\ 0 & P \end{bmatrix}^{-1} \end{bmatrix} < 0 \quad (27)$$

即

$$\begin{bmatrix} -P & \sqrt{p}\bar{A}_0^T & \sqrt{1-p}\bar{A}_1^T \\ \sqrt{p}\bar{A}_0 & -P^{-1} & 0 \\ \sqrt{1-p}\bar{A}_1 & 0 & -P^{-1} \end{bmatrix} < 0 \quad (28)$$

为了使用matlab中的LMI toolbox对该不等式进行求解，还需将上式化为LMI 形式。为了将 P 和 P^{-1} 统一起来，将上式同时左乘右乘 $\text{diag}(I, P, P)$ ，可得

$$\begin{bmatrix} -P & \sqrt{p}\bar{A}_0^T P & \sqrt{1-p}\bar{A}_1^T P \\ \sqrt{p}P\bar{A}_0 & -P & 0 \\ \sqrt{1-p}P\bar{A}_1 & 0 & -P \end{bmatrix} < 0 \quad (29)$$

可以看到这个式子和之前求取两套控制参数时使用的条件是完全等价的。然而在进一步的推导中，我们未能成功实现将上式转化为LMI形式，可能是因为思维比较狭窄，没有想到特定的解决办法，也可能是数学理论知识比较缺乏，对双线性矩阵不等式转化LMI的方法理解不够透彻，下面给出我们考虑的思路：

1、如果在一开始运用Markovian线性跳变系统的稳定性条件时，将不等式进行放缩，从一个不等式转化为三个更“紧”的不等式，即

$$\begin{cases} P > 0 \\ p\bar{A}_0^T P \bar{A}_0 < 0 \\ (1-p)\bar{A}_1^T P \bar{A}_1 < 0 \end{cases}$$

但是实际上，我们担心不等式放缩程度太过了，LMI求解时会无解。

2、用Markovian线性跳变系统的稳定性条件求解LMI时，由于 P 只要存在就可以，所以我们可以用试凑的方法，例如规定矩阵 P 为对角阵，我们没有尝试过规定 P 为单位阵的结果，这种情况下应该放缩过头了。当 P 为对角阵时，若定义 P 为分块矩阵，则有

$$P = \begin{bmatrix} P_1 & P_2 \\ P_3 & P_4 \end{bmatrix} \quad (30)$$

其中，由于 P 为对角阵，即除了对角线外都是零，所以有 $P_2 = P_3^T = 0$ ，由shur补引理得到的不等式可以化为如下形式

$$\begin{bmatrix} -\begin{bmatrix} P_1 & 0 \\ 0 & P_4 \end{bmatrix} & * & * \\ \sqrt{p} \begin{bmatrix} P_1 \bar{A}_0 & P_1 \bar{B}_0 C_c \\ P_4 B_c \bar{C}_0 & P_4 A_c \end{bmatrix} & -\begin{bmatrix} P_1 & 0 \\ 0 & P_4 \end{bmatrix} & * \\ \sqrt{1-p} \begin{bmatrix} P_1 \bar{A}_1 & P_1 \bar{B}_1 C_c \\ P_4 B_c \bar{C}_1 & P_4 A_c \end{bmatrix} & \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} & -\begin{bmatrix} P_1 & 0 \\ 0 & P_4 \end{bmatrix} \end{bmatrix} < 0 \quad (31)$$

可以看到在简化 P 之后，不等式左边矩阵的第二行第一列和第三行第一列的块矩阵少了几个求和项，在经过多次尝试化简后依然没有得到令人满意的LMI的形式。

4.2 实验曲线图

将上述计算所得的控制器参数输入到控制节点，我们可以得到各变量随时间变化的曲线图如下：

图中可以看出，丢包率为0.05时系统能较好的趋向稳定，当丢包率增大到0.12时系统趋于临界状态，有波动和发散的风险，故丢包率微小的改变可能对系统稳定性就会有很大影响。

