**《自动控制实践B》**

**综合实验**

**实验报告**

学院 机器人与先进制造学院

姓名 psp

学号

日期 2025/6/10

目录

[任务一 电机控制库认知 3](#_Toc13852)

[1.1 Workbench编码器对齐参数配置 3](#_Toc3303)

[1.2 速度波形显示界面 3](#_Toc25088)

[1.3 说说你对ST MC SDK5.x（电机控制库）的认知 3](#_Toc25331)

[任务二 按键控制滑台回零 4](#_Toc31058)

[2.1 程序流程图 4](#_Toc23743)

[2.2 功能代码 5](#_Toc10310)

[2.3 实验总结 7](#_Toc26619)

[任务三 控制系统设计 8](#_Toc30451)

[3.1 控制系统建模 8](#_Toc245)

[3.1.1 机械谐振模态分析 8](#_Toc6810)

[3.1.2 被控对象的系统建模 8](#_Toc20828)

[3.2 控制系统辨识 9](#_Toc25079)

[3.2.1 在主控板上实现正弦扫频信号生成算法 9](#_Toc4527)

[3.2.2在主控板上实现扫频辨识功能 9](#_Toc7517)

[3.2.3使用matlab系统辨识工具箱，获得辨识的系统模型 11](#_Toc11140)

[3.3 控制器设计 13](#_Toc10345)

[3.4 控制器仿真验证 18](#_Toc19164)

[3.5控制程序开发 23](#_Toc10892)

[3.6控制系统调试 25](#_Toc16142)

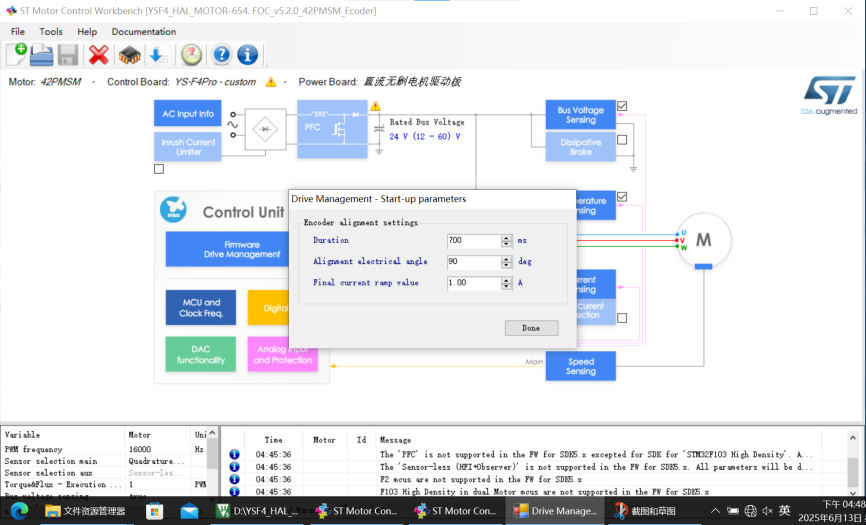
[3.7 控制器设计的不足与改进 31](#_Toc3272)

[3.8 实验总结 31](#_Toc20496)

# 任务一 电机控制库认知

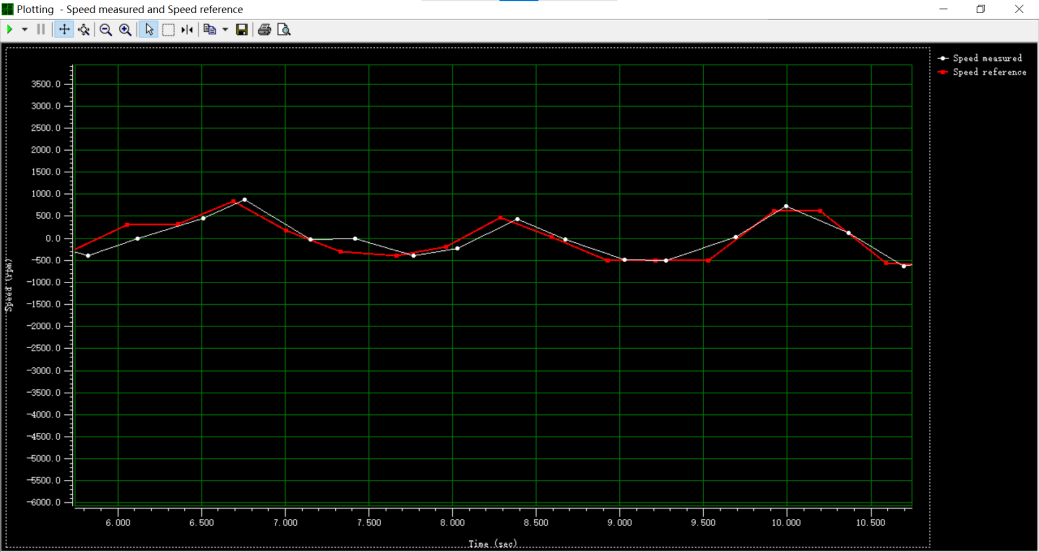
## Workbench编码器对齐参数配置

Workbench软件中参数配置页面如下：



## 速度波形显示界面

速度波形显示界面如下，实测速度能跟随上设定速度：



## 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。

答：ST MC SDK 5.x（电机控制库）是意法半导体（STMicroelectronics）为其微控制器系列设计的一套高级电机控制软件开发工具。该SDK旨在帮助开发者更轻松地创建高效、可靠的电机控制系统，适用于各种类型的电机，包括无刷直流电机（BLDC）、永磁同步电机（PMSM）等，其主要特性包括：广泛的适用性：支持多种电机类型和控制算法，如矢量控制（FOC）、标量控制等；优化的性能：通过高度优化的代码实现高效的电机控制，减少资源消耗，提高系统响应速度；易于使用的API：提供简单直观的应用程序接口，简化了开发过程，使得用户能够快速上手；丰富的示例代码：包含多个示例项目，展示了如何使用SDK进行电机控制应用的开发。

