北京郵電大學

《智能控制》

实验报告

学院：自动化学院

专业：自动化专业

班级：2016211404

姓名：张晓燕　张晓媛

学号：2016211783

2016211780

2019 年 1 月 9 日

目 录

实验一 模糊控制在角度随动系统中的应用

一、 实验目的与意义

学习 Matlab 中建立模糊控制器的方法;了解模糊控制在角度随动系统中的应用。

二、 实验环境

PC 平台:Matlab、Ubuntu 交叉编译环境;

目标机:友善之臂 MINI2440

三、 实验内容

在 Matlab 中建立模糊控制器，将生成的模糊规则表插入程序代码中，交叉编译代码，下载到目标版中进行测试。

(1) Matlab 文本模式建立模糊控制器。

(2) 利用 Matlab 模糊逻辑工具箱建立模糊控制器。

(3) 模糊控制器 Simulink 仿真。

(4) 嵌入式程序交叉编译。

四、 实验内容及步骤

1. Matlab 文本模式建立模糊控制器

(1) 打开 Matlab 软件。

(2) 创建一个 FIS (Fuzzy Inference System) 对象：

```
>> a=newfis('fuzzy');
```

(3) 增加模糊语言变量：

```
>> a=addvar(a,'input','e',[-48 48]);
```

(4) 增加模糊语言名称，即模糊集合。

(5) 增加控制规则，即模糊推理的规则。

(6) 给定输入，得到输出，即进行模糊推理。

(7) 运行 matlab 脚本，在 Matlab 界面显示结果为：

得到的模糊矩阵为：

Ulist =

15	13	10	7	5	3	2
13	11	8	5	3	1	0
10	8	6	3	1	-1	-3
6	4	3	1	-2	-3	-5

4	2	0	-2	-5	-7	-9
1	0	-2	-4	-7	-10	-12
-1	-2	-4	-6	-9	-12	-14

