《自动控制实践 B》

综合实验

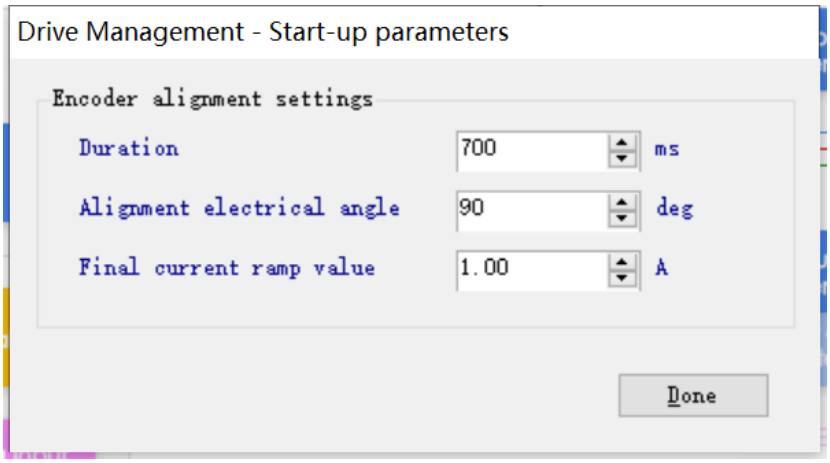
实验报告

学院　　机电工程与自动化
姓名　　吕家昊
学号　　210320111
日期　　2024 年 5 月 20 日

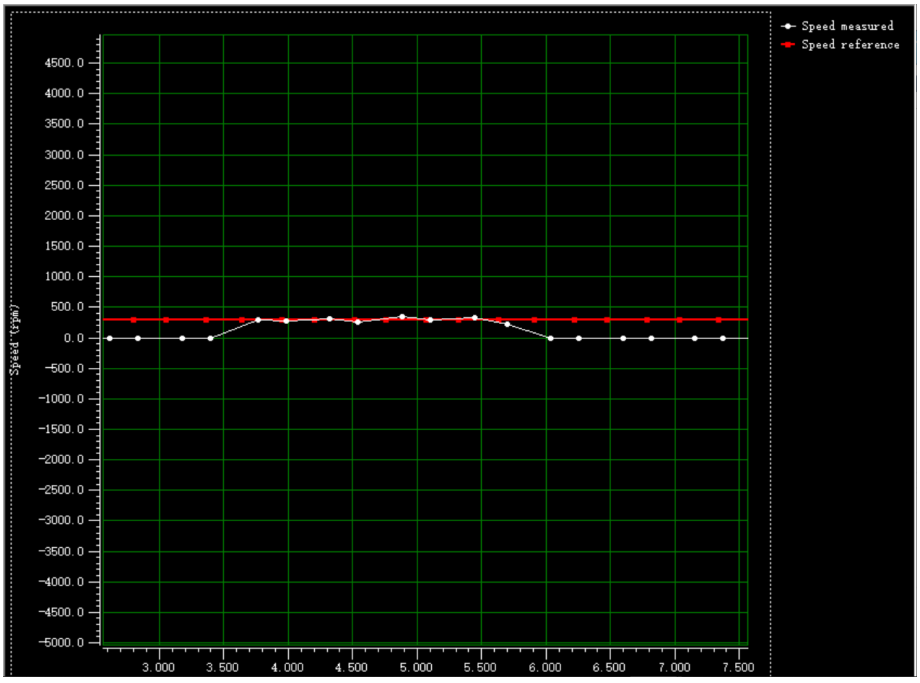
目录

任务一 电机控制库认知

1.1 Workbench 编码器对齐参数配置



1.2 速度波形显示界面



1.3 说说你对 ST MC SDK5.x（电机控制库）的认知

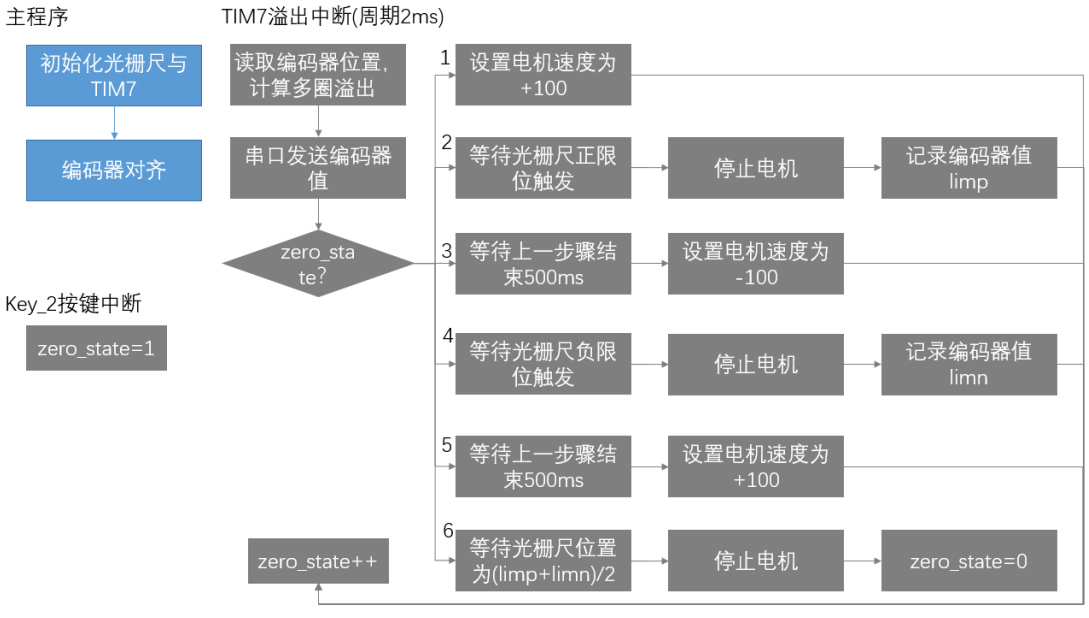
可以从电机控制库的开发背景、发展历程、组成结构、功能、软件使用等任意方面阐述自己的理解，字数不限。

MC SDK5.x 是意法半导体为 STM32 提供的电机控制库，用于实现各类电机的 FOC 以及其它控制方法。
控制库的架构分为芯片外设库（使用 HAL/LL 库）、电机层（实现控制算法）与电机应用层（供开发者直接进行调用），实现了电机启停、速度/力矩控制、状态反馈等多种功能。
使用 MC SDK5.x 时，可在软件内配置电机的各项参数，并选定 STM32 对应的外设引脚，生成 CubeMX 与 STM32 工程文件。编译完成后，还可通过 MCSDK 的 UI 界面实现与下位机通信，实时调整电机的运行状态。
在代码使用过程中，可通过 pMCI 结构体，在 Debug 模式下对电机状态进行实时监测，如控制模式、接收指令、错误状态等。

任务二 按键控制滑台回零

2.1 程序流程图

画出滑台回零程序流程图。



2.2 功能代码

在此处粘贴自己编写的代码，并写好代码注释。

main()：（省略 CubeMX 自动生成函数）

```
int8_t encode_round = 0;  // 编码器溢出圈数
uint16_t last_encode = 0;  // 上一采样点编码器位置
int encode_32;  // 32位编码器位置
uint8_t zero_state = 0;  // 回零步骤变量
uint16_t limp_pos = 0, limn_pos = 0, limp_round = 0, limn_round = 0;  // 限位记录编码器位置
uint8_t txbuf[11];  // 串口发送内容

TIM7->CNT = 1999;
HAL_TIM_Base_Start_IT(&htim7);

HAL_TIM_Encoder_Start(&htim3, TIM_CHANNEL_ALL);
MC_AlignEncoderMotor1();  // 对齐编码器
HAL_Delay(1000);

while (1)
{
}
```

中断：

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
static uint32_t wait_time = 0;  // 记录开始等待时间

if(htim == &htim7){  // TIM7溢出中断, 2ms执行周期
//  处理正负溢出
```

```
    if(last_encode > 0xFF00 && TIM3->CNT < 0x0100) encode_round++;
    else if (last_encode < 0x0100 && TIM3->CNT > 0xFF00) encode_round--;
    last_encode = TIM3->CNT;

    encode_32 = (int32_t)encode_round * 0x10000 + (int32_t)(TIM3->CNT); // 当前
编码器位置

    // 任务3中以下代码移至main的while(1)中
    // 串口发送
    memset(txbuf, 0, 11);
    sprintf((char*)txbuf, "%d\n", encode_32);
    HAL_UART_Transmit(&huart5, txbuf, strlen((char*)txbuf), 2);