该软件开发工具包是专门为加速产品上市时间而设计的，无论是对于新手还是有经验的开发者来说，都是一个强大的辅助工具。通过利用ST MC SDK 5.x，开发者可以专注于其应用的独特需求，而不必担心底层电机控制算法的实现细节。

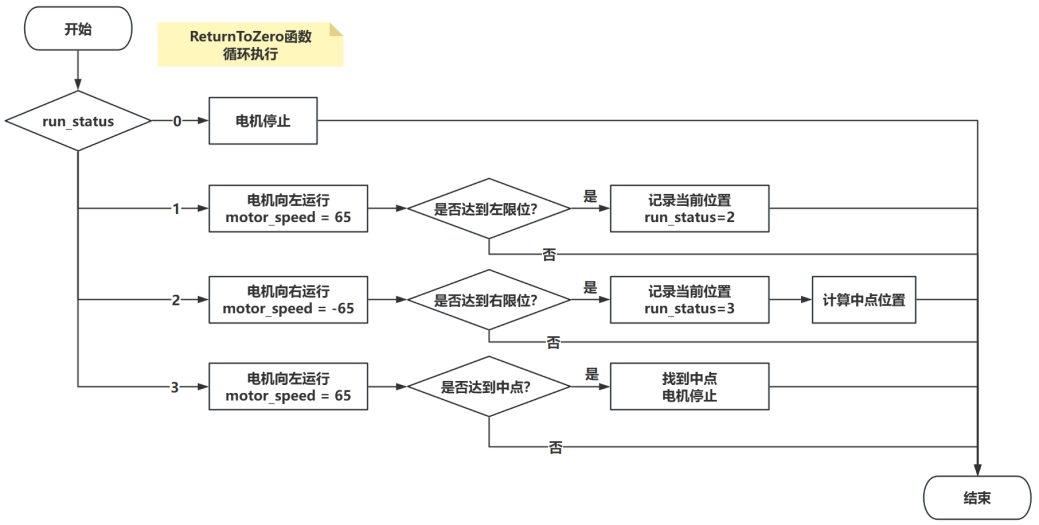
# 任务二 按键控制滑台回零

## 2.1 程序流程图

画出滑台回零程序流程图。

答：流程图如下：包括主函数、按键中断部分、ReturnToZero函数部分

## 



## 2.2 功能代码

**初始化部分**：为了在调试时将变量加入检测窗口中，将它们在全局中定义

1. ***/\* USER CODE BEGIN 0 \*/***
2. **uint8\_t is\_finish\_flag = 0;**
3. **int8\_t motor\_target\_speed = 0;**
4. **uint8\_t run\_status = 0;**
5. **int32\_t encoder\_value = 0;**
6. **int32\_t encoder\_value\_left\_edge = 0;**
7. **int32\_t encoder\_value\_mid = 0;**
8. **int32\_t encoder\_value\_right\_edge = 0;**
9. **int32\_t gap = 0;**
10. **float cur\_position = 0;**
11. **int32\_t ENCODER\_RULER\_TIM\_PERIOD = 65535;**
12. **int32\_t encoder\_ruler\_overflow\_count = 0;**
13. **void ReturnToZero(void);**
14. **short MyAbs(short);**
15. **GPIO\_PinState state1 = GPIO\_PIN\_SET;**
16. **GPIO\_PinState state0 = GPIO\_PIN\_SET;**
17. ***/\* USER CODE END 0 \*/***

**主函数部分：**

1. ***/\* USER CODE BEGIN WHILE \*/***
2. **while (1) {**
3. ***/\* USER CODE END WHILE \*/***
4. ***/\* USER CODE BEGIN 3 \*/***
5. **encoder\_value = \_\_HAL\_TIM\_GET\_COUNTER(&htim3);  *// 获取TIM3编码器值***
6. **int32\_t Period = ENCODER\_RULER\_TIM\_PERIOD + 1;**
7. **ReturnToZero();                                  *// 调用回零函数***
8. **if (is\_finish\_flag == 0 && motor\_target\_speed != 0) {      *// 若未寻得中点***
9. **MC\_ProgramSpeedRampMotor1(motor\_target\_speed, 5);   *// 设置电机速度，速度在ReturnToZero函数中指定***
10. **MC\_StartMotor1();                            *// 启动电机***
11. **}**
12. **cur\_position = (encoder\_value + Period \* encoder\_ruler\_overflow\_count) \* 0.005f;  *// 将编码器值转化为位置***
13. **printf("%f\r\n", cur\_position);**
14. **}**
15. ***/\* USER CODE END 3 \*/***