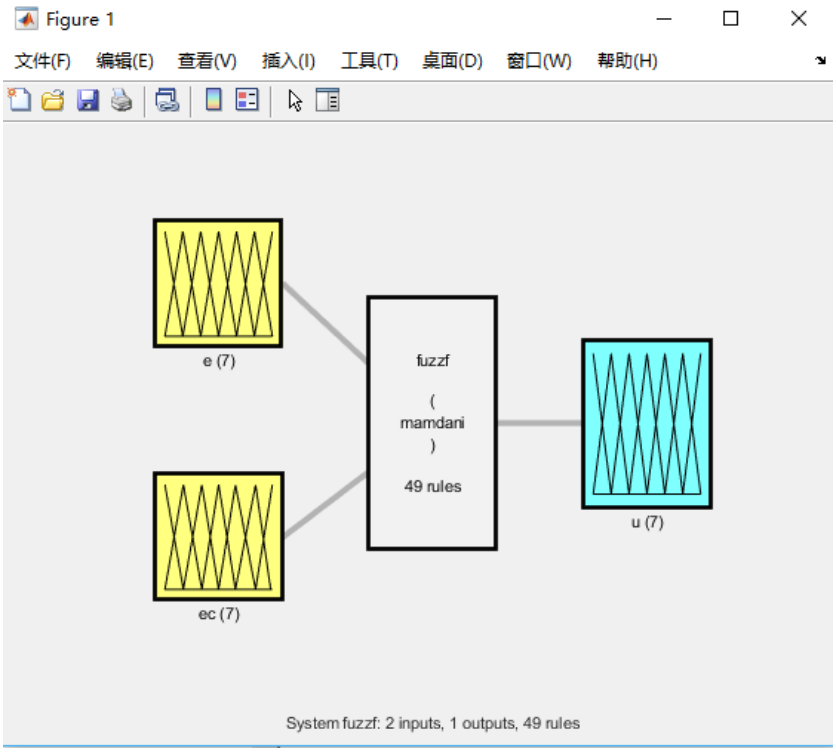


图 1 实验一图

建立的隶属度函数如下图所示：

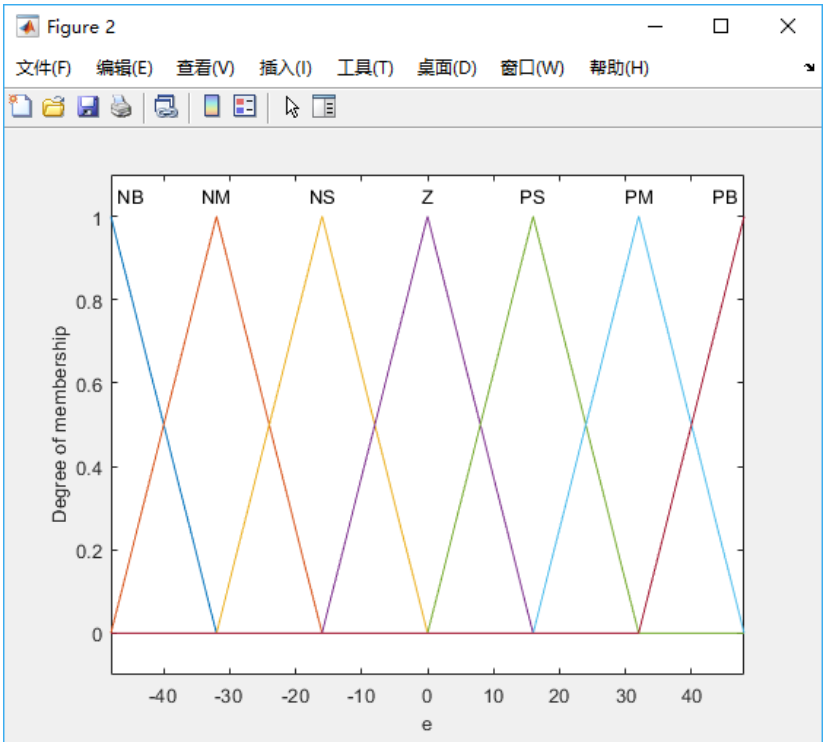


图 2 误差 e 的隶属度函数

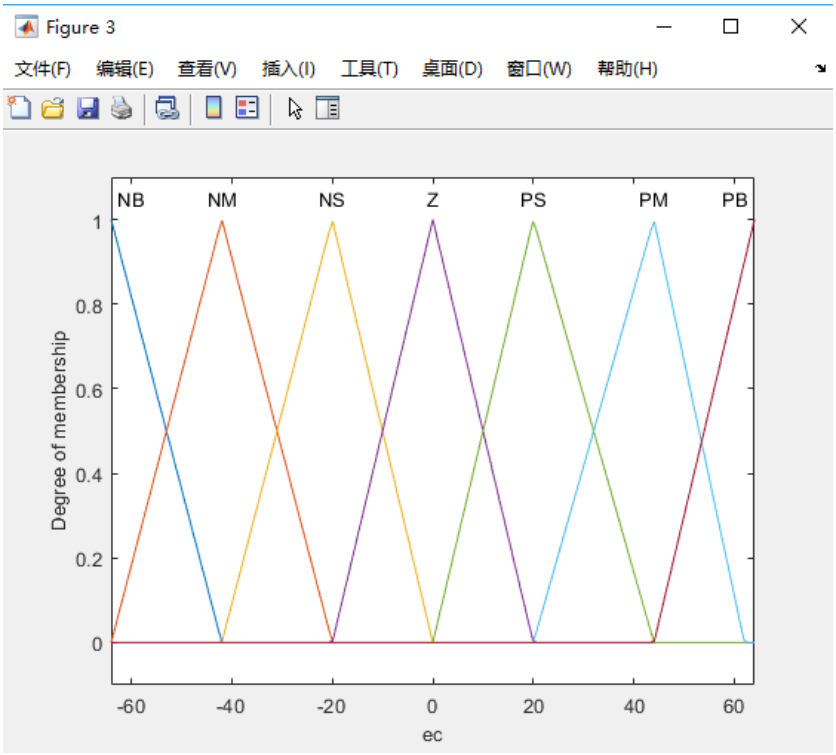


图 3 误差变化率 ec 的隶属度函数

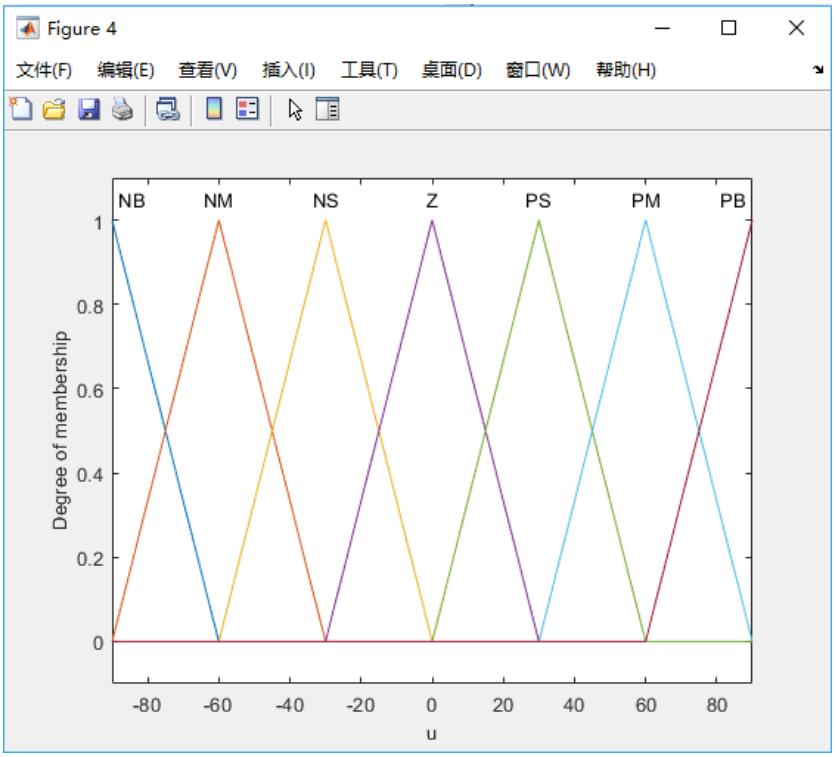


图 4 误差变化率 u 的隶属度函数

2. 利用 Matlab 模糊逻辑工具箱建立模糊控制器步骤

(1) 在 matlab 工作窗口输入:fuzzy+回车进入图形界面编辑，如下图所示：

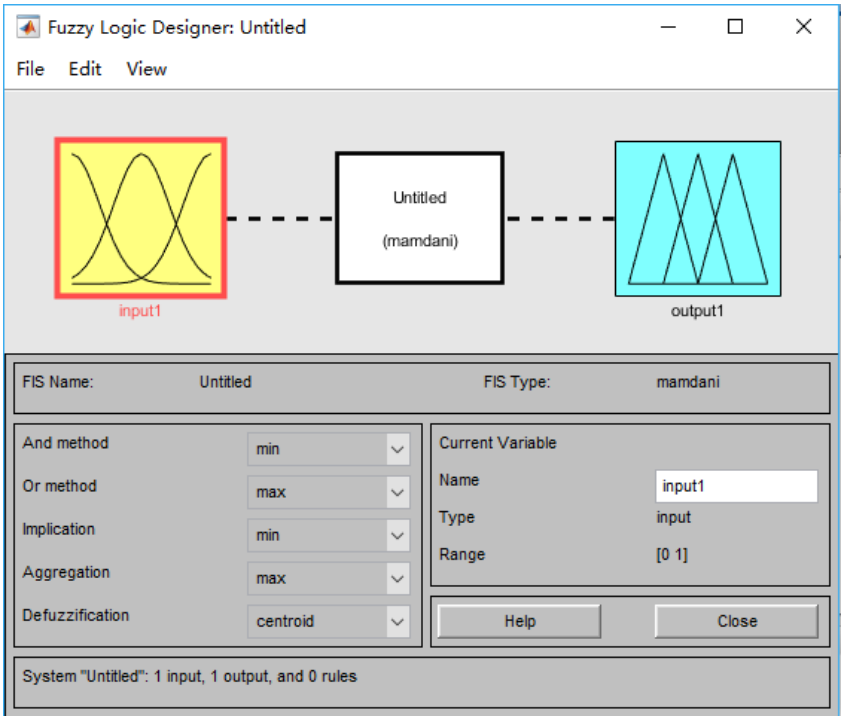


图 5 界面编辑

(2) 点击工具栏 Edit - -Add Variable... - -Input，添加两个输入模糊变量，一个输出模糊变量。

(3) 分别选中 input1、input2、output1,在其 Name 选项中将其名称改变为，e、ec、u，如下图所示。

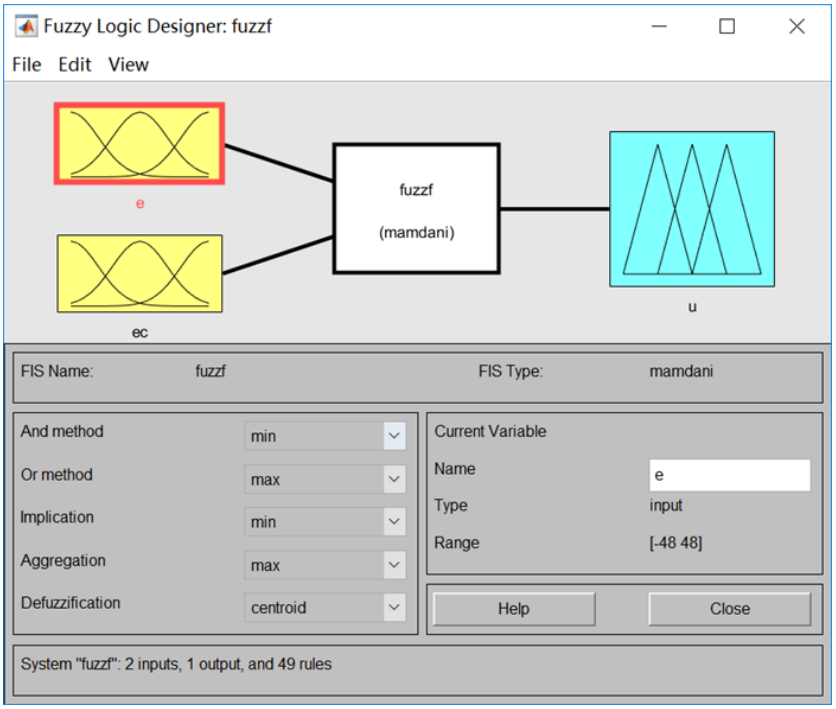


图 6 添加模糊变量

(4) 双击图标，打开隶属度函数编辑器，分别编辑 e、ec、u 的隶属度函数。

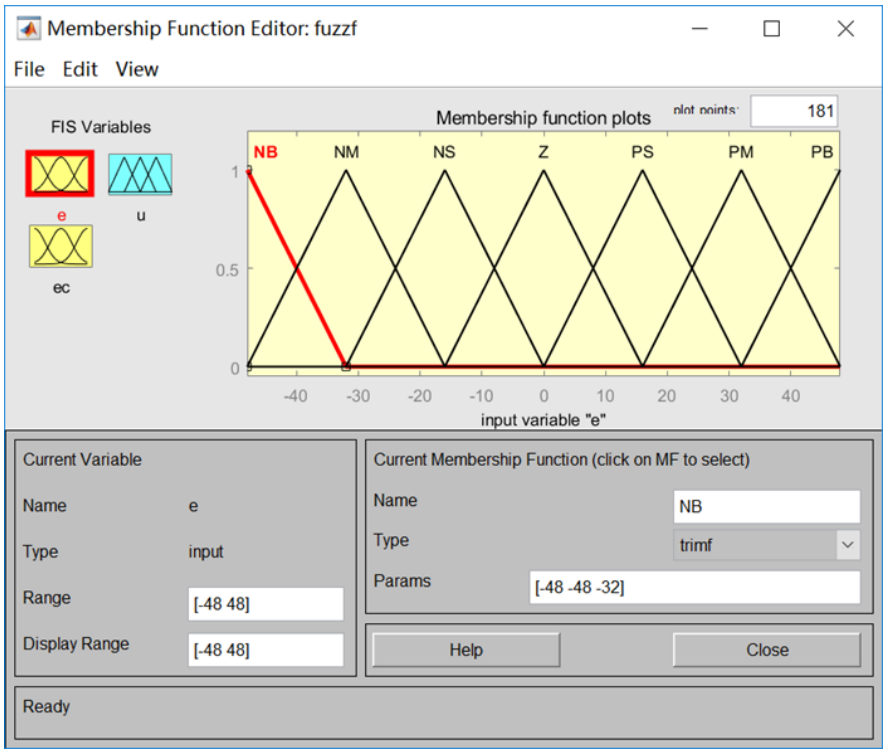


图 7 e 的隶属度函数曲线

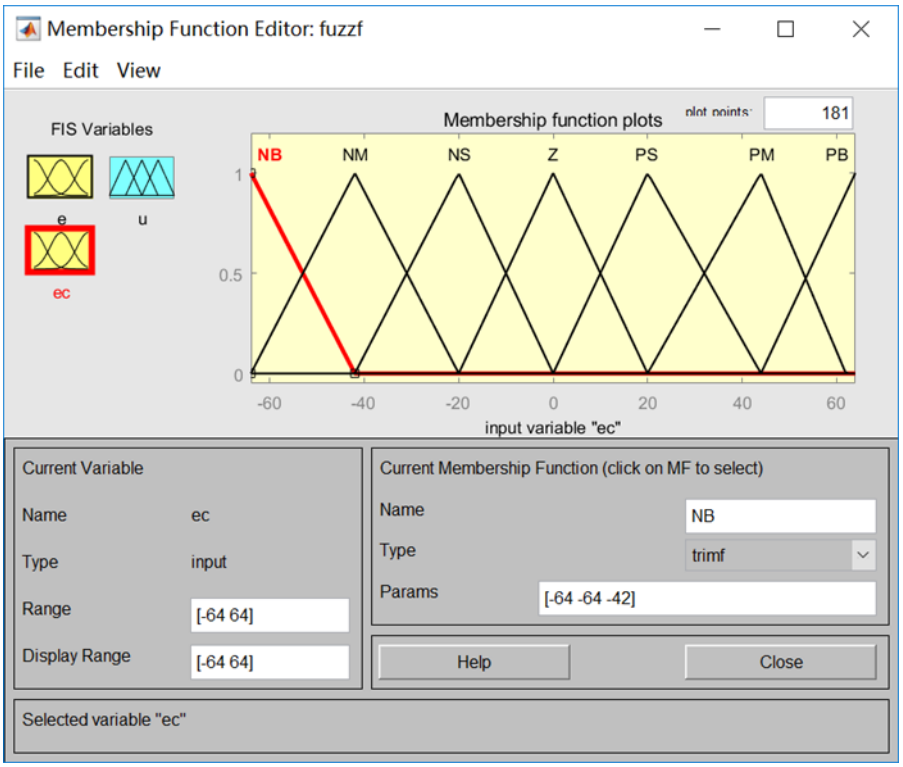


图 8 ec 的隶属度函数曲线

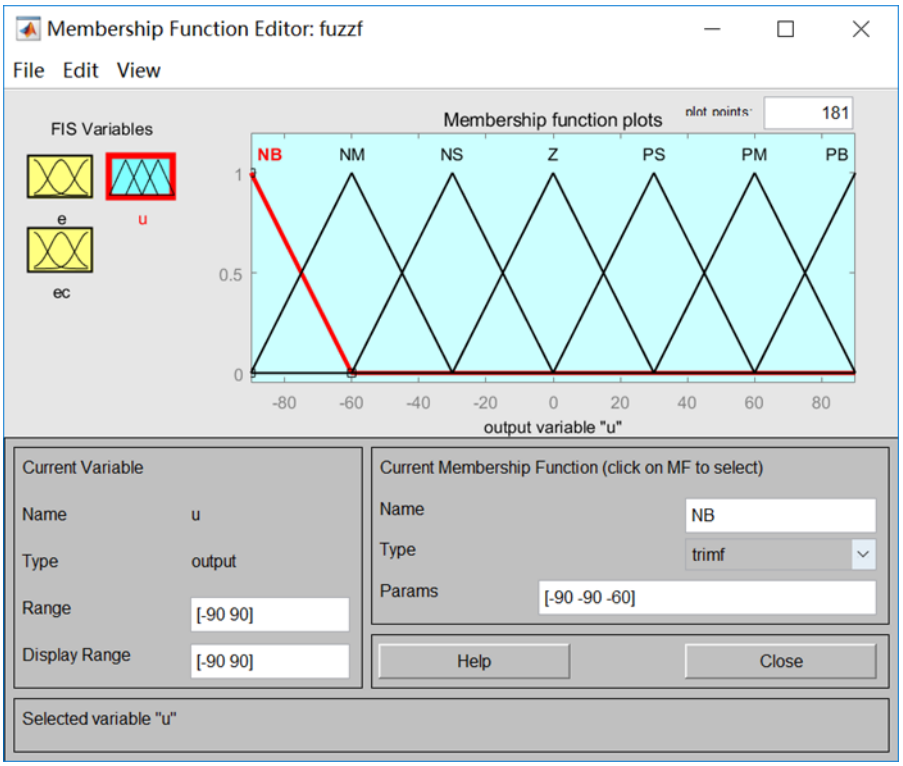


图 9 u 的隶属度函数曲线

(5) 点击 Edit 菜单，选 Rules. 选中 e, ec, u,中隶属度函数。and 选项，权重 Weight 均设为 1，然后点击 Add rule 添加规则，同理添加其他规则。