    // 按下Key_2后，依次执行以下步骤
    if(zero_state == 1){  // 电机正转
    MC_ProgramSpeedRampMotor1(100, 100);
    MC_StartMotor1();
    zero_state = 2;
    } else if(zero_state == 2){  // 到达光栅尺正限位后停止
    if(!HAL_GPIO_ReadPin(GPIOG, GPIO_PIN_1)){
    limp_pos = TIM3->CNT;
    limp_round = encode_round;
    MC_StopMotor1();
    zero_state = 3;
    wait_time = HAL_GetTick();  // 等待500ms
    }
    } else if(zero_state == 3){  // 电机反转
    if(HAL_GetTick() - wait_time > 500){
```

```
    MC_ProgramSpeedRampMotor1(-100, 100);
    MC_StartMotor1();
    zero_state = 4;
    }
    } else if(zero_state == 4){  // 到达光栅尺正限位后停止
    if(!HAL_GPIO_ReadPin(GPIOG, GPIO_PIN_0)){
    limn_pos = TIM3->CNT;
    limn_round = encode_round;
    MC_StopMotor1();
    zero_state = 5;
    wait_time = HAL_GetTick();
    }
    } else if(zero_state == 5){  // 电机正转
    if(HAL_GetTick() - wait_time > 500){
    MC_ProgramSpeedRampMotor1(100, 100);
    MC_StartMotor1();
    zero_state = 6;
    }
    } else if(zero_state == 6){  // 到达计算出的零位后停止
    // 在32位下计算编码器中间位置, 防止溢出
    int32_t target_encode_32 = (limp_pos + limn_pos + (limp_round +
    limn_round)*0x10000) / 2;
    if(encode_32 > target_encode_32){
    MC_StopMotor1();
    zero_state = 0;
    }
    }
```

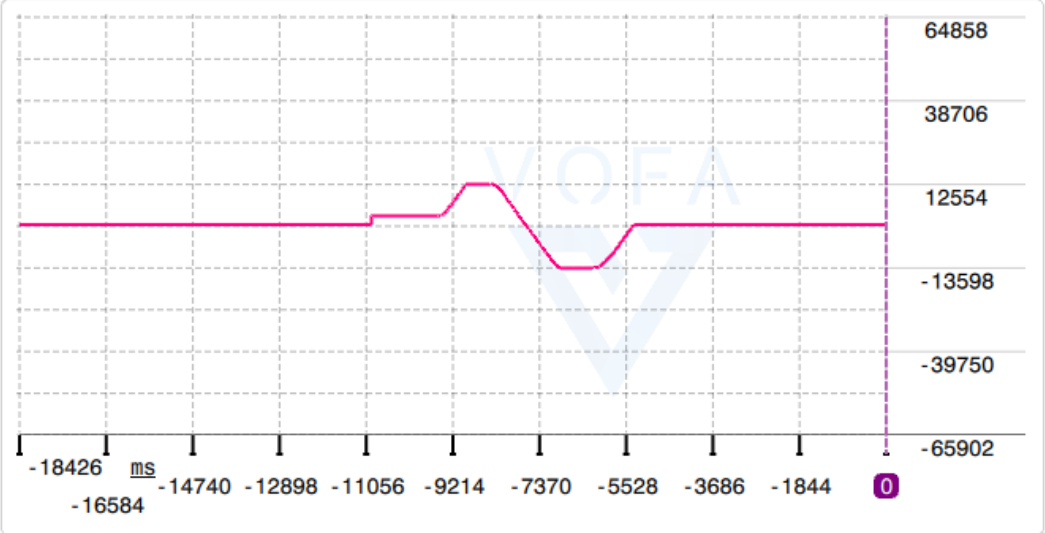
```
    }
    }

    void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
    {
    // PE1(Key2)按键中断
    if(GPIO_Pin == GPIO_PIN_1){
    if(zero_state == 0){ // 防止重复触发
    zero_state = 1;
    }
    }
    }
```

2.3 实验总结

说说自己在开发中遇到的主要问题，及最终解决方法。

运行结果：

编码器返回值如下。将电机向正方向移动一段距离，按下 Key_2 后，电机能自动回零。



问题：

添加串口发送功能后回零过程无法进行。
解决方法：
①构建串口发送内容时，最初使用如下函数实现整型转字符串：

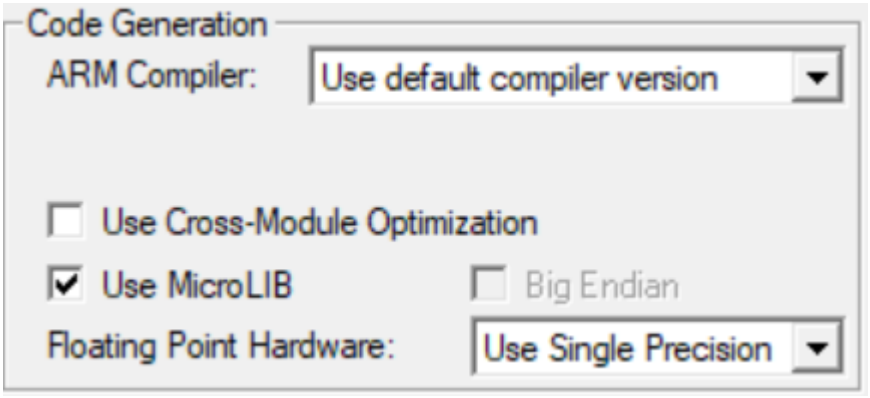
```
uint8_t index = 0;
if(encode_32 < 0){
txbuf[0] = '-';
index = 1;
}