**USER CODE BEGIN 4部分：**

设电机的需求速度为60

1. ***/\* USER CODE BEGIN 4 \*/***
2. **void HAL\_TIM\_PeriodElapsedCallback(TIM\_HandleTypeDef\* htim) {**
3. **if (htim->Instance == TIM3) {**
4. **if (\_\_HAL\_TIM\_IS\_TIM\_COUNTING\_DOWN(&htim3)) {**
5. **encoder\_ruler\_overflow\_count--;**
6. **} else {**
7. **encoder\_ruler\_overflow\_count++;**
8. **}**
9. **}**
10. **}**
11. **void HAL\_GPIO\_EXTI\_Callback(uint16\_t GPIO\_Pin) {**
12. **if (GPIO\_Pin == GPIO\_PIN\_1) {**
13. **for (uint32\_t i = 0; i < 500000; ++i);  *// 软件消抖***
14. **if (HAL\_GPIO\_ReadPin(GPIOE, GPIO\_PIN\_1) == GPIO\_PIN\_RESET) {**
15. **run\_status = 1;  *// 回零开始运行***
16. **is\_finish\_flag = 0;**
17. **}**
18. **}**
19. **}**
20. **int fputc(int ch, FILE\* f) {**
21. **HAL\_UART\_Transmit(&huart5, (uint8\_t\*)&ch, 1, 0xffff);**
22. **return ch;**
23. **}**
24. **void ReturnToZero(void) {**
25. **if (run\_status == 1) {**
26. **motor\_target\_speed = 60;  *// 朝一方向（左）运动***
27. **}**
28. **state1 = HAL\_GPIO\_ReadPin(GPIOG, GPIO\_PIN\_1);**
29. **if (state1 == GPIO\_PIN\_RESET && run\_status == 1) {  *// 左限位***
30. **run\_status = 2;**
31. **motor\_target\_speed = 0;**
32. **MC\_StopMotor1();**
33. **encoder\_value\_left\_edge = encoder\_value;**
34. **motor\_target\_speed = -60;  *// 朝另一方向（右）运动***
35. **}**
36. **state0 = HAL\_GPIO\_ReadPin(GPIOG, GPIO\_PIN\_0);**
37. **if (state0 == GPIO\_PIN\_RESET && run\_status == 2) {  *// 右限位***
38. **run\_status = 3;**
39. **motor\_target\_speed = 0;**
40. **MC\_StopMotor1();**
41. **encoder\_value\_right\_edge = encoder\_value;**
42. **encoder\_value\_mid = encoder\_value\_left\_edge - (encoder\_value\_left\_edge - encoder\_value\_right\_edge) / 2;**
43. **motor\_target\_speed = 60;  *// 又朝一方向（左）运动***
44. **}**
45. **gap = MyAbs(encoder\_value - encoder\_value\_mid);  *// 计算中点与当前编码器值的差异***
46. **if (run\_status == 3 && gap < 5) {**
47. **motor\_target\_speed = 0;**
48. **MC\_ProgramSpeedRampMotor1(0, 1);**
49. **MC\_StartMotor1();**
50. **run\_status = 0;**
51. **is\_finish\_flag = 1;  *// 找到中点***
52. **}**
53. **}**
54. **short MyAbs(short value) {**
55. **if (value < 0) {**
56. **return -value;**
57. **}**
58. **return value;**
59. **}**
60. ***/\* USER CODE END 4 \*/***

## 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。

答：实验结论：我们的实验结论显示，运行上述代码后，滑台成功实现了预期目标，即先向左运动至限位点，记录当前位置后转向右侧移动直至另一限位点，再次记录位置后，根据这两点的位置信息计算出中心点，并将滑台准确停止于该位置。

问题：实验中发现了串口软件能够通过串口连接开发板，但是无法获得开发板通过串口回传的消息。为了解决这个问题，我们首先进行了全面的排查，包括检查物理连接、电缆完整性及两端接口的匹配性等基础问题。经过检查得知是代码中串口设置错误，应使用。

# 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

请参考理论课件《第四章 控制系统的设计约束（2）》中的4.3.2章节中关于结构模态的内容，以及上一个小结对控制系统的介绍，计算联轴器的谐振频率；与控制系统要求的带宽对比，确定机械谐振模态是否可以忽略；写出计算过程。