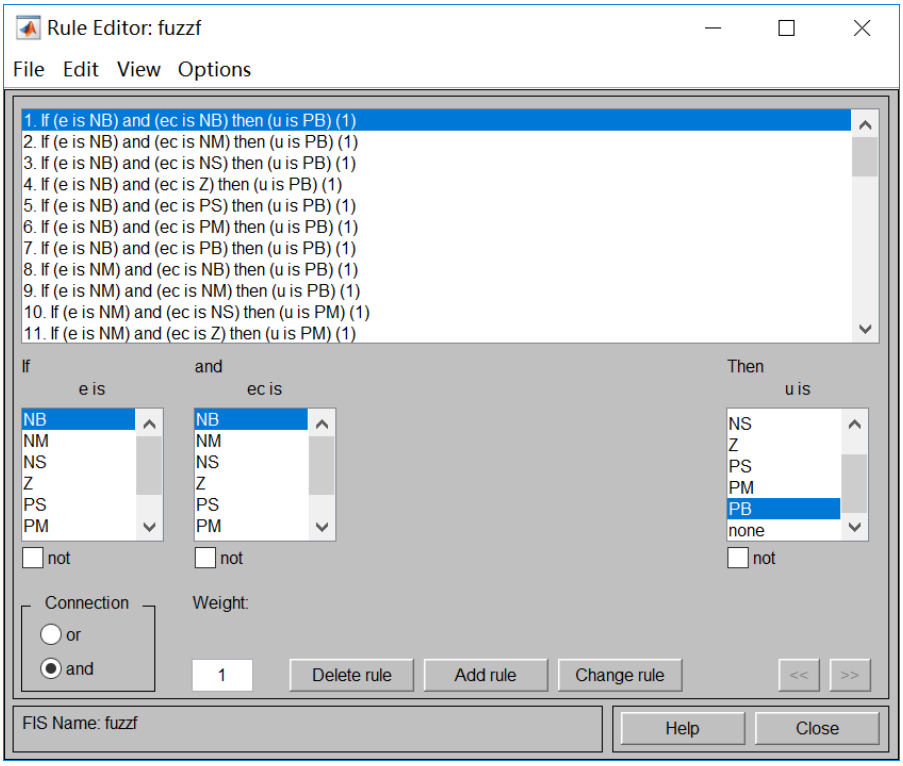


图 10 添加规则

(6) 保存文件。

(7) 输出到 Workspace 中，名为 a2，打开 Rule Viewer 如下图：

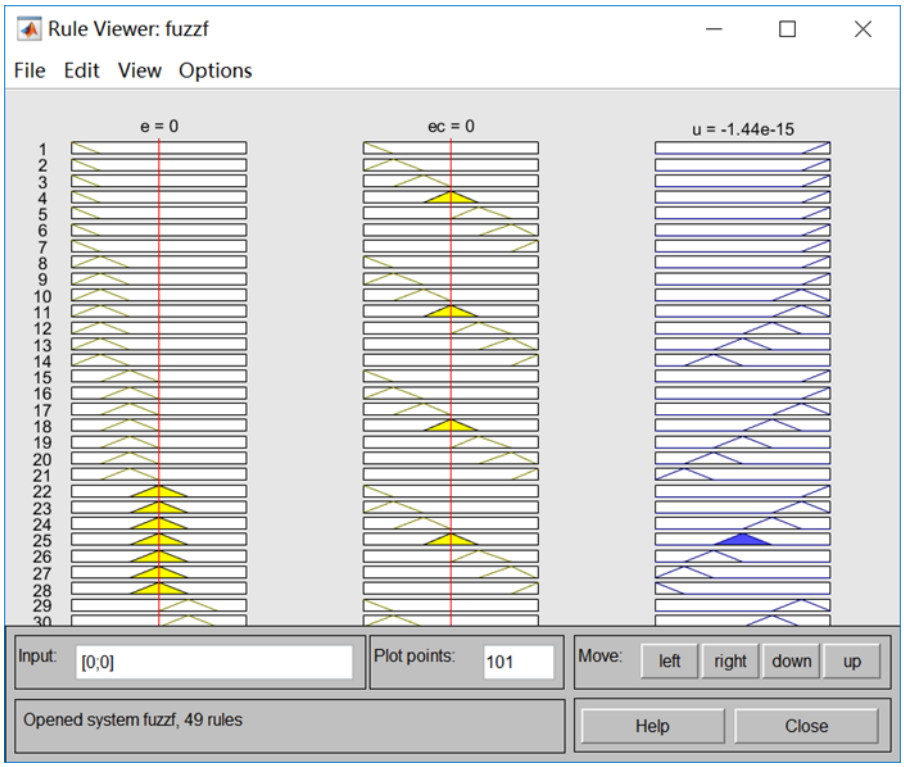


图 11 Rule Viewer

(8) View - >Surface 打开 Surface Viewer 表面如下图：

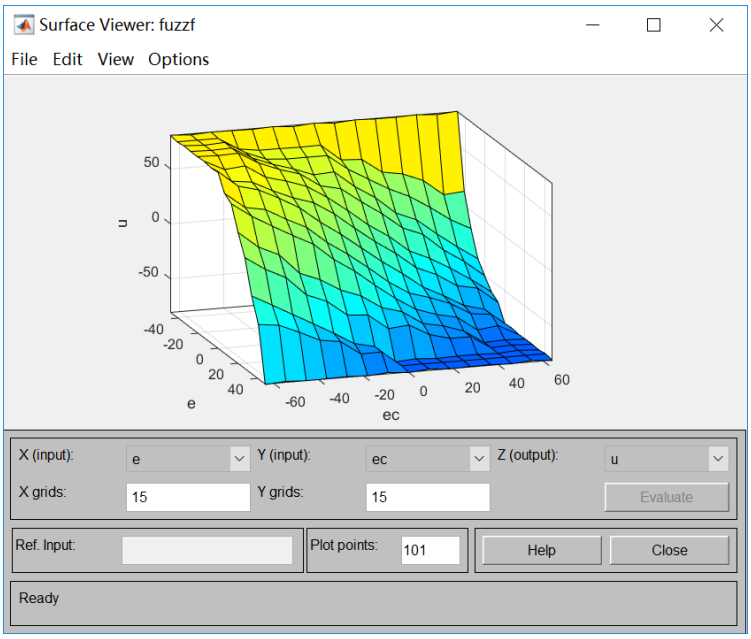


图 12 Surface Viewer

MATLAB 代码：

```
%Fuzzy Controller Design
clear all;
close all;

a=newfis('fuzzf');

%Ê äÈ ëe £¬ ·¶Î§[-48,48],7 ¸öÄ £ºÓ ïÑÔ £¬NB,NM,NS,Z,PS,PM,PB
a=addvar(a,'input','e',[-48 48]);                    %Parameter e
a=addmf(a,'input',1,'NB','trimf',[-48 -48 -32]);
a=addmf(a,'input',1,'NM','trimf',[-48 -32 -16]);
a=addmf(a,'input',1,'NS','trimf',[-32 -16 0]);
a=addmf(a,'input',1,'Z','trimf',[-16 0 16]);
a=addmf(a,'input',1,'PS','trimf',[0 16 32]);
a=addmf(a,'input',1,'PM','trimf',[16 32 48]);
a=addmf(a,'input',1,'PB','trimf',[32 48 48]);

%Ê äÈ ëec £¬ ·¶Î§[-64,64],7 ¸öÄ £ºÓ ïÑÔ £¬NB,NM,NS,Z,PS,PM,PB
a=addvar(a,'input','ec',[-64 64]);                   %Parameter ec
a=addmf(a,'input',2,'NB','trimf',[-64 -64 -42]);
a=addmf(a,'input',2,'NM','trimf',[-64 -42 -20]);
a=addmf(a,'input',2,'NS','trimf',[-42 -20 0]);
a=addmf(a,'input',2,'Z','trimf',[-20 0 20]);
a=addmf(a,'input',2,'PS','trimf',[0 20 44]);
a=addmf(a,'input',2,'PM','trimf',[20 44 62]);
a=addmf(a,'input',2,'PB','trimf',[44 64 64]);


%Ê ä³öu £¬ ·¶Î§[-90,90] £¬7 ¸öÄ £ºÓ ïÑÔ £¬NB,NM,NS,Z,PS,PM,PB
a=addvar(a,'output','u',[-90 90]);                   %Parameter u
a=addmf(a,'output',1,'NB','trimf',[-90 -90 -60]);
a=addmf(a,'output',1,'NM','trimf',[-90 -60 -30]);
a=addmf(a,'output',1,'NS','trimf',[-60 -30 0]);
a=addmf(a,'output',1,'Z','trimf',[-30 0 30]);
a=addmf(a,'output',1,'PS','trimf',[0 30 60]);
a=addmf(a,'output',1,'PM','trimf',[30 60 90]);
a=addmf(a,'output',1,'PB','trimf',[60 90 90]);


%Ä £ºý¼æÔ ò± í£¬7*7 = 49ÐÐ £¬5ÁÐ
rulelist=[ 1 1 7 1 1;
           1 2 7 1 1;
           1 3 7 1 1;
           1 4 7 1 1;
```

1 5 7 1 1;
1 6 7 1 1;
1 7 7 1 1;

2 1 7 1 1;
2 2 7 1 1;
2 3 6 1 1;
2 4 6 1 1;
2 5 5 1 1;
2 6 4 1 1;
2 7 3 1 1;

3 1 7 1 1;
3 2 7 1 1;
3 3 6 1 1;
3 4 5 1 1;
3 5 4 1 1;
3 6 3 1 1;
3 7 2 1 1;

4 1 7 1 1;
4 2 6 1 1;