int encode_32_abs = abs(encode_32);
if(encode_32_abs == 0){
txbuf[index] = '0';
index++;
} else {
int8_t digits = (int8_t)(log10(encode_32_abs) + 1);
for(;digits > 0; digits--){
txbuf[index] = '0' + encode_32_abs / (int)pow(10, digits - 1);
encode_32_abs %= (int)pow(10, digits - 1);
index++;
}
}
```

删除标红代码部分后代码正常运行。
因此，考虑代码中可能存在非法内存访问。将此部分代码改为更安全的 sprintf((char*)txbuf, "%d\n", encode_32);后正常运行。
②程序执行频率过高（TIM7 中断周期为 1ms）导致串口发送阻塞。
txbuf 数组设定长度为 11bytes，串口发送波特率为 115200，无校验位，停止位 1 位，则每 byte 需要发送 10 个数据位，每个数据包总发送时间为

$$\frac{10\text{bit} * 11}{115200\text{bit/s}} = 0.955\text{ms}$$

考虑到两个 byte 之间存在空闲时间，发送整个数据包的时间与 TIM7 其他代码运行时间之和很可能超过 TIM7 周期 1ms，因此代码无法正常运行。

将 HAL_UART_Transmit 的数组长度改为 strlen((char*)txbuf)，则串口发送数据最大长度为 7bytes。另一方面，将 TIM7 的频率改为 2ms，程序正常运行。
③在 Keil 设置中，选择 Use MicroLIB。



任务三 控制系统设计

3.1 控制系统建模

3.1.1 机械谐振模态分析

计算联轴器的谐振频率；与控制系统要求的带宽进行对比，确定机械谐振模态是否可以忽略；写出计算过程。

联轴器扭矩刚性$\mathrm{K}=950\mathrm{N}\cdot\mathrm{m/rad}$
求解转动惯量：电机转动惯量$J_1=0.28kg\cdot m^2=2.8*10^{-5}kg\cdot m^2$
滑台转动惯量$J_2=m_2\left(\frac{h}{2\pi}\right)^2=0.54*\left(\frac{10*10^{-3}}{2\pi}\right)^2=1.368*10^{-6}kg\cdot m^2$
丝杆转动惯量$J_3=\frac{m_3r^2}{2}=\frac{0.6*\left(7.5*10^{-3}\right)^2}{2}=1.69*10^{-5}kg\cdot m^2$
总转动惯量$J=J_1+J_2+J_3=4.627*10^{-5}kg\cdot m^2$
系统固有频率$\omega_{\mathrm{BW}}=\sqrt{\frac{\mathrm{K}}{\mathrm{J}}}=4.531*10^3rad/s$
系统要求输入正弦信号为 1Hz，扫频辨识中最大频率为 50Hz，远低于谐振频率$\mathrm{f}=\frac{\omega}{2\pi}=721.13Hz$（满足$\omega_{\mathrm{m}}<\frac{\omega_{BW}}{5}$），因此机械谐振可忽略。

3.1.2 被控对象的系统建模

结合课上所学内容，画出控制系统方框图，对被控对象进行建模，并对模型进行简化处理。

电机采用 FOC 控制，通过计算出所需电流$\mathrm{I_{qref}}$，与实际电流$\mathrm{I_{qfdb}}$得到电流误差，利用 PI 控制器得到$\mathrm{V_q}$。此后，通过逆变器得到各项电压V_a, V_b, V_c并生成对应占空比的 PWM 波。

此时电机可等效为电压为$\mathrm{V_q}$的直流电机。由以下表达式对系统进行建模：

$$\mathrm{V_q} = E_a + I_a R_a + L_a \frac{dI_a}{dt}$$

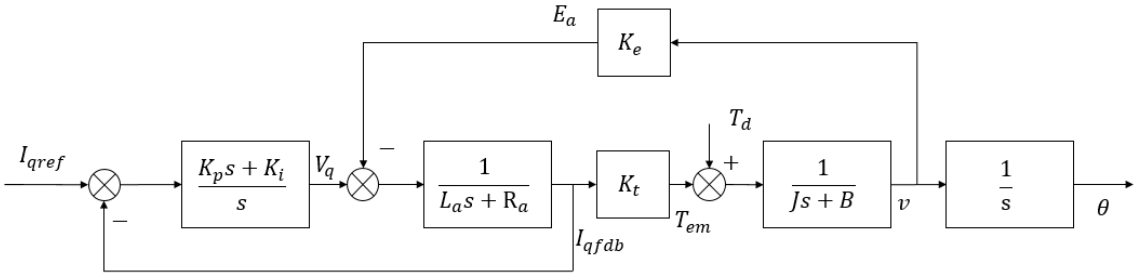
$$\mathrm{T_{em}} = K_t I_a$$

$$\mathrm{T_{em}} + T_d = J\frac{dv}{dt} + Bv$$

$$\mathrm{K_e} v = E_a$$

$$v = \frac{\mathrm{d\theta}}{\mathrm{dt}}$$

被控对象（电机及 FOC 控制系统）方框图如下：



考虑对系统进行简化：取阻尼$\mathrm{B} = 0$。经整理，v对$\mathrm{I_{qref}}$传递函数有如下形式：

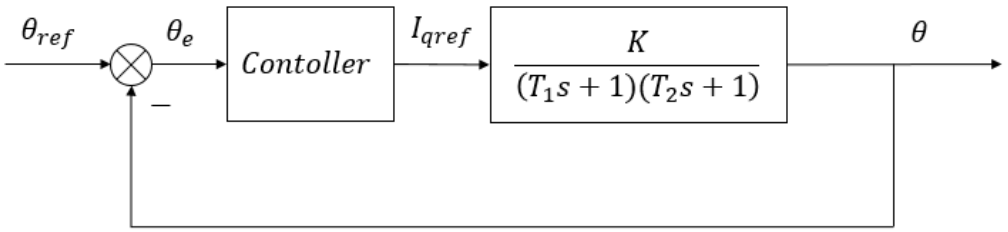
$$\frac{\mathrm{v(s)}}{\mathrm{I_{qref(s)}}} = \frac{K(T_z s + 1)}{(T_1 s + 1)(T_2 s + 1)}$$

即$\frac{\theta(\mathrm{s})}{\mathrm{I_{qref(s)}}} = \frac{K}{s(T_1 s+1)(T_2 s+1)}$。以下将对$\frac{\mathrm{v(s)}}{\mathrm{I_{qref(s)}}}$进行整定。

事实上，采用以上模型与$\frac{\mathrm{v(s)}}{\mathrm{I_{qref(s)}}} = \frac{K}{(T_1 s+1)(T_2 s+1)}$整定匹配度大致相同，且得到$\mathrm{T_z} =$

-0.0043，即该模型对T_z有较大不确定性。为简化起见，整定采用模型$\frac{K}{(T_1s+1)(T_2s+1)}$。

控制系统整体框图如下：



3.2 控制系统辨识

3.2.1 在主控板上实现正弦扫频信号生成算法

将正弦扫频信号生成算法的源代码粘贴在下面的方框中，并写好代码注释。

```
uint32_t t = HAL_GetTick() - wait_time;  // 执行扫频的时间, 单位ms
// 扫频函数  y = chirp(t=0:0.002:10, f0=0.5, t1=10, f1=10);
// 计算参数
// k = exp(ln(f1/f0) / t1) = 1.3493
// p = 2pif0/ln(k) = 10.4864
I = sin(10.4864 * (pow(1.3493, t / 1000.0) - 1));  // 电流上限3.13A
int16_t I_16 = (int16_t)(I * 1596.7f);  // 系数65536*0.02*4.02/3.3, 电流值上限对应4997
// 输入限幅
if(I_16 > 4500) I_16 = 4500;
else if(I_16 < -4500) I_16 = -4500;
```

3.2.2 在主控板上实现扫频辨识功能

1) 将扫频辨识功能源代码粘贴在下面的方框中，并写好代码注释。

```
main():
while(1){
static uint32_t wait_time = 0;  // 记录开始等待时间

// 控制while(1)执行频率
static uint32_t last_run_time = 0;
if(HAL_GetTick() - last_run_time < period){
continue;
}
last_run_time = HAL_GetTick();

/* 省略回零代码 */

if(sweep_state == 1){
wait_time = HAL_GetTick();
MC_ProgramTorqueRampMotor1(0, 0);  // 初始转矩设0
MC_StartMotor1();
sweep_state = 2;
} else if(sweep_state == 2){
uint32_t t = HAL_GetTick() - wait_time;  // 执行扫频的时间, 单位ms
// 扫频函数  y = chirp(t=0:0.002:10, f0=0.5, t1=10, f1=10);
// 计算参数
// k = exp(ln(f1/f0) / t1) = 1.3493
// p = 2pif0/ln(k) = 10.4864
I = sin(10.4864 * (pow(1.3493, t / 1000.0) - 1));  // 电流, 上限3.13A
```

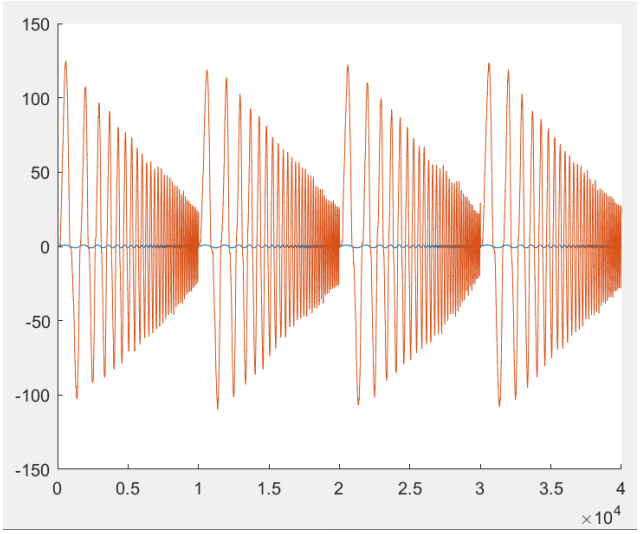
```
int16_t I_16 = (int16_t)(I * 1596.7f);  // 系数65536*0.02*4.02/3.3, 电流值上限对应4997
// 输入限幅
if(I_16 > 4500) I_16 = 4500;
else if(I_16 < -4500) I_16 = -4500;

if(t <= 10000){
MC_ProgramTorqueRampMotor1(I_16, 0);
output = MC_GetMecSpeedAverageMotor1();  // 函数返回单位0.1Hz, 转换为rad/s
output *= 0.6283f;
// 串口发送整数
sprintf((char*)txbuf, "%d,%d,%d\r\n", t+10000*sweep_cnt, (int)(I*1000), (int)(output*1000));
HAL_UART_Transmit(&huart5, txbuf, strlen((char*)txbuf),period);
} else {
// t>10结束扫频
MC_StopMotor1();
sweep_state = 0;
sweep_cnt++;
}
}
}
```

PE2中断：

```
// PE2(Key3) 扫频
if(GPIO_Pin == GPIO_PIN_2){
if(sweep_state == 0){ // 防止重复触发
sweep_state = 1;
```