解：滑台的转动惯量为

丝杆转动惯量为

电机转动惯量为

系统总转动惯量为

已知扭矩刚性为

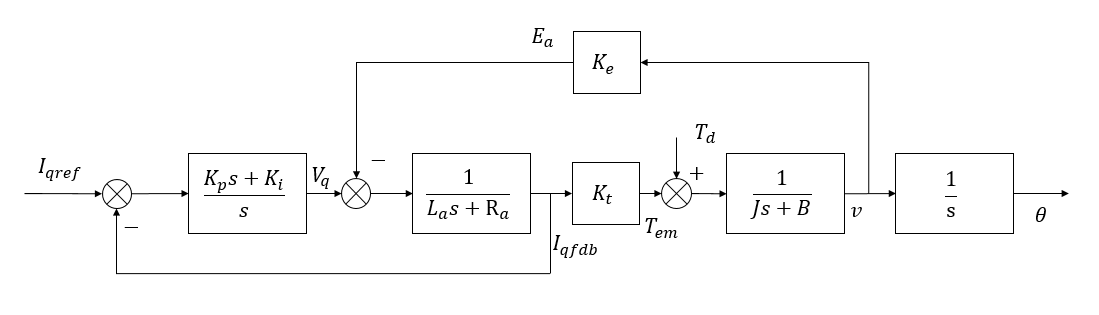
故谐振频率为，

结论：谐振频率远大于系统输入的正弦信号频率以及扫频信号的频率（1Hz），作为高频未建模动态对系统影响不大，故机械谐振模态可以忽略。

### 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。

解：经过分析和系统建模，被控对象（电机及FOC控制系统）方框图如下：



经过模型降阶与非线性处理、平均化等，可以得出系统的传递函数简化为：



## 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。.

1、**初始化函数**，用于设置扫频信号的信息：

1. **int init\_my\_sweep(my\_sweep\_t \*sweep, unsigned int t\_0, unsigned int t\_01, float f0, float f1, float A) {**
2. **if ((t\_01 == 0) || (f0 <= 0.0f) || (f1 <= 0.0f) || (f0 == f1) || (A == 0) || (!sweep)) {**
3. **return -1; */\*非法入参\*/***
4. **}**
5. **sweep->t\_0 = t\_0;**
6. **sweep->t\_01 = t\_01;**
7. **sweep->f0 = f0;**
8. **sweep->f1 = f1;**
9. **sweep->A = A;**
10. ***/\* start add code here \*/***
11. **sweep->k = exp(1 / (0.001 \* t\_01) \* log(f1 / f0)); */\*计算指数函数的底数 k，注意时间的单位要转换为秒\*/***
12. **sweep->p = 2 \* 3.1415926 \* f0 / (log(sweep->k));   */\*计算系数 p, 注意单位转换\*/***
13. ***/\* end add code here \*/***
14. **return 0;**
15. **}**

2、**获取正弦扫频信号的函数**

1. **float run\_my\_sweep(my\_sweep\_t \*sweep, unsigned int t\_now) {**
2. **float t = 0.0f;  *// 相对时间 t***
3. **float y = 0.0f;  *// 扫频信号***
4. **if (!sweep) {**
5. **return 0.0f; */\*非法入参\*/***
6. **}**
7. **if (t\_now < sweep->t\_0) {**
8. **return 0.0f; */\*时间还未得到\*/***
9. **}**
10. **t = (t\_now - sweep->t\_0) % sweep->t\_01; */\*通过求余操作实现，周期性扫频的过程\*/***
11. **t = t \* 0.001f;                         */\*将单位转换为 s\*/***
12. ***/\* start add your code here \*/***
13. **y = sweep->A \* sin(sweep->p \* (pow(sweep->k, t) - 1));**
14. ***/\* end add your code here \*/***
15. **return y;**
16. **}**

### 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。

1、**实现扫频辨识功能**的代码：

1. **void function\_sweep\_identification(void) {**
2. **int16\_t sweep\_input = 0;**
3. **int16\_t sweep\_output = 0;**
4. **uint32\_t sys\_tick = 0;**
5. **static uint32\_t init\_flag = 0;**
6. **static uint32\_t last\_sys\_tick = 0;**
7. ***// 频率在 10s 内 ，从 0.5hz 变化到 10hz，幅度为 1500 digit current***
8. **uint32\_t t\_period\_ms = 10 \* 1000;  *// 10s***
9. **float f0 = 0.5;**
10. **float f1 = 10;**
11. **float Amp = 1500.0f;**
12. **float time = 0.0f;**
13. **sys\_tick = HAL\_GetTick();       *// 获取当前时刻，单位 ms***
14. **time = 0.001f \* sys\_tick;       *// 单位 s***
15. **if (last\_sys\_tick != sys\_tick)  *// 如果当前时刻发生了变化，这个条件每 ms 都会成立一次***
16. **{**
17. **last\_sys\_tick = sys\_tick;**
18. **if (sys\_tick % 10 == 0)  *// 通过 % 把频率从 1000hz 降低到 100hz，即每 10ms 发生一次***
19. **{**
20. ***// 初始化扫频配置***
21. **if (init\_flag == 0) {**
22. **init\_my\_sweep(&sweep, sys\_tick, t\_period\_ms, f0, f1, Amp);**
23. ***// printf("sweep-init:k=%.5f,p=%.5f\r\n",(float)sweep.k,(float)sweep.p);***
24. **init\_flag = 1;**
25. **}**
26. ***// 获取正弦扫频信号***
27. **sweep\_input = (int16\_t)run\_my\_sweep(&sweep, sys\_tick);**
28. ***// 将正弦扫频信号输入到 ST MC SDK 的力矩控制 API 中***
29. **MC\_ProgramTorqueRampMotor1(sweep\_input, 0);**
30. **MC\_StartMotor1();**
31. ***// 获取丝杆的转速信息，单位为 0.1hz***
32. **sweep\_output = MC\_GetMecSpeedAverageMotor1();**
33. ***// 把时间，input, output 发送到 matlab***
34. **send\_data\_2\_matlab(time, (float)sweep\_input, (float)sweep\_output);**
35. **}**
36. **}**
37. **}**
38. **用于将数据发送到 MATLAB 上位机的代码**是：

**注意要设置为串口5，否则无法发送**

1. ***/\****
2. **函数：send\_data\_2\_matlab**
3. **功能：往 matlab 发送三个浮点数**
4. **输入：三个浮点数**
5. **输出：无**
6. **\*/**
7. **void send\_data\_2\_matlab(float data1, float data2, float data3) {**
8. **frame\_matlab\_t frame = {0};**
9. ***/\* 填充帧头 \*/***
10. **frame.start\_flag = 0xAA;**
11. **frame.frame\_len = sizeof(frame);**
12. **frame.header\_check = get\_uint8\_sum\_check((uint8\_t \*)&frame, 2);**
13. ***/\* 填充数据 \*/***
14. **memcpy((uint8\_t \*)&frame.data\_buf[0], (uint8\_t \*)&data1, 4);**
15. **memcpy((uint8\_t \*)&frame.data\_buf[4], (uint8\_t \*)&data2, 4);**
16. **memcpy((uint8\_t \*)&frame.data\_buf[8], (uint8\_t \*)&data3, 4);**
17. ***/\* 计算数据求和值,用于接收方校验数据的完好性 \*/***
18. **frame.frame\_check = get\_uint8\_sum\_check((uint8\_t \*)&frame, frame.frame\_len - 1);**
19. ***/\* 通过 串口 5 发送到电脑 \*/***
20. **HAL\_UART\_Transmit(&huart5, (uint8\_t \*)&frame, frame.frame\_len, 0xffff);**
21. **}**
22. 将扫频辨识过程中Matlab采集波形保存并粘贴在下方。