4 3 5 1 1;
4 4 4 1 1;
4 5 3 1 1;
4 6 2 1 1;
4 7 1 1 1;

5 1 6 1 1;
5 2 5 1 1;
5 3 4 1 1;
5 4 3 1 1;
5 5 2 1 1;
5 6 1 1 1;
5 7 1 1 1;

6 1 4 1 1;
6 2 3 1 1;
6 3 2 1 1;
6 4 1 1 1;
6 5 1 1 1;
6 6 1 1 1;
6 7 1 1 1;

```
        7 1 1 1 1;
        7 2 1 1 1;
        7 3 1 1 1;
        7 4 1 1 1;
        7 5 1 1 1;
        7 6 1 1 1;
        7 7 1 1 1];

a=addrule(a,rulelist);
%showrule(a)                                % Show fuzzy rule base

a1=setfis(a,'DefuzzMethod','centroid');    % Defuzzy
writefis(a1,'fuzzf');                       % save to fuzzy file "fuzz.fis" which can be
                                                % simulated with fuzzy tool
a2=readfis('fuzzf');
disp('------------------------------------------------------');
disp('        fuzzy controller table:e=[-48,+48],ec=[-64,+64]          ');
disp('------------------------------------------------------');

Ulist=zeros(7,7);

for i=1:7
    for j=1:7
        e(i)=-4+i;
        ec(j)=-4+j;
        Ulist(i,j)=evalfis([e(i),ec(j)],a2);
    end
end

Ulist=ceil(Ulist)

figure(1);
plotfis(a2);
figure(2);
plotmf(a,'input',1);
figure(3);
plotmf(a,'input',2);
figure(4);
plotmf(a,'output',1);
```

3. 模糊控制器 Simulink 仿真

(1) 打开 simulink 文件，运行。

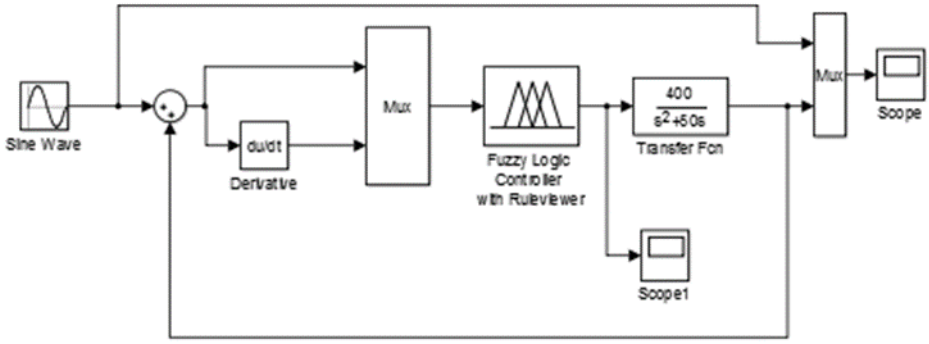


图 13 simulink 仿真文件

(2) 可以看到仿真结果如图所示。

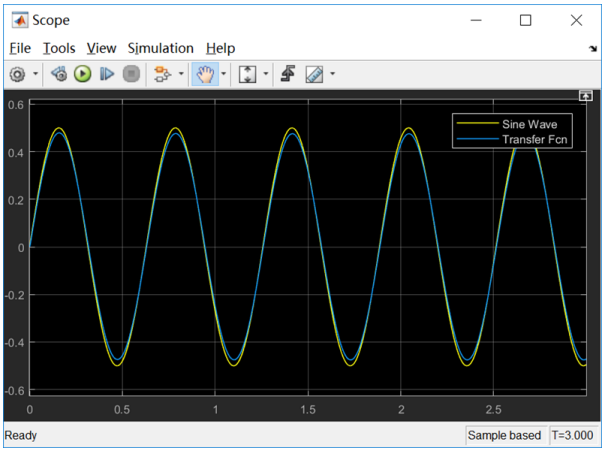


图 14 simulink 仿真结果

4. 交叉编译

1) 打开角度随动控制代码

打开Vmware 虚拟机软件，运行Ubuntu 虚拟机，登陆系统，用户名robot, 密码123456。点击主界面工具栏Places--Home Folder 打开用户文件夹，新建一个文件夹，例如“fuzzy”，使用鼠标拖放或者Ctrl+C，将Windows 中的程序复制到新建的文件夹内。

打开gedit 编辑器：主界面工具栏Applications--Accessories--geditText Editor。从gedit 中打开上面各个源代码。将第二步生成的Ulist 填到fuzzy.c 第16 行的规则表rule 中并保存。rule是一个7*7 的二维数组。

2) 交叉编译程序

打开终端：主界面工具栏Applications--Accessories--Terminal.终端中$提示符表示以普通用户执行命令，#提示符表示以root 用户（类似于Windows 系统的管理员账户Administrator，具有很大的权限）执行命令。

3) 编译成功后，将可执行程序重命名为学号后三位（下面000 处应为学号后三位），并将可执行程序复制到用户桌面。
4) 使用Ctrl+C 或者鼠标将桌面可执行程序拖到Windows 下的桌面。
5) 设置超级终端。
6) 运行程序

(1) 下载文件到开发板

超级终端窗口中，点鼠标右键，在弹出的菜单中选择“发送文件”，设置好要发送的文件和使用的协议，开始向开发板发送文件。

(2) 在超级终端中运行程序(000 为可执行程序名，应为学号后三位)，chmod +x 000 表示为文件增加可执行权限，./000 表示执行当前目录下的000程序。