```
    }

    }
```

2） 将扫频辨识过程中Matlab采集波形保存并粘贴在下方。
（扫频辨识波形数据，参考图 3.22）
时间单位为 ms，蓝色为输入电流，红色为转矩输出。



3.2.3 使用 matlab 系统辨识工具箱，获得辨识的系统模型

1） 将输入输出数据导入Matlab系统辨识工具箱后，Time Plot查看输入输出曲线，将曲线截图粘贴在下方。
（输入输出曲线截图，参考图 3.26）

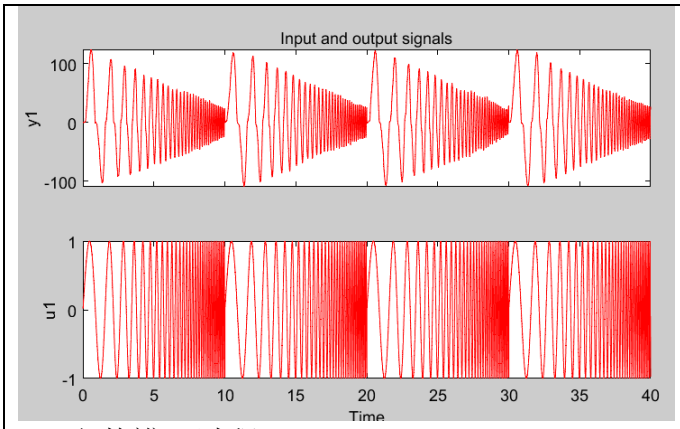
2） 参数辨识结束后，将参数辨识计算过程截图粘贴在下方。
（参数辨识计算过程截图，参考图 3.29）

3） 参数辨识结束后，勾选 Model output,可以看到辨识的模型输出和实际的输出，将曲线截图并粘贴在下方。
（模型输出与实际输出截图，参考图 3.30）

4） 将参数辨识结果截图并粘贴在下方。
（参数辨识结果截图，参考图 3.31）

5） 写出系统开环传递函数。

1）输入电流(u1)单位为 A，输出转速(y1)单位为 rad/s。



2）参数辨识过程：

```
Process Model Identification

Estimation data: Time domain data mydata
Data has 1 outputs, 1 inputs and 8000 samples.
Model Type:
    'P2'
```

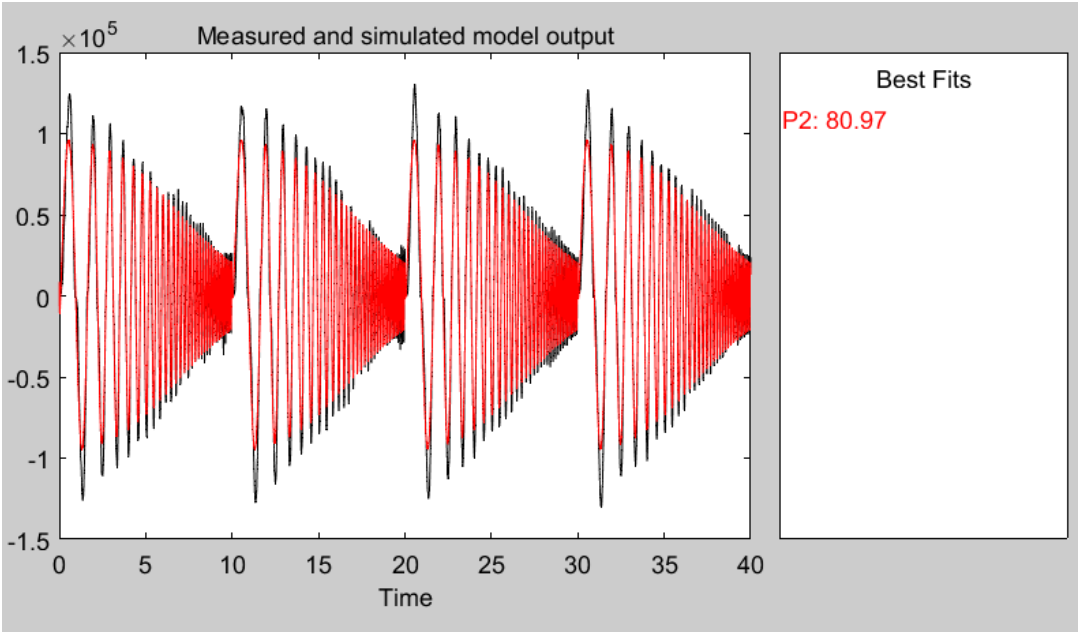
Estimation Progress

```
    Name          New Value      Previous Value      Direction
    Kp:            98.1384          97.9771             0.322653
    1/Tp1:        15.6721          15.4272             0.489798
    1/Tp2:        110.407          151.578             -82.3419
    Step size: 41.172
    First-order optimality: 11.3445
    Expected improvement: 0.0348011%
    Achieved improvement: 3.04455%
Iteration 3:
    Current cost: 106.008    Previous cost: 106.146
    Name          New Value      Previous Value      Direction
    Kp:            98.6278          98.1384             0.489354
    1/Tp1:        15.7743          15.6721             0.102221
    1/Tp2:        103.734          110.407             -6.67267
    Step size: 6.69137
    First-order optimality: 4.22736
    Expected improvement: 0.00141517%
    Achieved improvement: 0.129708%
Iteration 4:
    Current cost: 106.001    Previous cost: 106.008
    Name          New Value      Previous Value      Direction
    Kp:            98.7024          98.6278             0.074676
    1/Tp1:        15.7136          15.7743             -0.060705
    1/Tp2:        106.166          103.734             2.43181
    Step size: 2.43371
    First-order optimality: 0.526116
    Expected improvement: 8.5954e-05%
    Achieved improvement: 0.00665268%
Estimation complete.
Final improvement: 0.00665268%
Final first-order optimality (largest slope): 0.526116
Final cost: 106.001
```

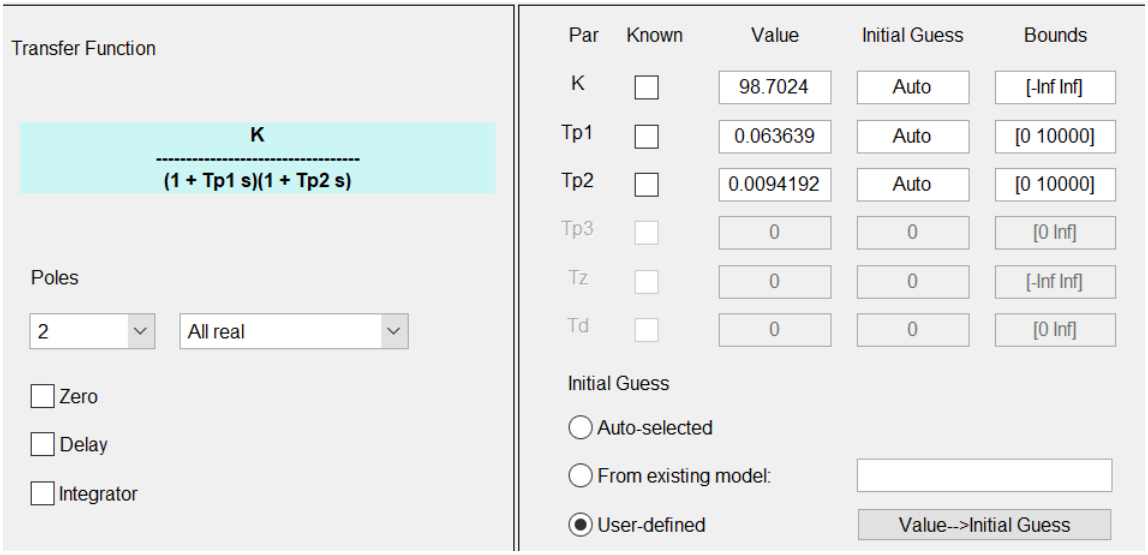
Result

```
Termination condition: Near (local) minimum, (norm(g) < tol)..
Number of iterations: 4, Number of function evaluations: 12