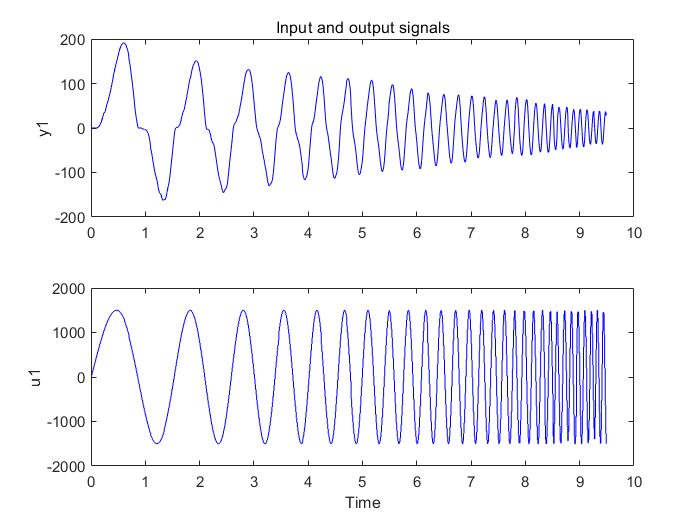
如**下图**所示，蓝色为输入正弦指令信号，橙色为实际输出信号：



### 3.2.3使用matlab系统辨识工具箱，获得辨识的系统模型

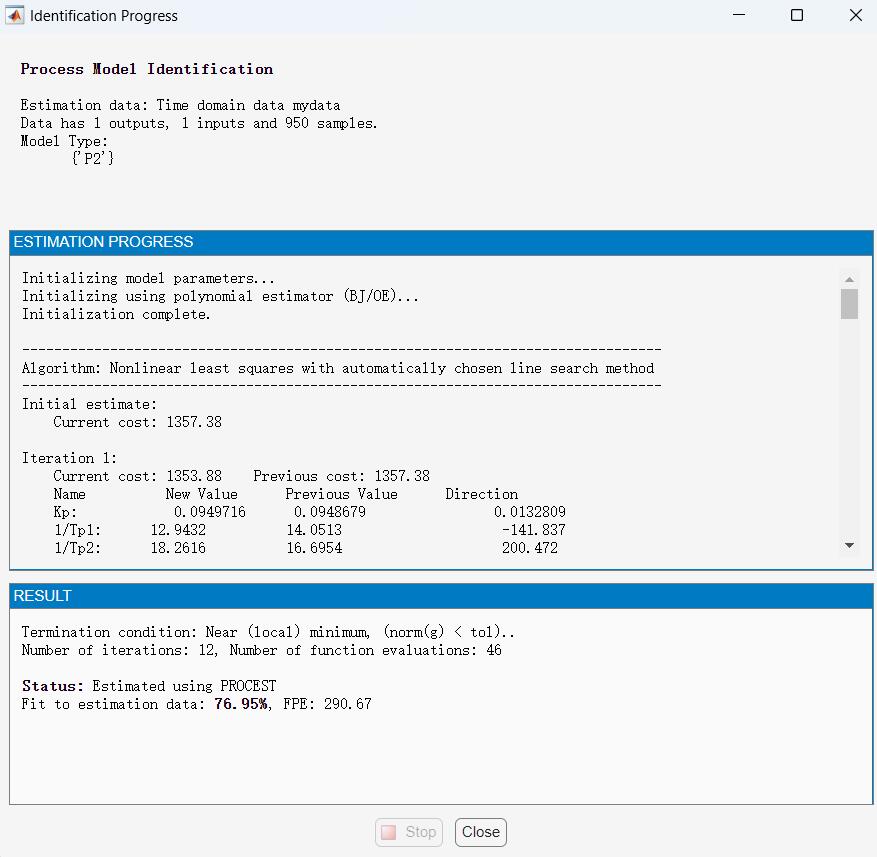
1. 将输入输出数据导入Matlab系统辨识工具箱后，Time Plot查看输入输出曲线，将曲线截图粘贴在下方。