(3) 运行程序，此时可以观察从动电机是否跟随主动电机做出相应的动作。

五、 实验总结

在多次测试改变隶属度函数以后，模糊控制系统可以得到很好的仿真结果。采用不同的模糊规则，会有不同的仿真效果，系统的随动效果不同，经过变更规则修正因子，改变了误差和误差变化率所占权重对规则的影响，得到不同的规则，从而可以观察在不同规则下系统输出。

通过此次模糊控制在角度随动系统中的应用实验，逐步学习了在 MATLAB 中建立模糊控制器的方法，对模糊规则的建立有了一定了解，知道了模糊语言在实际应用中的实现过程，了解了模糊控制在角度随动系统中的应用。

实验二 神经网络在角度随动系统中的应用

一、 实验目的与意义

学习 Matlab 中建立单神经元自适应神经网络控制器的方法;了解神经网络在角度随动系统中的应用。

二、 实验环境

PC 平台:Matlab、Ubuntu 交叉编译环境;

目标机:友善之臂 MINI2440

三、 实验内容

(1) 根据直流伺服电机的传递函数求出差分方程

(2) 在 Matlab 中建立单神经元自适应控制器

(3) 调整学习速率和比例系数 K 得到较好的控制效果

(4) 编写单神经元自适应控制器的 c 程序，编译并在角度随动系统中验证。

四、 实验内容及步骤

1. 直流伺服电机传递函数

$$G(s)=\frac{7.463}{0.0002202s^2+0.006s+1}$$

2. 根据直流伺服电机的传递函数求出差分方程

一、 使用 tf 函数建立传递函数:

```
sys=tf(7.463,[0.0002202,0.006,1]);
```

得到传递函数：

$$G(s)=\frac{7.463}{0.0002202s^2+0.006s+1}$$

二、 设置采样时间:

```
ts = 0.0001;
```

三、 使用 c2d 函数进行 Z 变换:

```
dsys=c2d(sys,ts,'z');
```

得到：

$$G(Z)=\frac{0.01678Z+0.01663}{Z\text{^}2-1.969Z+0.9731}$$

四、 交叉相乘得到:

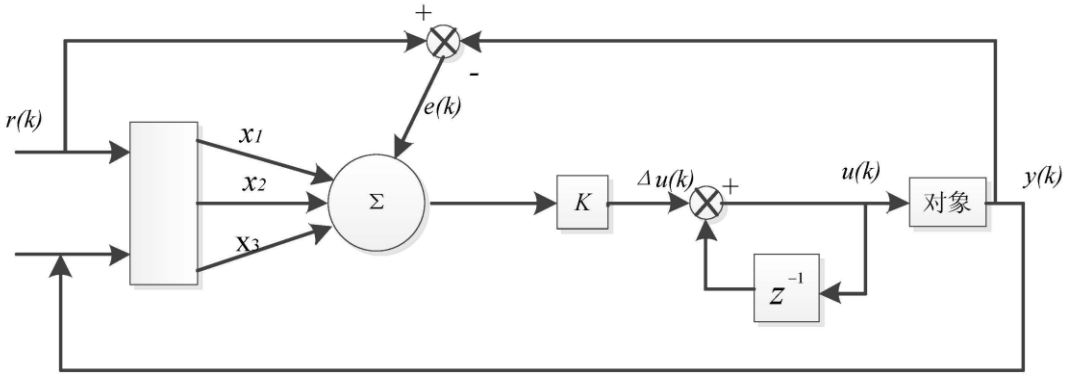
$$Y(Z)(1+a_1z^{-1}+\cdots+a_nz^{-n})=U(Z)(b_0+b_1z^{-1}+\cdots+b_mz^{-m})$$

五、 对 U(Z)和 Y(Z)进行 z 逆变换得到差分方程:

$$y(k)-1.969y(k-1)+0.973(k-2)=0.01678u(k-1)+0.01633u(k-2)$$

3. 单神经元自适应控制算法——方波

单神经元自适应控制的结构如下图所示。



单神经元自适应控制器是通过对加权系数的调整来实现自适应、自组织功能，控制算法为：

$$\mathrm{u(k)} = \mathrm{u(k-1)} + \mathrm{K}\sum_{i=1}^{3} \omega_i(k)x_i(k)$$

如果权系数的调整按有监督的 Hebb 学习规则实现，在学习算法中加入监督项 z(k)，神经网络权值学习算法为：

$$\begin{cases} w_1(k) = w_1(k-1) + \eta z(k)u(k-1)x_1(k) \\ w_2(k) = w_2(k-1) + \eta z(k)u(k-1)x_2(k) \\ w_3(k) = w_3(k-1) + \eta z(k)u(k-1)x_3(k) \end{cases}$$

式中：

$$\begin{cases} z(k) = e(k) \\ x_1(k) = e(k) \\ x_2(k) = e(k) - e(k-1) \\ x_3(k) = e(k) - 2e(k-1) + e(k-2) \end{cases}$$

η 为学习速率，K 为神经元的比例系数，K>0，η∈(0,1)。

K 值得选择非常重要。K 越大，则快速性越好，但超调量大，甚至可能使系统不稳定。当被控对象时延增大时，K 值必须减小，以保证系统稳定。K 值选择过小，会使系统快速性变差。

被控对象为 y(k)（为 Matlab 中建立的直流伺服电机的差分方程），输入指令为一方波信号：r(k)=0.5sign(sin(4π×k×ts))，采样时间 ts=0.001s，采用有监督的 Hebb 学习规则实现权值的学习，初始权值取 W=[ω1 ω2 ω3]= [0.10.10.1]，η=0.1，K =0.01。以此为条件在 Matlab 中建立单神经元自适应控制器。

在调节过程中可能会出现超调量过大的问题，导致仿真失败如下图所示。此时应该重点调节神经元的比例系数，使其超调量下降到可以接受的范围。

在仿真时，还会出现仿真精度不够的现象，如下图所示。应观察仿真图像，按照 K 与 η 的不同特点，分别调试参数大小，以期得到理想的仿真图像。

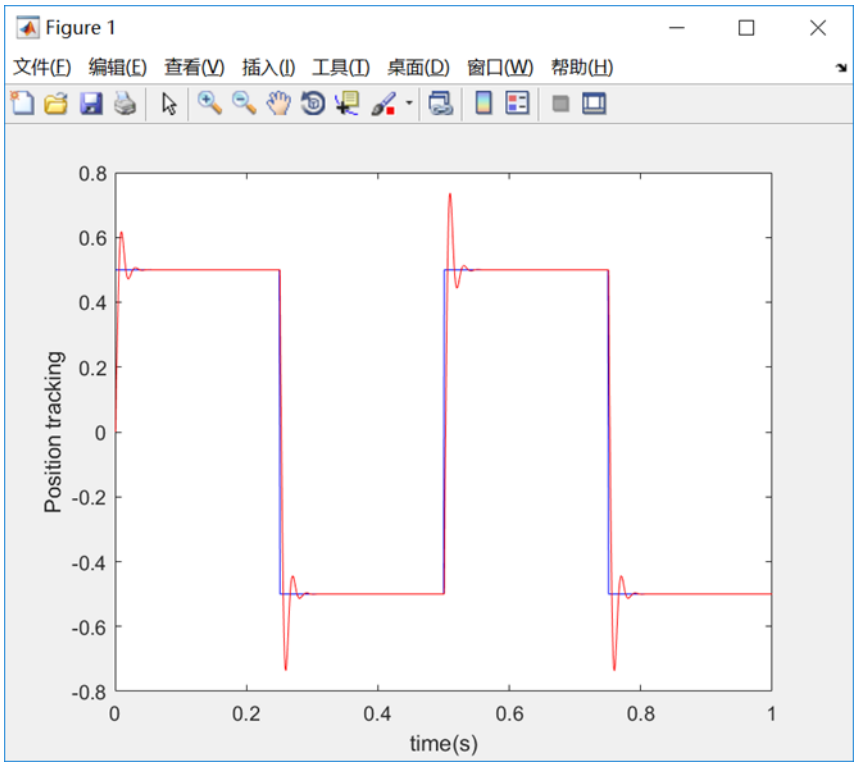


图 15 方波跟随结果

MATLAB 代码：

```
clear all;
close all;

x=[0,0,0]';
xite=0.1;
K=0.01;
ts=0.001;
w1_1=0.10; w2_1=0.10;
w3_1=0.10;

e_1=0;
e_2=0;
y_1=0;y_2=0;
  u_1=0;u_2=0;%
for k=1:1:1000
       time(k)=k*ts;
            r(k)=0.5*sign(sin(4*pi*k*ts));
       y(k)=0.8464*y_1 + 32.01*u_1+0.1064*u_2;
       e(k)=r(k)-y(k);% Îó²î

     e(k)=r(k)-y(k);
       z(k)=e(k);
       x1(k)=e(k)-e_1;
```

```
    x2(k)=e(k);
    x3(k)=e(k)-2*e_1+e_2;
    w1(k)=w1_1+xite*0.07*z(k)*u_1*x1(k);
    w2(k)=w2_1+xite*0.005*z(k)*u_1*x2(k);
    w3(k)=w3_1+xite*0.02*z(k)*u_1*x3(k);

        w1p(k) = w1(k)/(abs(w1(k)) + abs(w2(k)) + abs(w3(k)));
        w2p(k) = w2(k)/(abs(w1(k)) + abs(w2(k)) + abs(w3(k)));
        w3p(k) = w3(k)/(abs(w1(k)) + abs(w2(k)) + abs(w3(k)));



    u(k)=u_1+K*(w1p(k)*x1(k)+w2p(k)*x2(k)+w3p(k)*x3(k));
    y_2=y_1;
    y_1=y(k);
    e_2=e_1;
    e_1=e(k);
    w1_1=w1(k);
    w2_1=w2(k);
    w3_1=w3(k);
    u_2=u_1;
    u_1=u(k);
end
figure(1);
plot(time,r,'b',time,y,'r');
xlabel('time(s)');ylabel('Position tracking');
figure(2);
plot(time,e,'r');
xlabel('time(s)');ylabel('error');
figure(3);
plot(time,w1,'r');
xlabel('time(s)');ylabel('w1');
figure(4);
plot(time,w2,'r');
xlabel('time(s)');ylabel('w2');
figure(5);
plot(time,w3,'r');
xlabel('time(s)');ylabel('w3');
```