Status: Estimated using PROCEST
Fit to estimation data: 79.96%, FPE: 106.08
```

3）模型与实际输出：



4）参数辨识结果：



5）电机开环传递函数：

$$G_0(s) = \frac{98.7024}{(0.063639s + 1)(0.0094192s + 1)} = \frac{164661}{s^2 + 121.9s + 1668}$$

若输入为代码中发送给电机的控制信息 I_16(±4997)，输出单位为 0.1Hz（如指导书 3.4），则开环传递函数表示为

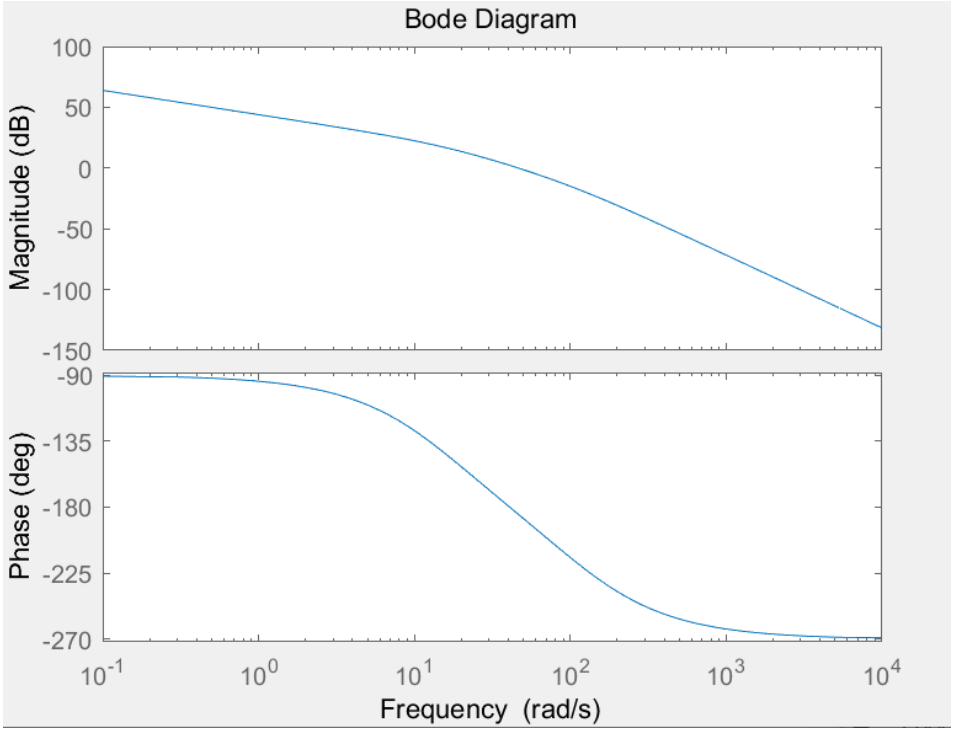
$$G_0'(s) = \frac{103.14}{s^2 + 121.9s + 1668}$$

3.3 控制器设计

推荐使用：频域的方法进行控制器的设计，使得控制器的带宽，相位裕度等满足控制系统任务指标。
请同学在这里给出控制器设计的详细过程（可以是理论推导设计过程，或者 matlab 辅助设计过程），并结合 bode 图给出相位裕度和幅值裕度的情况。

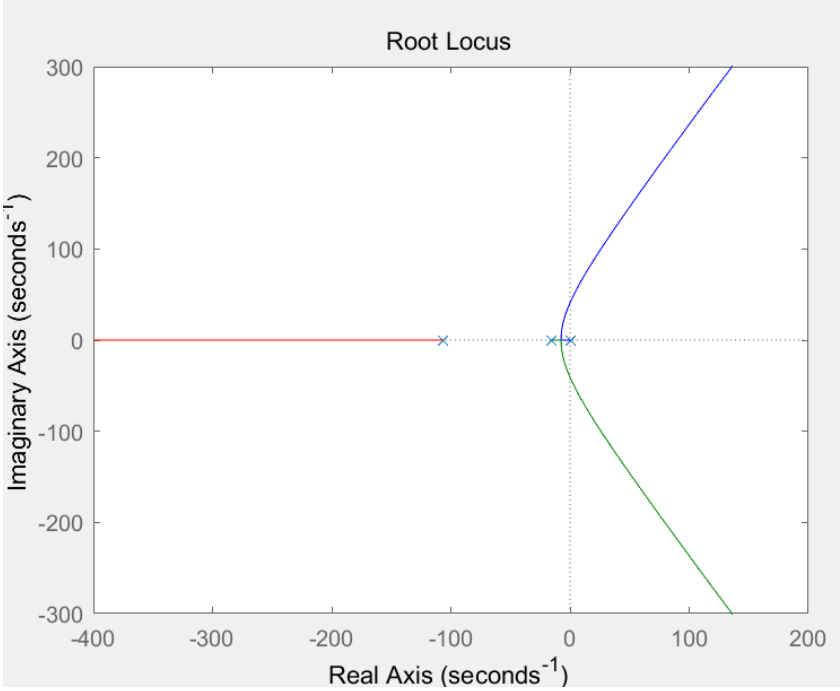
①控制器设计

若以位移为输出，此时开环传递函数$G_0(s)=\frac{262066}{s^3+121.9s^2+1668s}$，Bode 图如下：



此时剪切频率$\omega_c=46.3rad/s$，相角裕度$\gamma=-5°$。、
若输入 20mm 阶跃信号，稳态误差小于 1mm，则系统开环增益$K\geq 20$。
考虑到时域指标已给出，因此采用根轨迹校正的方式。
取阶跃信号$\Delta=5\%$上升时间$t_s=0.08s$，超调量$\sigma=8\%$，则$\xi=0.627$，$\omega_n=59.84rad/s$，则系统期望经过极点$s=-37.52+j46.62$。

系统根轨迹如图。由于系统无法经过期望极点，故需使根轨迹左移，采用超前校正。

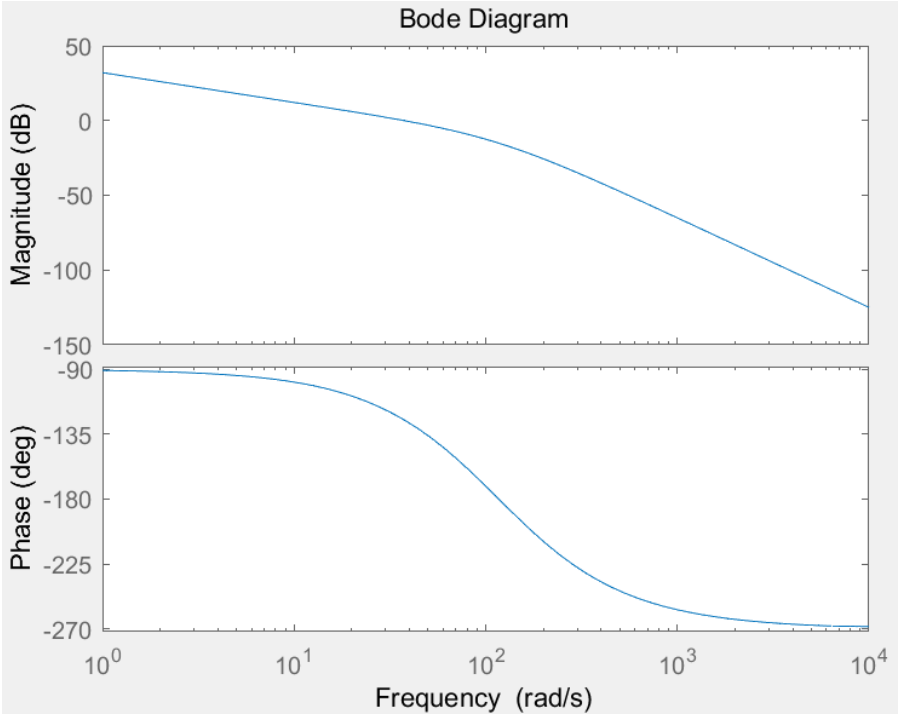


校正环节$\mathrm{G_c}(s)=K_a\frac{s+z_c}{s+p_c}$由以下代码给出：

```
sigma=0.1;
ts=0.08;
K=40;

xi=sqrt((log(1/sigma)/pi)^2/((log(1/sigma)/pi)^2+1));
omega_n=3/ts/xi;

s=omega_n*(-xi+j*sqrt(1-xi^2));

theta=acos(xi);
phi=2*pi-(angle(s)+angle(s+1/0.063639)+angle(s+1/0.0094192));