输入输出曲线如下：其中上图为输出曲线，下图为输入曲线

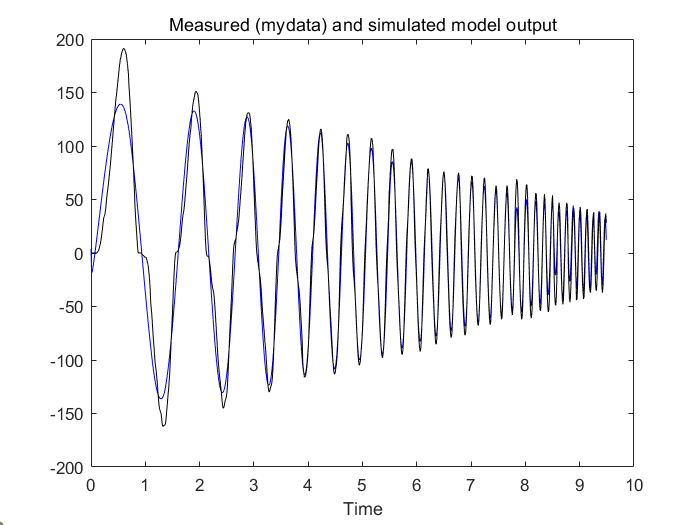


1. 参数辨识结束后，将参数辨识计算过程截图粘贴在下方。

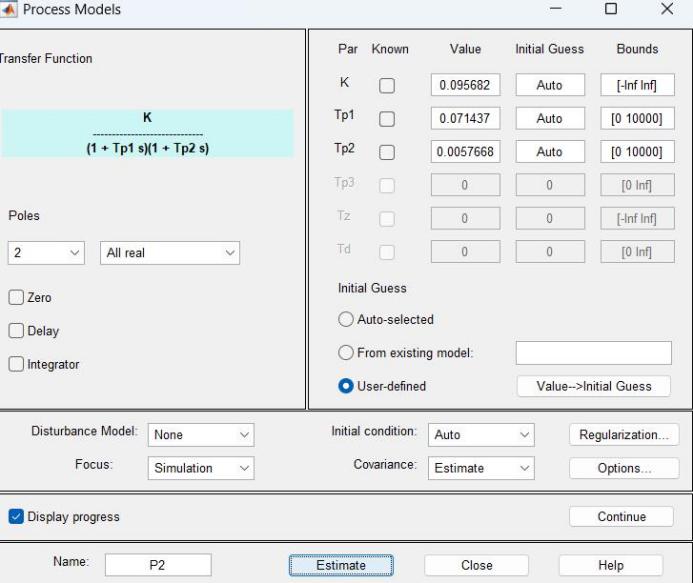
在matlab的控制窗口中输入systemIdentification，打开系统辨识工具箱，参数识别计算过程如下：