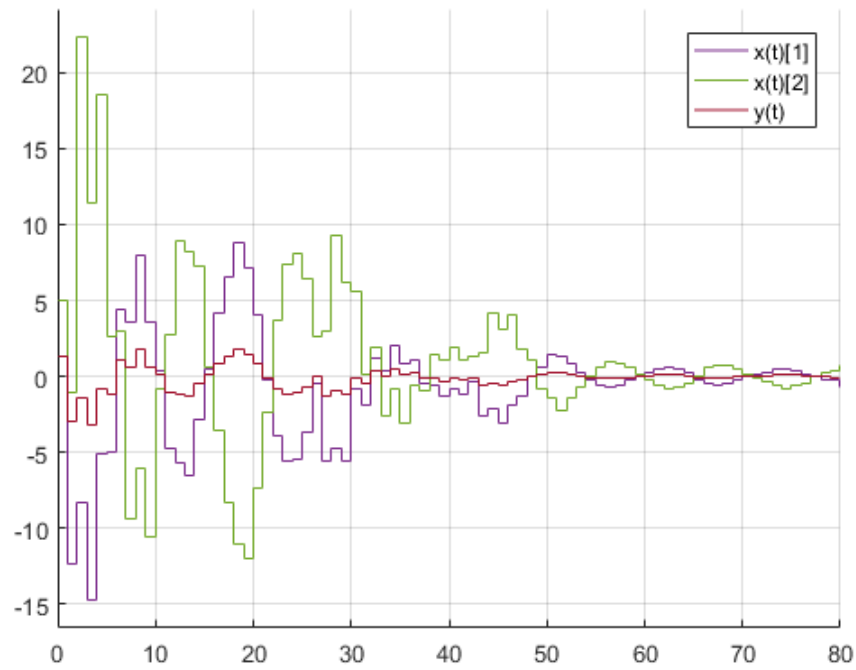


Figure 7: 丢包率为0.05时系统状态变量 $x(k)$, 输出变量 $y(k)$

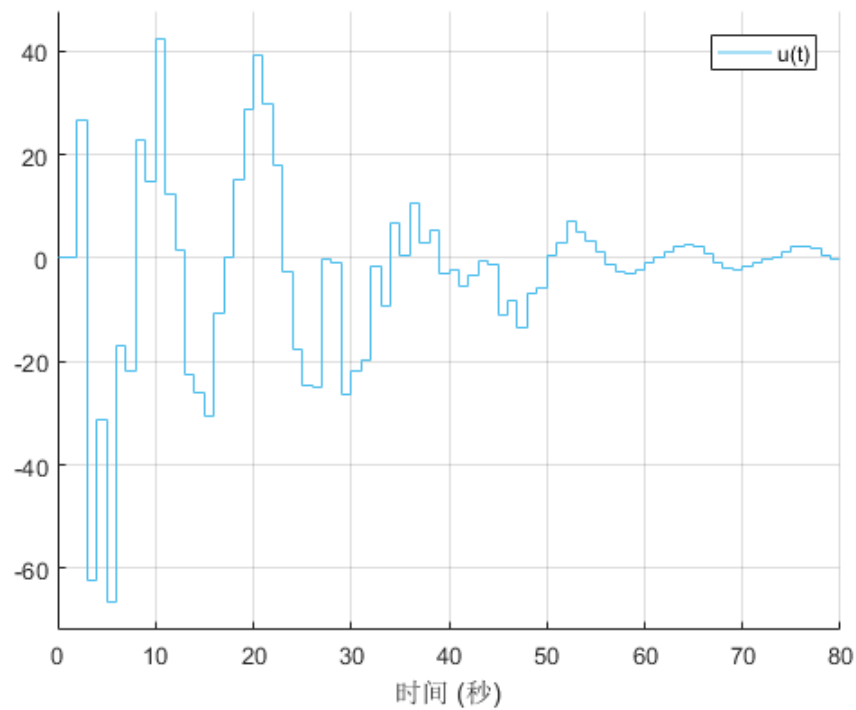


Figure 8: 丢包率为0.05时系统控制输入 $u(k)$

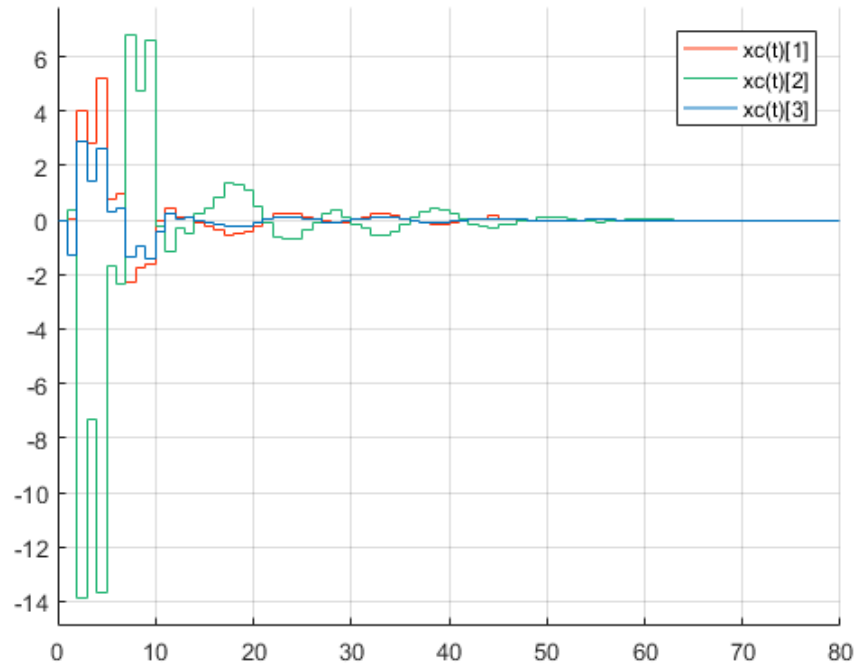


Figure 9: 丢包率为0.05时动态输出反馈控制器状态 $x_c(k)$

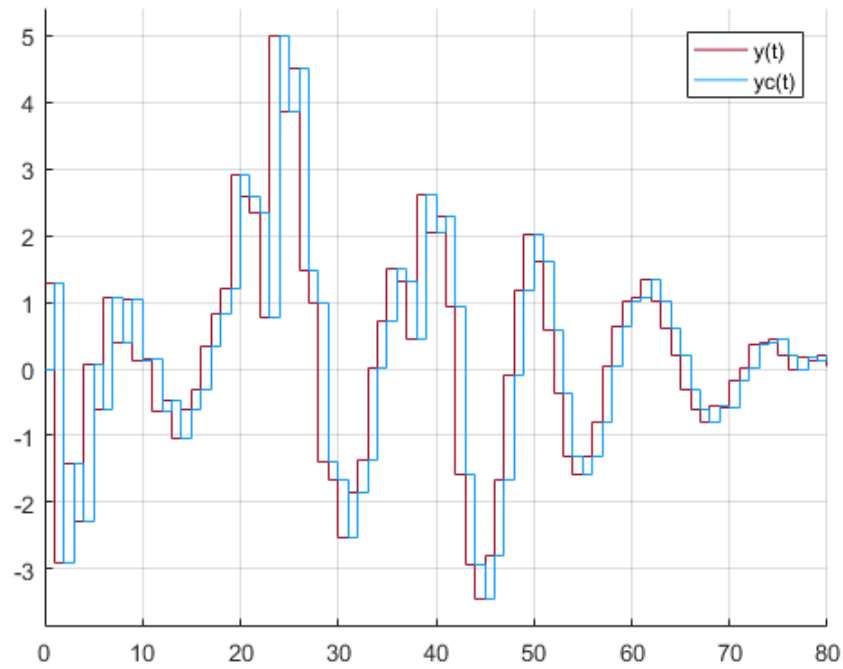


Figure 10: 丢包率为0.05时随机丢包概率条件下 $y_c(k)$ 与系统输出变量 $y(k)$

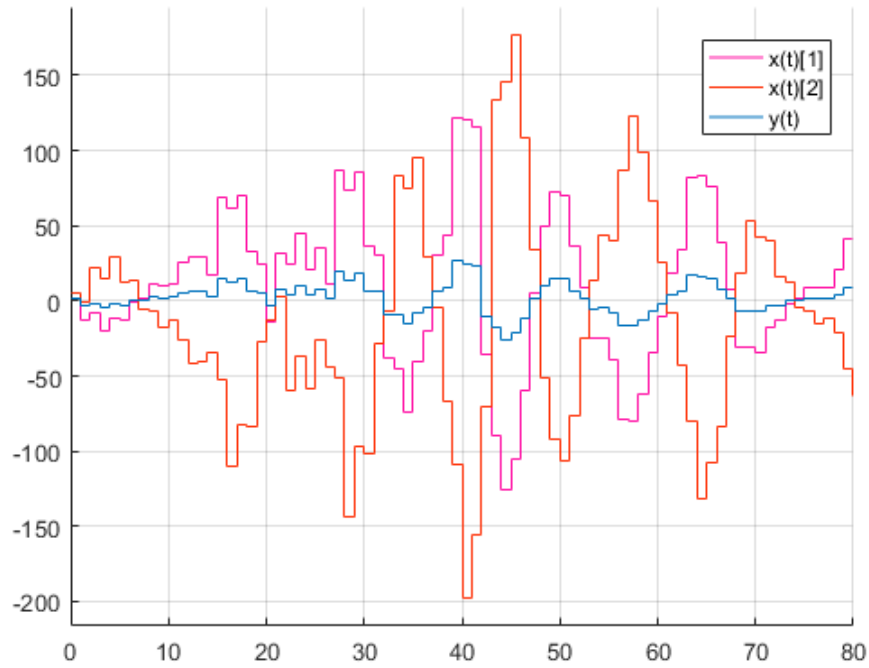


Figure 11: 丢包率为0.12时系统状态变量 $x(k)$, 输出变量 $y(k)$