4. 单神经元自适应控制算法——正弦波

被控对象为 y(k)（为 Matlab 中建立的直流伺服电机的差分方程），输入指令为一方波信号：r(k)=20sin(4π×k×ts)，采样时间 ts=0.001s，采用有监督的 Hebb 学

习规则实现权值的学习，初始权值取 W=[ω1 ω2 ω3]= [0.10.10.1] ，η=0.1，K =0.01。以此为条件在 Matlab 中建立单神经元自适应控制器。

在调节过程中可能会出现超调量过大的问题。此时应该重点调节神经元的比例系数，使其超调量下降到可以接受的范围。在仿真时，还会出现仿真精度不够的现象。应观察仿真图像，按照 K 与 η 的不同特点，分别调试参数大小，以期得到理想的仿真图像。当上述方法都无法获得理想的仿真图像时，应该考虑增加采样数量，提高仿真精度与效果。

仿真效果如下图所示。

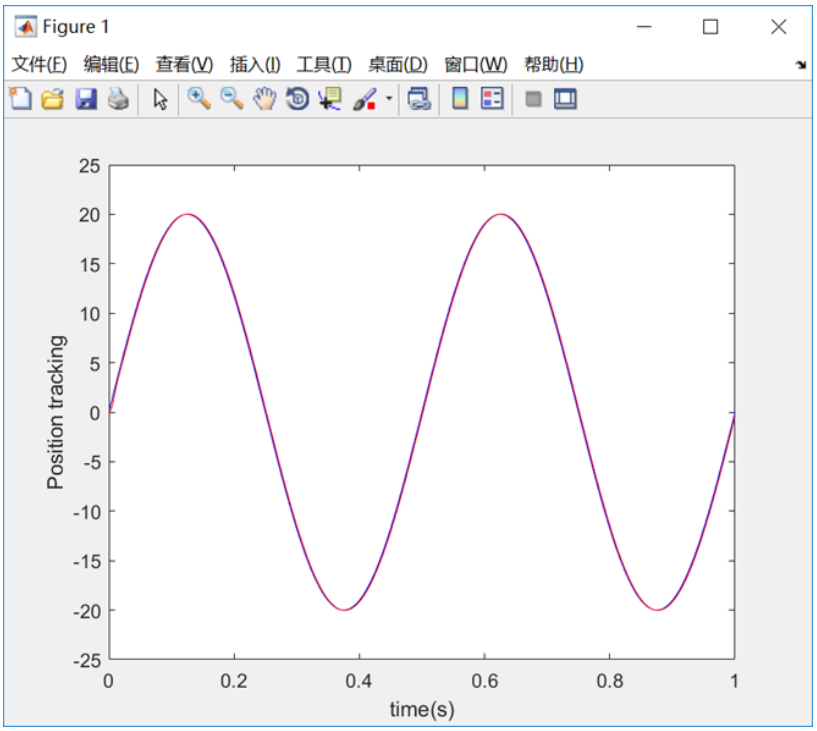


图 16 正弦波跟随结果

MATLAB 代码：

```
clear all;
close all;

x=[0,0,0]'; %x1,x2,x3µÄ ³õÊ¼Öµ
xite=0.1;      %
K=0.01;          %
ts=0.001;      %
w1_1=0.10;
w2_1=0.10;
w3_1=0.10;

e_1=0;%
```

```
e_2=0;
y_1=0;y_2=0;%
u_1=0;u_2=0;%
for k=1:1:1000
       time(k)=k*ts;
       r(k)=2*sin(4*pi*k*ts);
       y(k)=0.8464*y_1 + 32.01*u_1+0.1064*u_2;
       e(k)=r(k)-y(k);
      e(k)=r(k)-y(k);
       z(k)=e(k);
       x1(k)=e(k)-e_1;
       x2(k)=e(k);
       x3(k)=e(k)-2*e_1+e_2;
       w1(k)=w1_1+xite*0.07*z(k)*u_1*x1(k);
       w2(k)=w2_1+xite*0.005*z(k)*u_1*x2(k);
       w3(k)=w3_1+xite*0.02*z(k)*u_1*x3(k);

            w1p(k) = w1(k)/(abs(w1(k)) + abs(w2(k)) + abs(w3(k)));
            w2p(k) = w2(k)/(abs(w1(k)) + abs(w2(k)) + abs(w3(k)));
            w3p(k) = w3(k)/(abs(w1(k)) + abs(w2(k)) + abs(w3(k)));

       u(k)=u_1+K*(w1p(k)*x1(k)+w2p(k)*x2(k)+w3p(k)*x3(k));
       y_2=y_1;
       y_1=y(k);
       e_2=e_1;
       e_1=e(k);
       w1_1=w1(k);
       w2_1=w2(k);
       w3_1=w3(k);
       u_2=u_1;
       u_1=u(k);
end
figure(1);
plot(time,r,'b',time,y,'r');
xlabel('time(s)');ylabel('Position tracking');
figure(2);
plot(time,e,'r');
xlabel('time(s)');ylabel('error');
figure(3);
plot(time,w1,'r');
xlabel('time(s)');ylabel('w1');
figure(4);
plot(time,w2,'r');
xlabel('time(s)');ylabel('w2');
```

```
figure(5);
plot(time,w3,'r');
xlabel('time(s)');ylabel('w3');
```

5. 交叉编译

Neural.c 文件如下：

```
#include <stdio.h>
#include <math.h>
#include "neural.h"
/*****************************************/
float neural_input_data[3] = {0, 0, 0}; //神经网络输入值初始值为零
float xite = 0.01;     //学习率
float w1_1 = 0.10;     //k-1 时刻的初始权值
float w2_1 = 0.10;     //k-1 时刻的初始权值
float w3_1 = 0.10;     //k-1 时刻的初始权值
float e_1 = 0;         //e(k-1)
float e_2 = 0;         //e(k-2)
float y_1 = 0, y_2 = 0; //y(k-1)和  y(k-2)初始值为零
float u_1 = 0, u_2 = 0; //u(k-1)和  u(k-2)初始值为零
/*****************************************/
//单神经元自适应控制程序
//输入：  control_sys_y --  被控对象的角度
//        control_sys_r --  实际电机的角度
//输出：  output --          电机角度
float neural_adaptive_control(float control_sys_y, float control_sys_r)
{   float control_sys_e = 0;     //k 时刻的系统误差
    float control_sys_u = 0;     //k 时刻的系统控制量
    float w1=0;                  //k 时刻的神经元的第一个输入量的权值
    float w2=0;                  //k 时刻的神经元的第二个输入量的权值
    float w3=0;                  //k 时刻的神经元的第三个输入量的权值
    float K = 0.01;              //单神经元自适应控制器的 K 增益
    float z;
    control_sys_e = control_sys_r - control_sys_y;//k 时刻的误差
    printf("r is:%5.2f    y is %5.2f\n", control_sys_r, control_sys_y);
    printf("e is: %7.2f\n", control_sys_e);
    //printf("w1_1:   %7.2f, e: %7.2f, u_1: %7.2f, data0: %7.2f\n", w1_1,
control_sys_e,u_1, neural_input_data[0]);
    z = control_sys_r - control_sys_y; control_sys_e = z;
    neural_input_data[0] = control_sys_e;                    //误差的
位置量
    neural_input_data[1] = control_sys_e -e_1;               //误差的
速度量
    neural_input_data[2] = control_sys_e -2*e_1+e_2;         //误差的
加速度量
printf("w1_1:   %7.2f, xite: %7.2f, z: %7.2f, u_1: %7.2f, data0: %7.2f\n", w1_1, xite,
z,u_1, neural_input_data[0]);
```

```
    w1 = 0.1;
    w2 = 0.1;
    w3 = 0.1;
    if (w1 < WMIN || w1 > WMAX) {
        w1 = 0;
    }
    if (w2 < WMIN || w2 > WMAX) {
        w2 = 0;
    }
    if (w3 < WMIN || w3 > WMAX) {
        w3 = 0;
    }
    printf("w1: %7.2f, w2: %7.2f, w3: %7.2f\n", w1, w2, w3);
    control_sys_u
=u_1+K*(w1*neural_input_data[0]+w2*neural_input_data[1]+w3*neural_input_data
[2]) ;   //Control law
    printf("u is %7.2f\n\n", control_sys_u);
    if (control_sys_u > 10000 || control_sys_u < -10000) {
        control_sys_u   = 0;
    }
    float output;
    output =0.8464*y_1 + 32.01*u_1+0.1064*u_2;     //直流伺服电机的传递函数
得到的差分方程
    printf("output is %7.2f\n\n", output);
    //保存参数
    w1 = w1_1 + xite*z*u_1*neural_input_data[0];
    w2 = w2_1 + xite*z*u_1*neural_input_data[1];
    w3 = w3_1 + xite*z*u_1*neural_input_data[3];
    w1_1 = w1; w2_1 = w2; w3_1= w3;
    u_2 = u_1; u_1 = control_sys_u;
    e_2 = e_1; e_1 = control_sys_e;
    y_2 = y_1; y_1 = control_sys_y;
    return(output);
}
```