1. 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。



1. 将参数辨识结果截图并粘贴在下方。



吻合度：**76.95%**，符合要求

1. 写出系统开环传递函数。

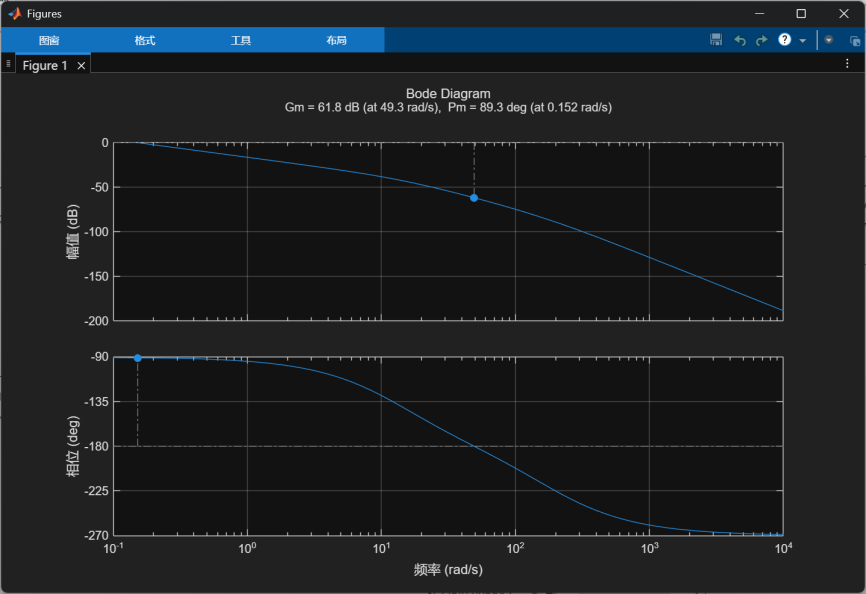


加入额外的环节（积分环节和角度转位移的环节）后，开环传函为：



## 3.3 控制器设计

原开环传函的Bode图如下：

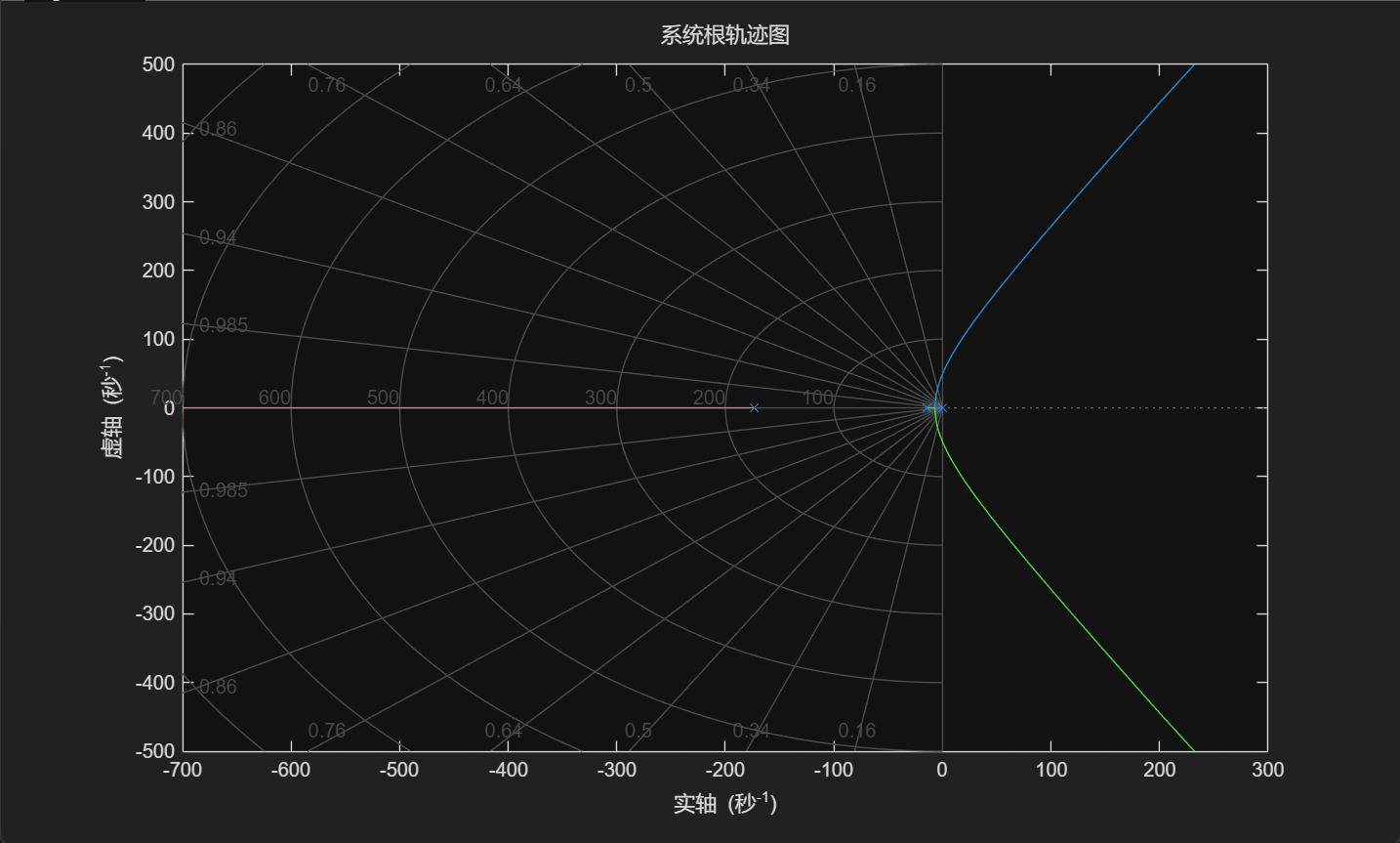


使用的代码为（其中变量G表示开环传函）：

1. **% 清除工作区和命令窗口**
2. **clear; clc; close all;**
3. **s = tf('s');**
4. **G = 0.1522826 / (s \* (0.071437\*s + 1) \* (0.0057668\*s + 1));**
5. **figure;**
6. **margin(G);**
7. **grid on;**

可见系统剪切频率极小，动态特性极差系统的带宽也不足（-3dB带约为0.1rad/s），输入 1Hz 正弦波将会被严重削去。

原系统的根轨迹如下，稳定程度很差，期望的主导极点离原系统根轨迹很远。



使用的代码为（其中变量G表示开环传函）：

1. **clear; clc; close all;**
2. **s = tf('s');**
3. **G = 0.1522826 / (s \* (0.071437\*s + 1) \* (0.0057668\*s + 1));**
4. **figure;**
5. **rlocus(G); % 绘制根轨迹**
6. **title('系统根轨迹图');**
7. **xlabel('实轴');**
8. **ylabel('虚轴');**
9. **grid on;**