M=abs(s^3+121.9*s^2+1668*s);
eta=atan(1/(M/(K*1668)/sin(phi)-1/tan(phi)));
delta=pi-eta-theta;

zc=omega_n*sin(eta)/sin(delta);
pc=omega_n*sin(phi+eta)/sin(delta-phi);
Ka=K*pc/zc/262066*1668;
```

得$\mathrm{G_c}(s)=2.1419\frac{s+15.1784}{s+127.6945}$。

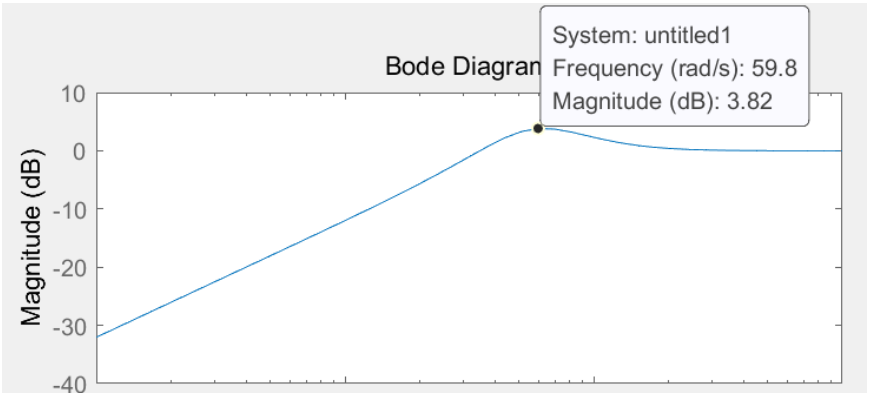
校正后系统 Bode 图如下，相角裕度$\gamma = 55°$，幅值裕度$K_g = 5.754$。



系统采用 Anti-Windup 设计，对$c^{-1} - c_\infty^{-1}$部分进行离散化（$T = 0.005s$）得$\frac{0.2529}{z-0.9269}$。

②鲁棒性验证

对校正后的名义系统，计算敏感度函数$S = \frac{1}{1+GK}$，得$S_{max} = 3.82dB = 1.552$。



利用 Matlab 的 getpvec 函数，可获取参数整定过程中的不确定性。对辨识得到的电机传递函数

$$G_0(s) = \frac{164661}{s^2 + 121.9s + 1668} = \frac{K}{s^2 + a_1 s + a_0}$$

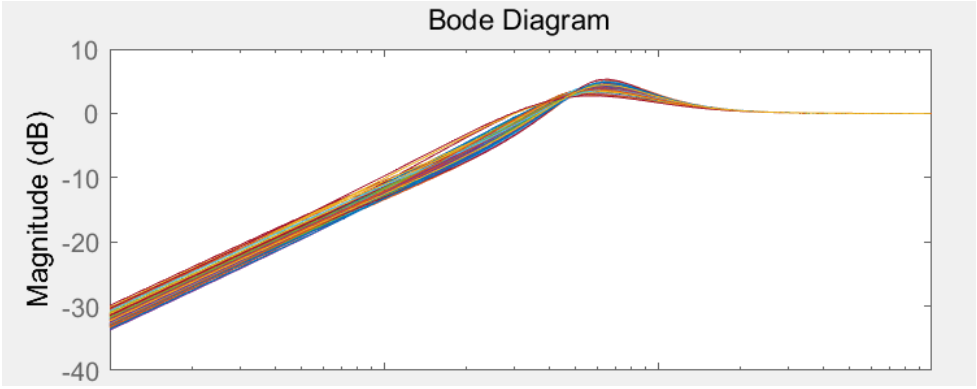
运行以下语句：

[param,uncertain] = getpvec(G0)
返回$uncertain = [11505,\ 8.723,\ 109.89,\ 0]^{\mathrm{T}}$，前 3 项即为$\mathrm{K}, \mathrm{a}_1, a_0$的标准差。

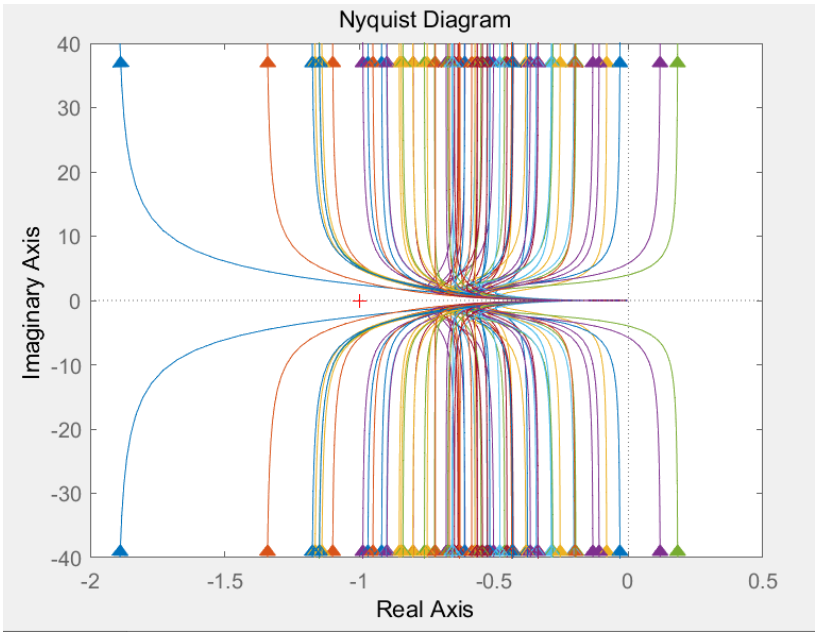
使用随机数方法模拟实际系统的参数不确定性，通过以下语句绘制参数摄动后的灵敏度函数 S：