五、 实验总结

本次实验是利用神经网络进行电机随动系统的实现，通过调节学习率和 K 增益，实现不同的随动效果，神经网络学习法可以建立较好的随动系统，但是稳定性不是很高，经过多次仿真可以得到稳定性较好的仿真效果。

此次实验模拟了单神经元自适应控制器，通过不断对其参数的调整来不断完善模型，以期达到理想的仿真结果，虽然本次实验时间紧凑，最后没有达到令人满意的效果，但加深了我们对神经元网络的理解与认识，同时也让我们进一步熟悉了MATLAB在智能控制这方面的功能与应用。

实验三 遗传算法在角度随动系统中的应用

一、 实验目的与意义

学习基于二进制编码遗传算法的 PID 整定方法;了解遗传算法在角度随动系统中的应用。

二、 实验环境

PC 平台:Matlab、Ubuntu 交叉编译环境

目标机:友善之臂 MINI2440

三、 实验内容

(1) 了解数字 PID 控制原理

(2) 了解基于二进制编码遗传算法的 PID 整定的方法、步骤;

(3) 在 Matlab 中建立基于二进制编码遗传算法的 PID 整定仿真程序;

(4) 自己设计一种适应度函数，并进行仿真实验。

(5) 在角度随动系统中验证得到的 PID 参数。

四、 实验内容及步骤

1. 数字 PID 控制

计算机控制是一种采样控制，它只能根据采样时刻的偏差值计算控制量。因此，连续 PID 控制算法不能直接使用，需要采用离散化方法。在计算机 PID 控制中，使用的是数字 PID 控制器。

按模拟 PID 控制算法，以一系列的采样时刻点 kT 代表连续时间 t，以矩形法数值计算近似代替积分，以一阶后向差分近似代替微分，即:

$$\begin{cases} \mathrm{k} \approx \mathrm{kT}(\mathrm{k}=0,1,2,\cdots) \\ \int_0^t error(t)dt \approx T\sum_{j=0}^{k} error(jT) = T\sum_{j=0}^{k} error(j) \\ \dfrac{derror(t)}{dt} \approx \dfrac{error(kT)-error[(k-1)T]}{T} = \dfrac{error(k)-error(k-1)}{T} \end{cases}$$

可得离散 PID 表达式:

$$u(k) = k_p\left(error(k) + \frac{T}{T_1}\sum_{j=0}^{k} + \frac{T_D}{T}\big(error(k)-error(k-1)\big)\right)$$

$$= k_p\left(error(k) + k_i\sum_{j=0}^{k} error(j)T + k_d\left(\frac{error(k)-error(k-1)}{T}\right)\right)$$

式中$k_i = \frac{k_p}{T}$，$k_d = k_pT_D$，T 为采样周期，k 为采样序号，k =1,2,… error(k -1) 和 error(k) 分别为第(k-1)和第 k 时刻所得的偏差信号。

2. 基于遗传算法的 PID 整定原理

(1) 参数的确定及表示

首先确定参数范围，该范围一般由用户给定，对其进行编码。选取二进制字串来表示每一个参数，并建立于参数间的关系。再把二进制串连接起来就组成一个长的二进制字串，该字串为遗传算法可以操作的对象。

(2) 选取初始种群

因为需要编程来实现各过程，所以采用计算机随机产生初始种群。针对二进制编码而言，先产生 0~1 之间均匀分布的随机数，然后规定产生的随机数 0~0.5 之间带来表 0，0.5~1 之间代表 1。此外，考虑到计算的复杂程度来规定种群的大小。

(3) 适配函数的确定

一般的寻优方法在约束条件下可以求得满足条件的一组参数，在设计中是从该组参数中寻找一个最好的。衡量一个控制系统的指标有三个方面，即稳定性、准确性和快速性。而上升时间反映了系统的快速性，上升时间越短，控制进行得就越快，系统品质也就越好。

如果单纯追求系统的动态特性，得到的参数很可能使控制信号过大，在实际应用中会因系统中固有的饱和特性而导致系统不稳定，为了防止控制能量过大，在实际应用中加入控制量。因此为了使控制效果更好，我们给出了控制量、误差和上升时间作为约束条件。因为适应函数同目标函数相关，所以目标函数确定后，直接将其作为适配函数进行参数寻优。最优的控制参数也就是在满足约束条件下使 f (x) 最大时，x 所对应的控制器参数。

(4) 遗传算法的操作

首先利用适应度比例法进行复制。即通过适配函数求得适配值，进而求每个串对应的复制概率。复制概率与每代字串的个数的乘积为该串在下一

代中应复制的个数。复制概率大的在下一代中将有较多的子孙，相反则会被淘汰。

其次进行单点交叉，交叉概率为 Pc。从复制后的成员里以 Pc 的概率选取字串组成匹配池，而后对匹配池的成员随机匹配，交叉的位置也是随机确定的。

最后以概率 Pm 进行变异。假如每代有 15 个字串，每个字串 12 位，则共有 15*12=180 个串位，期望的变异串位数为 180*0.01=2（位），即每代中有两个串位要由 1 变为 0 或由 0 变为 1。