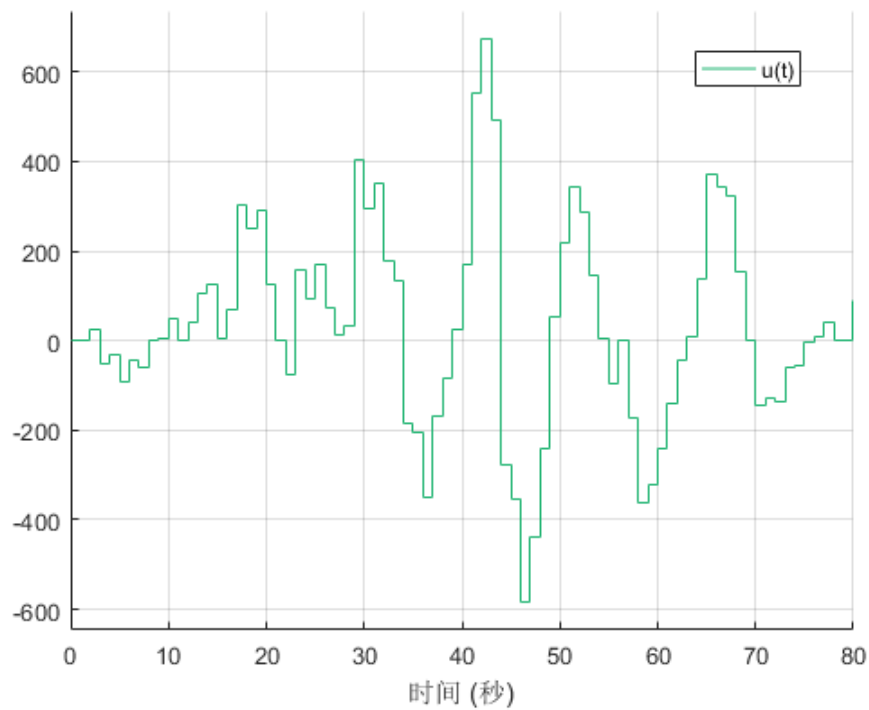


Figure 12: 丢包率为0.12时系统控制输入 $u(k)$

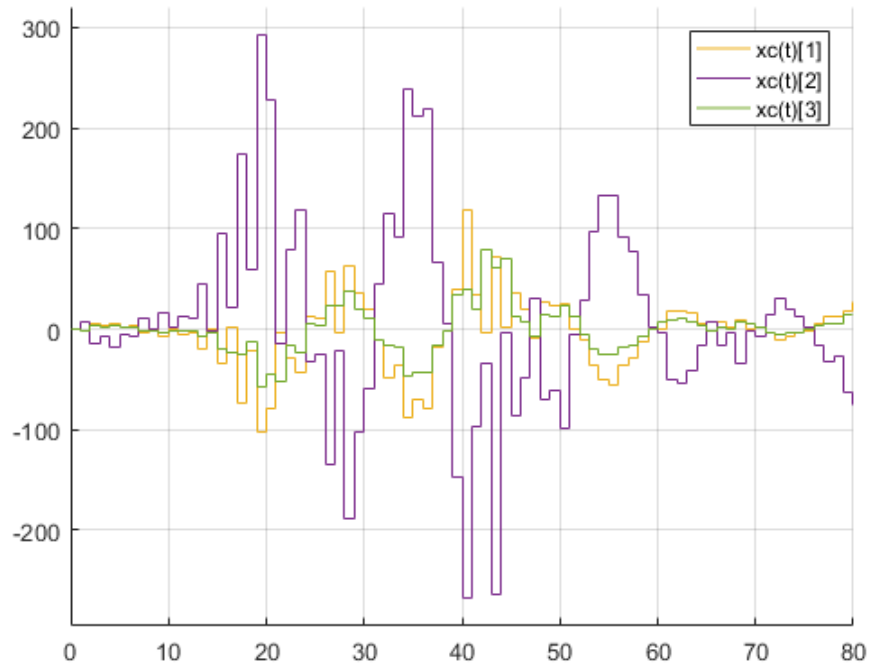


Figure 13: 丢包率为0.12时动态输出反馈控制器状态 $x_c(k)$

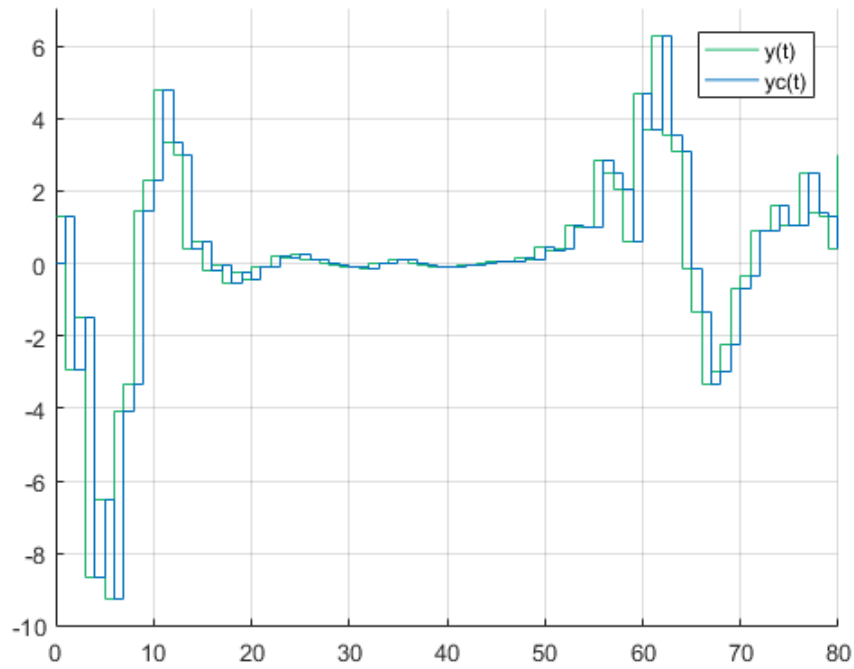


Figure 14: 丢包率为0.12时随机丢包概率条件下 $y_c(k)$ 与系统输出变量 $y(k)$

大影响, 在本项目的被控对象状态空间方程中, 通过LMI分析可得, 0.121 的丢包率是系统的临界稳定状态。

致谢

感谢陈彩莲老师和杨博老师这一学期高水平和高难度的教学, 网络系统与控制这门课的内容和课程项目都非常有挑战性, 在设计控制器的过程中解矩阵解到自闭:-)可能是因为现代控制理论的空白, 导致了课程中有些内容的理解会有些生硬和吃力, 希望下一届能够与现代控制理论一起修读。课程中我们也收获了很多, 对于工业网络方向有了一个大体的了解, 老师的课确实激起了我们对于控制, 工业互联网的一些思考, 和对IT 和OT 融合的极大兴趣, 在将来的职业选择中, 给了我们很多启发。同时, 感谢助教这一学期认真负责的答疑和帮助。

References

1. 王岩,孙增圻.网络控制系统分析与设计[M].北京:清华大学出版社,2009:92-95
2. P. Seiler and R. Sengupta, "Analysis of communication losses in vehicle control problems," Proceedings of the 2001 American Control Conference. (Cat. No.01CH37148), Arlington, VA, USA, 2001, pp. 1491-1496 vol.2.
3. S. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan. Linear Matrix Inequalities in System and Control Theory, volume 15 of Studies in Applied Mathematics. SIAM, Philadelphia, PA, 1994.
4. O.L.V. Costa and M. D. Fragoso. Stability results for discrete-time linear systems with markovian jumping parameters. Journal of Mathematical Analysis and Applications, 179:154-178, 1993.
5. 李凤霞. 不确定随机Markov跳跃系统的稳定性与镇定控制[D].中国石油大学（华东）,2013.