```
hold on
for i = 1:10
G0 = tf((randn()*11505+164661)*1.591549, [1 (randn()*8.723+121.9)
(randn()*109.89+1668) 0]);
Gc = tf(2.1419*[1 15.1784], [1 127.6945]);
bode(G0*Gc)
end
```

得$\mathrm{S}_{\max} = 5.86\mathrm{dB} = 1.963 < 2.0$。



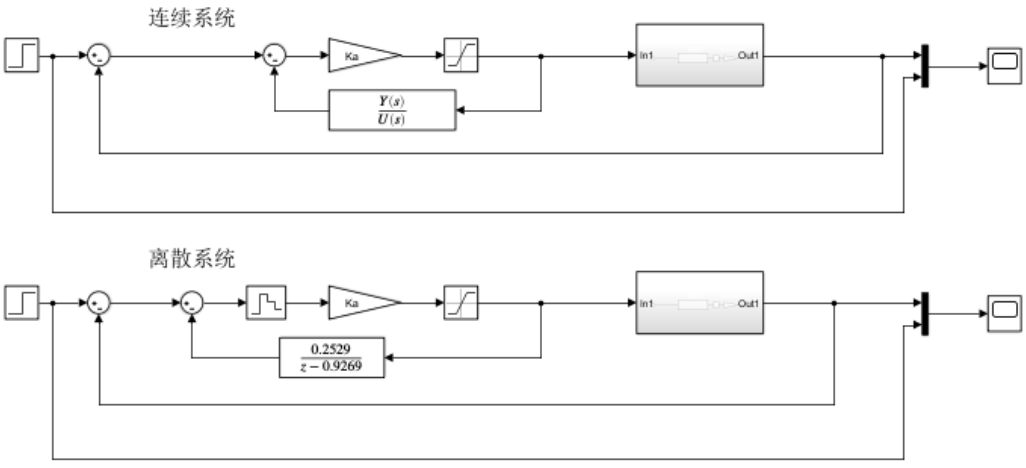
绘制 Nyquist 图，曲线始终不包围(-1,j0)，即系统保持稳定。因此，系统具有一定鲁棒性。



3.4 控制器仿真验证

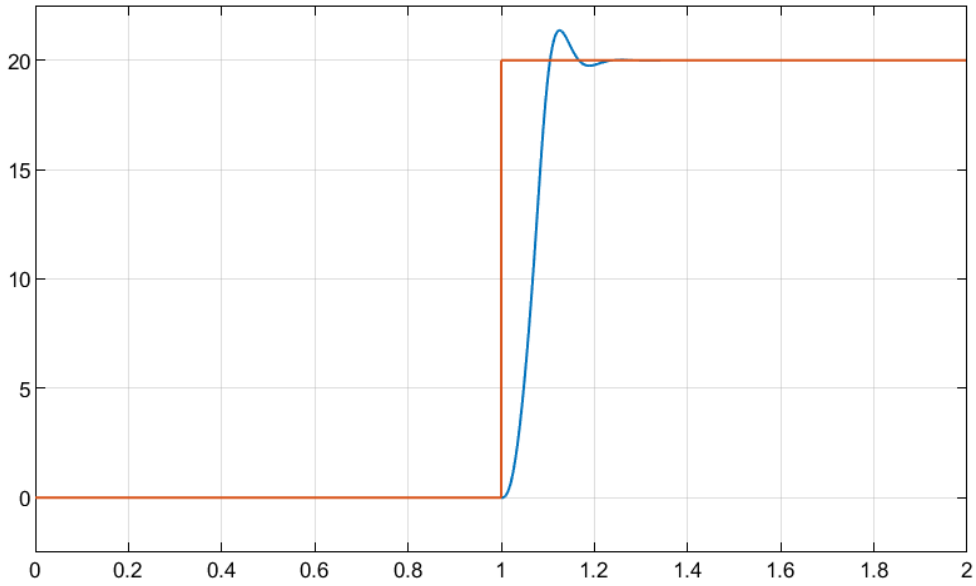
在 simulink 中搭建被控对象和控制算法的仿真系统，对控制算法进行快速的验证。使得仿真环境下，控制系统的任务指标满足要求。记录控制器仿真验证过程。

Simulink 仿真如下：

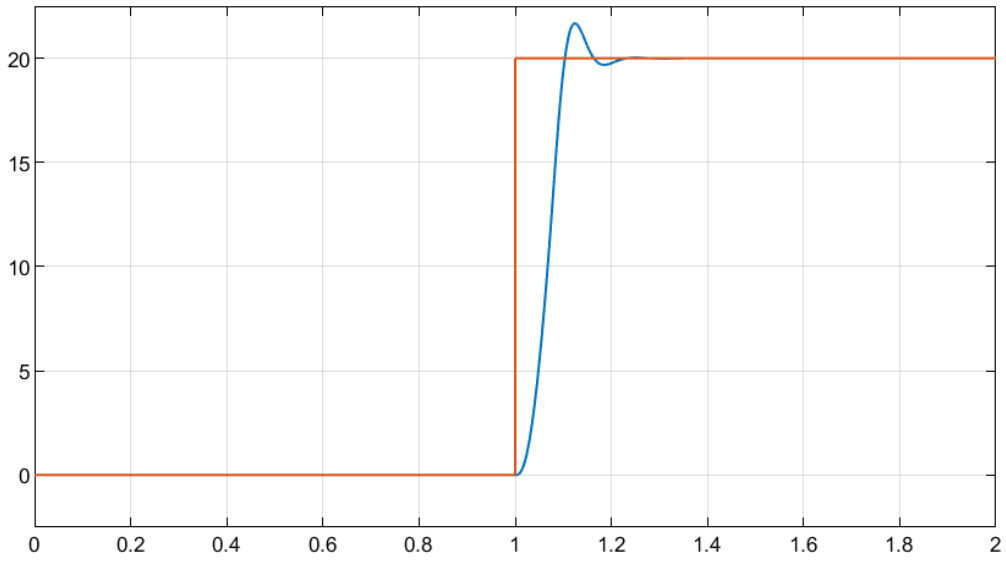


①20mm 阶跃输入：

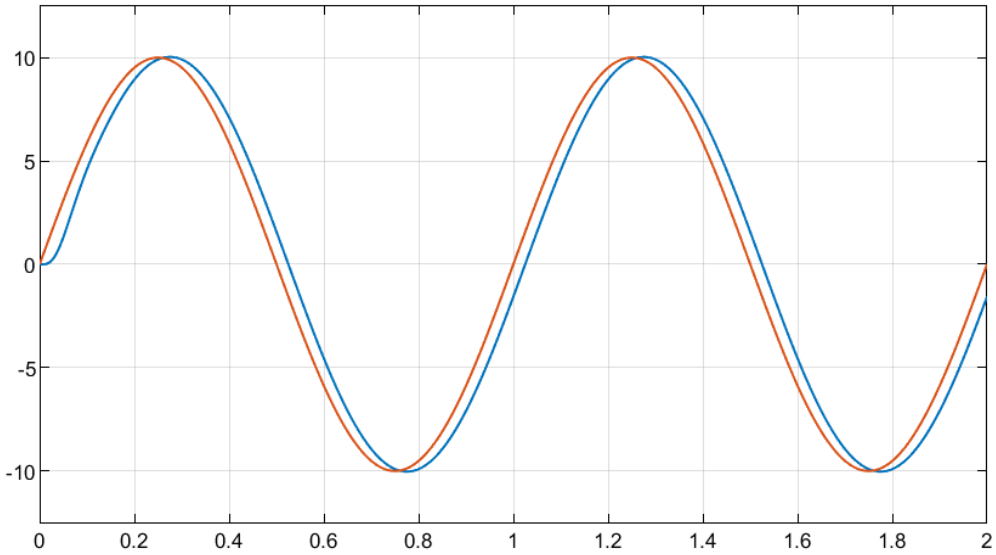
连续系统$t_s = 0.1s, \sigma = 6.7\%, e_{ss} = 0$。



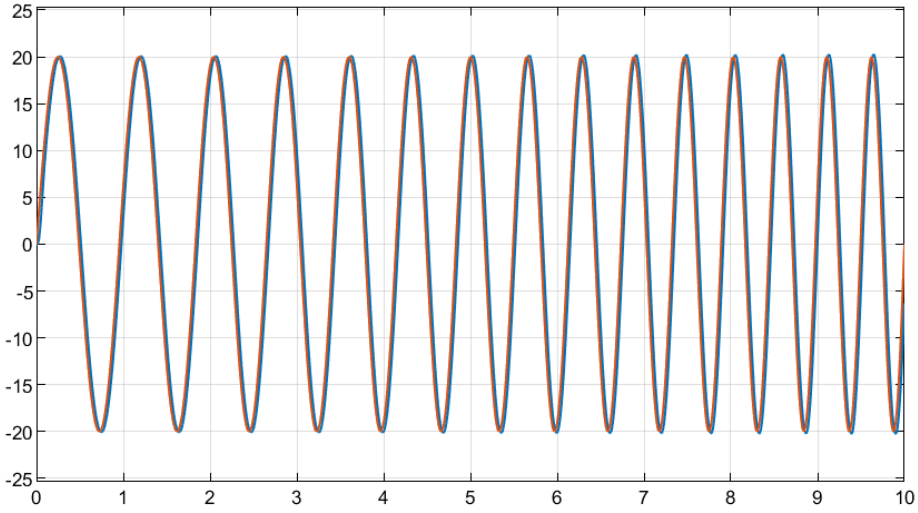
离散系统$t_s = 0.1s, \sigma = 8.4\%, e_{ss} = 0.$



②1Hz　10mm 幅值正弦信号输入：
稳定后输出信号幅值为 10mm，相位滞后1.43°。



③10s　1~2Hz　20mm 幅值扫频信号输入：



3.5 控制程序开发

1）　把上一节中验证好的控制器，转变为离散化的控制器。写出控制器离散化推导过程与结果；若用Matlab推导，则粘贴Matlab代码即可。

```
sys2=tf(pc-zc, Ka*[1 zc]);    % anti-windup c^{-1}-c_inf^{-1}部分离散化
dsys=c2d(sys2,0.005,'zoh')
```

离散化结果：

```
dsys =

    0.2529
  ----------
  z - 0.9269

Sample time: 0.005 seconds
```

2) 将控制器源代码粘贴在下方，并写好注释。

```
    uint32_t t = HAL_GetTick() - wait_time;
    float e = (step_target - encode_32) * 0.005;  // 当前误差, 单位mm
    float w = 0.9269 * w_last + 0.2529 * y_last;  // anti-windup输出
    float y = 2.1419 * (e - w);  // 电流值
    // 饱和函数
    if(y > 3.13) y = 3.13;
    else if(y < -3.13) y = -3.13;
    // 转换输入
    int16_t I_16 = (int16_t)(y * 1596.7f);
    MC_ProgramTorqueRampMotor1(I_16, 0);
    y_last = y;
    w_last = w;
```

3.6 控制系统调试

1) 将阶跃响应测试源代码粘贴在下方，并写好注释。

```
    // 阶跃信号
    if(step_state == 1){
    wait_time = HAL_GetTick();  // 计时
    pos_target = encode_32 + 4000;  // 设置目标位置为当前位置+20mm, TIM7 CNT->光栅尺0.005mm
```

```
    y_last = 0;
    w_last = 0;
    MC_ProgramTorqueRampMotor1(0, 0);
    MC_StartMotor1();
    step_state = 2;
    } else if(step_state == 2){
    uint32_t t = HAL_GetTick() - wait_time;
    float e = (pos_target - encode_32) * 0.005;  // 当前误差, 单位mm
    float w = 0.9269f * w_last + 0.2949f * y_last;
    float y = 2.1419f * (e - w);
    // 饱和函数
    if(y > 3.13f) y = 3.13f;
    else if(y < -3.13f) y = -3.13f;
    // 转换输入
    int16_t I_16 = (int16_t)(y * 1596.7f);
    MC_ProgramTorqueRampMotor1(I_16, 0);

    y_last = y;
    w_last = w;