初始种群通过复制、交叉及变异得到了新一代种群，该代种群经解码后代入适配函数，观察是否满足结束条件，若不满足，则重复以上操作直到满足为止。

结束条件由具体问题所定，只要各目标参数在规定范围内，则终止计算。

3. 遗传算法 MATLAB 仿真实验流程

四个运行参数的设定与调节：

(1) S:群体大小，即群体中所含个体的数量，取 20-100。

(2) G:遗传算法的终止进化代数，取 100-500。

(3) Pc:交叉概率，取 0.4-0.99。

(4) Pm:变异概率，取 0.0001-0.1。

4. 遗传算法的 MATLAB 仿真程序设计：

(1) 在 ga_pid.m 中对遗传算法的遗传代数、种群规模、编码长度以及 PID 初始值进行设置。

(2) 对 PID 三个参数进行编解码操作。

(3) 交叉步骤的编写。主要设计交叉概率与交叉操作。

(4) 进行变异步骤的编写。主要设计变异概率与变异操作。

(5) 在 ga_function.m 中填写采样时间以及电机传递函数。

(6) 运行仿真程序，可以获得最优指标图像与输出仿真图像如下两图：

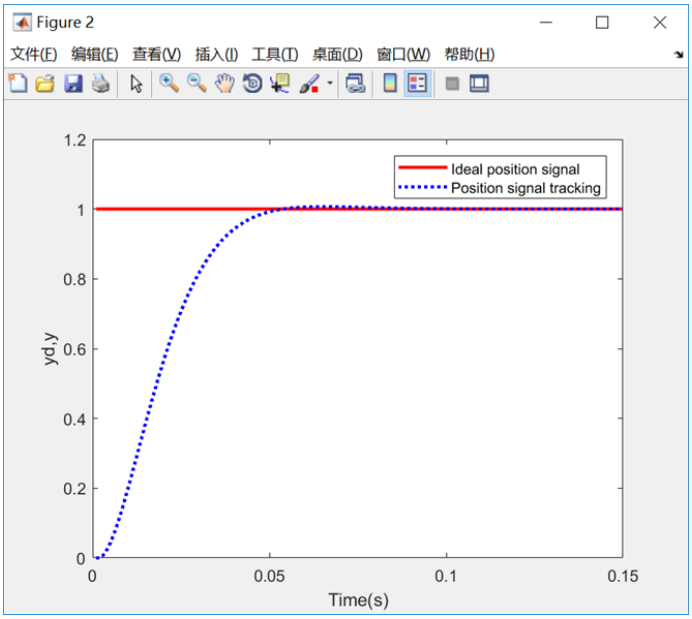


图 17 最优指标图像

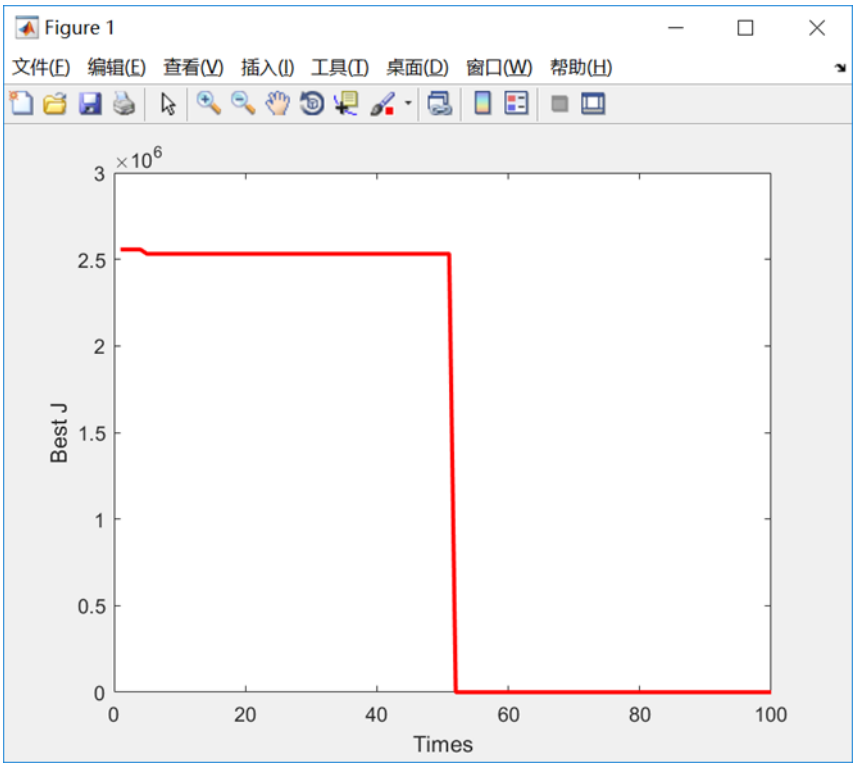


图 18 输出仿真图

(7) 直流伺服电机传递函数：

$$G(s) = \frac{7.463}{0.0002202s^2 + 0.006s + 1}$$

MATLAB 代码：

```
%基于遗传算法的 PID 整定
clear all;
close all;
global yd y timef

G=100;                                          %遗传代数？
Size=100;                                       %种群大小？
CodeL=8;                                        %编码位数？

MinX(1)=zeros(1);     %kp 取值范围下限为 0
MaxX(1)=20*ones(1); %kp 取值范围上限为 20
MinX(2)=zeros(1);     %ki 取值范围下限为 0
MaxX(2)=1.0*ones(1);%ki 取值范围上限为 1
MinX(3)=zeros(1);     %kd 取值范围下限为 0
MaxX(3)=1.0*ones(1);%kd 取值范围下限为 1

%************   参数编解码  ***********

E=round(rand(Size,3*CodeL));       %Initial Code!  先由 rand 函数生成 30X30 的矩阵，然后对矩阵元素进行四舍五入，即初始种群

BsJ=1;                    %所期望的最优指标为 0，即 J->0. BsJ 是 best J 的缩写

for kg=1:1:G              %100 代进化的主循环体
    time(kg)=kg;              %time 变量为代数，从 1 到 100

    for s=1:1:Size          %对每一代的种群中的 30 个样本（个体）进行循环
        m=E(s,:);                  %取 E30X30 矩阵的第 S 行，将其赋值给 m，m 的维度是 1X30，m 为当代种群中的某个个体
        y1=0;y2=0;y3=0;

        m1=m(1:1:CodeL);       %P 值编解码%取 m(1X30)的前 10 位
        for i=1:1:CodeL
            y1=y1+m1(i)*2^(i-1);%将前 10 位二进制编码转化为 10 进制
        end
        Kpid(s,1)=(MaxX(1)-MinX(1))*y1/1023+MinX(1);%对 kp 的染色体编码进行解码

        m2=m(CodeL+1:1:2*CodeL);                %I 值编解码%取 m(1X30)的中间 10 位
        for i=1:1:CodeL
            y2=y2+m2(i)*2^(i-1);                %将中间 10 位二进制编码转化为 10 进制
        end
```

```
        Kpid(s,2)=(MaxX(2)-MinX(2))*y2/1023+MinX(2);          %对 ki 的染色体编码进行解码

        m3=m(2*CodeL+1:1:end);                    %D 值编解码%取 m(1X30)的最后 10 位
        for i=1:1:CodeL
            y3=y3+m3(i)*2^(i-1);                  %将最后 10 位二进制编码转化为 10 进制
        end
        Kpid(s,3)=(MaxX(3)-MinX(3))*y3/1023+MinX(3);          %即对 kd 的染色体编码进行解码

%****** Step 1 : Evaluate BestJ ******
        Kpidi=Kpid(s,:);%取 Kpid 的第 s 行赋值给 Kpidi

        [Kpidi,BsJ]=ga_function(Kpidi,BsJ);%计算适应度

        BsJi(s)=BsJ;
    end

    [OderJi,IndexJi]=sort(BsJi);%对 BsJi 数组升序排序
    BestJ(kg)=OderJi(1);
    BJ=BestJ(kg);
    Ji=BsJi+1e-10;%避免分母为 0，即避免奇异

   fi=1./Ji;
   %Oderfi 为数值，Indexfi 为原序号，用 sort 函数对 fi 数组升序排序
   [Oderfi,Indexfi]=sort(fi);              %Arranging fi small to bigger

    Bestfi=Oderfi(Size);              % Let Bestfi=max(fi)，即 Bestfi 为最大的值
    BestS=E(Indexfi(Size),:);     % Let BestS=E(m), m is the Indexfi belong to max(fi)

    kg
    BJ
    BestS;

%****** Step 2 : 选择与复制操作 Select and Reproduct Operation******
   fi_sum=sum(fi);%对 fi 数组求和
   fi_Size=(Oderfi/fi_sum)*Size;

   fi_S=floor(fi_Size);              %Selecting Bigger fi value，向下取整，即 f_s 为不大于且最接近 fi_size 的整数
```

```
    r=Size-sum(fi_S);
    Rest=fi_Size-fi_S;                  %残差
    [RestValue,Index]=sort(Rest);%对 Rest 升序排序
    for i=Size:-1:Size-r+1
        fi_S(Index(i))=fi_S(Index(i))+1;        % Adding rest to equal Size
    end


    kk=1;
    for i=1:1:Size
        for j=1:1:fi_S(i)             %Select and Reproduce
         TempE(kk,:)=E(Indexfi(i),:);
            kk=kk+1;                    %kk is used to reproduce
        end
    end

E=TempE;
%************ Step 3 : 交叉操作  Crossover Operation ************
    pc=0.63;%交叉概率  建议 0.4~0.99
    n=ceil(20*rand);%n 为随机交叉点，使用 ceil 函数向离它最近的大整数取整
    for i=1:2:(Size-1)
        temp=rand;                      %设定交叉点
        if pc>temp                      %Crossover Condition，以 pc 的概率进行
交叉操作
            for j=n:1:20                %以下为交叉操作，在 n 到 20 之间的
染色体基因位进行交叉
                TempE(i,j)=E(i+1,j);    %?
                TempE(i+1,j)=E(i,j);    %?
            end
        end
    end
    TempE(Size,:)=BestS;
    E=TempE;

%************ Step 4:变异操作  Mutation Operation **************

    pm=0.002; %变异概率，建议  0.0001~0.1

    for i=1:1:Size
      for j=1:1:3*CodeL
        temp=rand;
        if pm>temp                      %Mutation Condition，变异操作
            if TempE(i,j)==0
                TempE(i,j)=1;
```

```
                else
                    TempE(i,j)=0;
                end
            end
        end
    end
    %Guarantee TempE(Size,:) belong to the best individual
    TempE(Size,:)=BestS;
    E=TempE;
%*********************************************************
end

Bestfi
BestS
Kpidi
Best_J=BestJ(G)
figure(1);
plot(time,BestJ,'r','linewidth',2);
xlabel('Times');ylabel('Best J');
figure(2);
plot(timef,yd,'r',timef,y,'b:','linewidth',2);
xlabel('Time(s)');ylabel('yd,y');
legend('Ideal position signal','Position signal tracking');
```

求得的 PID 的参数分别是：

```
Kpidi =    0         0       0.2483
Best_J = 20.3517
```

5. 角度随动系统实验

将在 MATLAB 仿真中所获得的 PID 参数填入，ga_pid.c 文件中的相应位置。并将其在虚拟机中编译成功后，将可执行文件下载到角度随动试验系统中。运行该程序，观察随动电机与主动电机间的运动关系，是否达到预期要求。

1、阶跃响应实验

(1) 给实验箱通电，打开 mini2440 开发板电源，将在虚拟机中生成的可执行文件导入到 mini2440 开发板中。

(2) 将 C8051 开发板电源线拆下，并将目标电机的转轴旋转一个角度，同时将左侧随动电机旋转至水平位置。

(3) 按下电机供电电源，运行程序，观察随动电机是否能够快速、准确的到达目标位置。

2、正弦随动实验

(1) 给实验箱通电，打开 mini2440 开发板电源，将在虚拟机中生成的可执行文件导入到 mini2440 开发板中。

(2) 将 C8051 开发板电源线装好，并将目标电机与随动电机的转轴旋转至水平位置。

(3) 按下电机供电电源，运行程序，观察随动电机是否能够快速、准确的到达目标位置。

五、 实验总结

将仿真的数据加填入实际的随动系统中，我们发现系统可以实现随动的功能，但是在稳定性和准确性上不是很好，所以，我们对参数进行了微调， 通过对 PID 参数的整定，达到较为优化的随动效果，电机跟踪性良好。

此次实验模拟了遗传算法优化PID参数的过程。通过对各个参数的调整和仿真，我们对遗传算法有了更加深刻的理解，并对MATLAB在智能控制方面的功能与应用有了更多的了解。