    // 发送fdb
    sprintf((char*)txbuf, "%f,%f\n", t/1000.0, 20-e);
    HAL_UART_Transmit(&huart5, txbuf, strlen((char*)txbuf),period);

    if(t >= 500){
    // t>0.5s结束
    MC_StopMotor1();
```

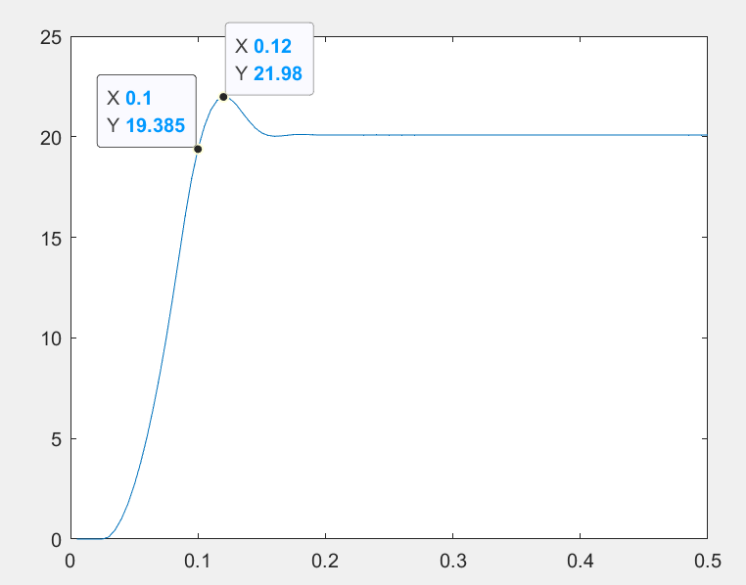
```
    step_state = 0;
    }
```

2) 将扫频跟随测试源代码粘贴在下方，并写好注释。

```
// 正弦信号
if(sine_state == 1){
wait_time = HAL_GetTick();  // 计时
pos_init = encode_32;
y_last = 0;
w_last = 0;
MC_ProgramTorqueRampMotor1(0, 0);
MC_StartMotor1();
sine_state = 2;
} else if(sine_state == 2){
uint32_t t = HAL_GetTick() - wait_time;
pos_target = pos_init + 20 * sin(6.283f * t / 1000.0f) / 0.005f;  // 目标位置
float e = (pos_target - encode_32) * 0.005f;  // 当前误差, 单位mm
float w = 0.9269f * w_last + 0.2949f * y_last;
float y = 2.1419f * (e - w);
// 饱和函数
if(y > 3.13f) y = 3.13f;
else if(y < -3.13f) y = -3.13f;
// 转换输入
int16_t I_16 = (int16_t)(y * 1596.7f);
MC_ProgramTorqueRampMotor1(I_16, 0);
```

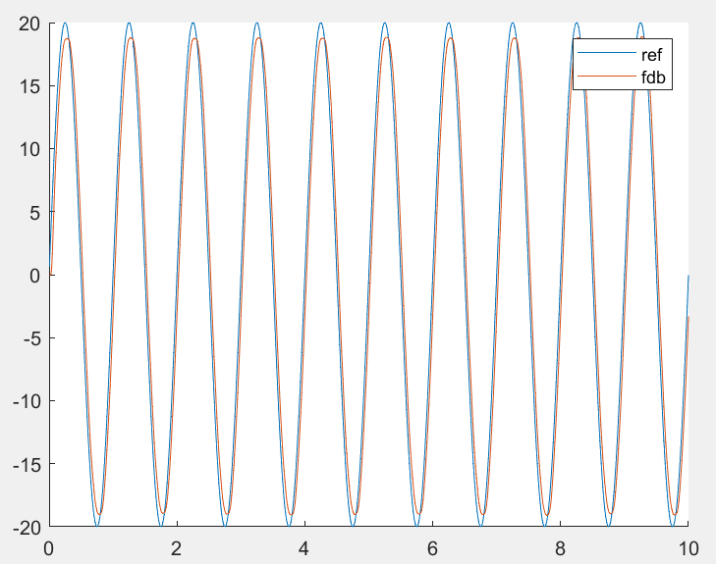
```
    y_last = y;
    w_last = w;

    // 发送fdb
    sprintf((char*)txbuf, "%f,%f,%f\r\n", t/1000.0f,
    (pos_target-pos_init)*0.005, (encode_32-pos_init)*0.005);
    HAL_UART_Transmit(&huart5, txbuf, strlen((char*)txbuf),period);

    if(t >= 10000){
    // t>10s结束
    MC_StopMotor1();
    sine_state = 0;
    }
```

3) 将调优后的阶跃响应曲线粘贴在下方，对于2cm的阶跃响应，要求95%的上升时间不超过0.1s；超调量不大于10%；稳态误差小于1mm；



0.1s 输出上升至 96.9%>95%（即 95%上升时间小于 0.1s），超调量为 9.9%。稳定后位置为 20.085mm，即稳态误差 0.085mm<1mm。

4） 将调优后的扫频跟随曲线粘贴在下方，要求对于幅值为2cm的正弦信号，以-3dB为截止带宽标准的闭环带宽不小于1hz；



对幅值 20mm 1Hz 正弦信号输入，输出幅值 18.99mm，衰减为-0.45dB<-3dB。

5） 写出经过调试后，最终的控制器传递函数。

$$G(s) = \frac{1778.413(0.06588s + 1)}{(0.063639s + 1)(0.0094192s + 1)(0.007831s + 1)}$$

3.7 控制器设计的不足与改进

说说你的控制器设计的不足之处，以及该如何改进？

控制器在设计时未考虑扰动与噪声，因此进行阶跃信号输入时，实际超调量会在$5\% - 12\%$之间变化。

控制器参数按照$\sigma = 8\%, t_s = 0.1s$进行计算，设计时可进一步提高指标要求，使得系统参数发生变化时仍能满足$\sigma \leq 10\%, t_s \leq 0.1s$。

3.8 实验总结

说说自己在整个控制系统设计过程中遇到的主要问题，及最终解决方法。

扫频辨识：

①扫频执行过程中，按下 Key_3，电机不转动。此前，扫频程序正常执行过多次。排查硬件问题：按下 Key_2，回零可正常执行,，且在其他设备按下 Key_3 仍不能执行扫频，因此确定为程序问题。

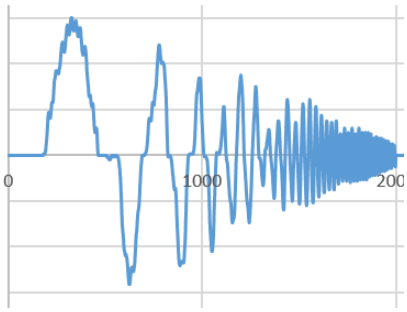
另一方面，由于同一区域的其它代码仍正常执行（可通过串口发送信息），且 MC_ProgramTorqueRampMotor1 处断点有效，因此考虑电机发生了错误使得控制中

断。
检查电机库中存放的 pMCI[0]（电机对象），pMCI[0]->pSTM-> hFaultNow 为 MC_FOC_DURATION，即 FOC 控制频率过高。但将电机执行频率由 2ms 改为 20ms 后，仍无法解决问题。
先前控制代码写在 htim7 的溢出中断中（任务一二正常完成），以控制采样时间相等。此时考虑**将回零、扫频的代码移动至主程序中，并利用 HAL_GetTick()控制执行频率**，htim7 中断仅保留 32 位编码器数值的计算，代码成功运行。

②辨识得到速度输出后，有时会出现过大的跳变值（速度在 190rad/s 左右，附近速度在 20rad/s 之内），使得辨识匹配度在 40%左右。
通过观察电机运动，发现此类跳变大多出现在电机换向处。对 MC_GetMecSpeedAverageMotor1()的返回值进行分析，得到跳变时该函数返回值为 314 的整数倍，由于速度测量采用平均值，且通过电机的编码器实现，因此考虑可能是编码器多圈问题导致。
由于无法对测速编码器原始数据进行分析，最终在代码中对速度返回值进行修正：

```
while(output >= 157) output -= 314;
while(output <= -157) output += 314;
```

此时扫频信号电流幅值 0.6A，匹配度达到 60%，但电机摩擦带来的死区特性非常明显（如下图），因此考虑增大幅值。



但增大幅值后 MC_GetMecSpeedAverageMotor1()的正常返回值会大于 157，导致修正同样会带来误差。
同时，扫频范围初选 0.1Hz~50Hz，第一次向左的运动距离过长，使得幅值增大时电机到达左限位而断电。
综合考虑以上两因素，最终**选取幅值 1A，扫频范围 0.5Hz~10Hz**，再对得到的数据进行修正（绘制出图表后进行修正，代码中不执行此步骤，此时可清晰看出具体发生跳变的数据，且此情形在 1A 时较少出现），辨识匹配度 80%。