下面我们用三种方法设计控制器，并通过仿真和实机实验对比其效果。

**设计方法一：**

设计**串联PD控制律**，目标是将剪切频率增大，并保持相位裕度在30°至 60°之间，形式如下：



其中N为滤波器系数，N趋于无穷时，第二项趋于理想的微分环节。N取足够大即可，为简便此处取为（这也是 simulink 仿真的默认值）。

附加此 PD 控制律后，系统的开环传递函数应为



则 PD 控制器给该传函附加了一个零点，另附加了一个频率较高（100）的极点，在系统高频段，提高抗噪声能力，对于动态性能影响相对较小。

系统为I型，理论上对于单位阶跃响应的误差为0，不妨取P = 500。求取系统的剪切频率。系统开环对数幅频特性如下：

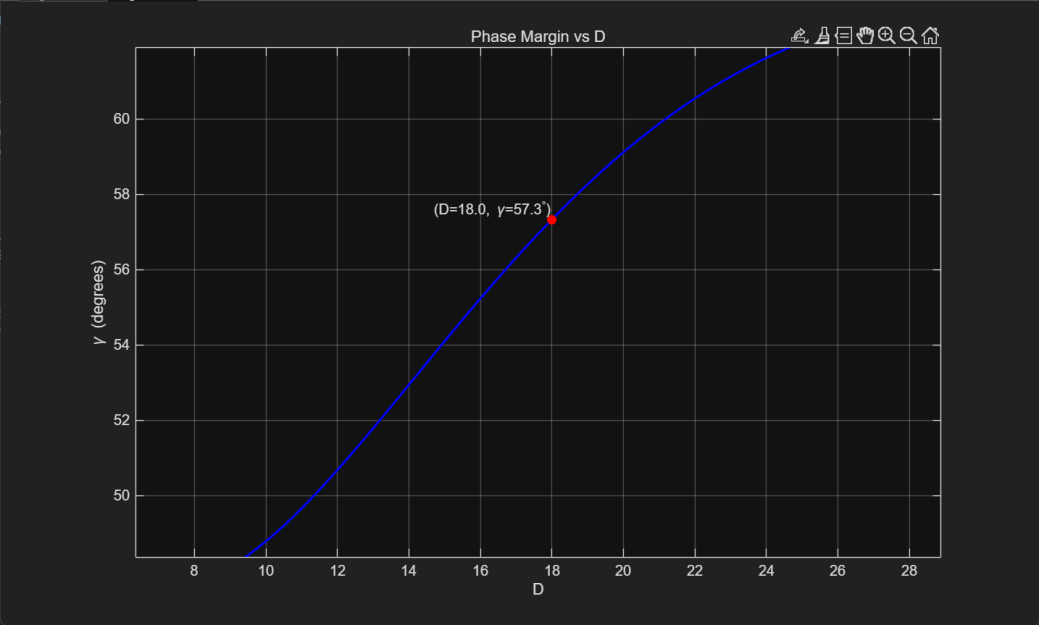


可知剪切频率落在第2段时，D较小，剪切频率满足；剪切频率落在第3段时，D较大，满足，即。

求取相角裕度的公式：



利用此式绘制出相角裕度的图线，可得：

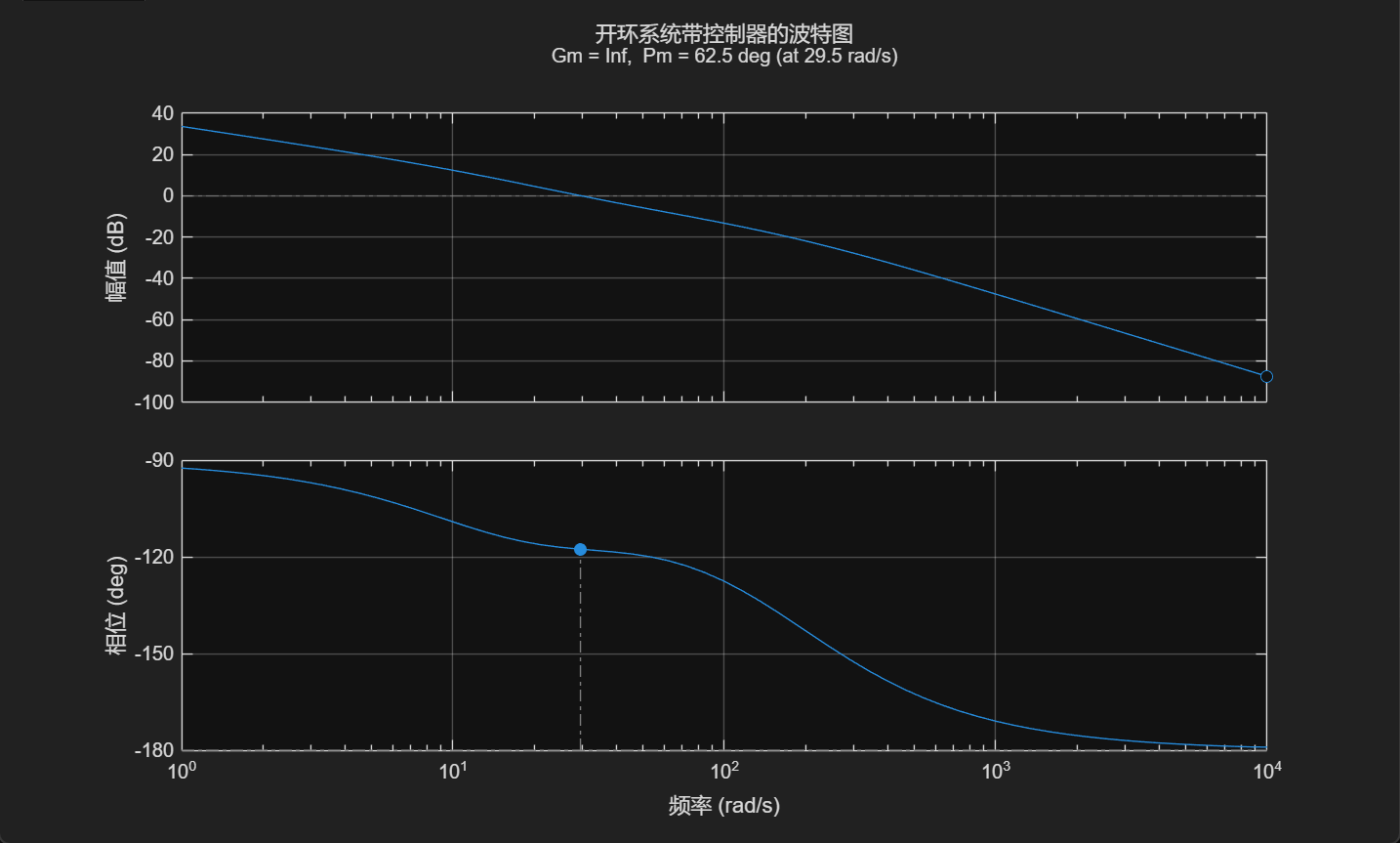


使用的代码为：

1. **% 参数设置**
2. **P = 500;**
3. **D\_values = 0:0.1:30; % D 从 0 到 30，步长 0.1**
4. **% 初始化结果数组**
5. **gamma\_values = zeros(size(D\_values));**
6. **for i = 1:length(D\_values)**
7. **D = D\_values(i);**
8. **% 计算剪切频率 omega\_c**
9. **omega\_c = 0.0115 \* (P + 100\*D);**
10. **% 计算相角裕度 gamma**
11. **term1 = atand(0.095682 \* omega\_c);**
12. **term2 = atand(((P + 100\*D)/(100\*P))\*omega\_c);**
13. **term3 = atand(0.01 \* omega\_c);**
14. **gamma = 90 - term1 + term2 - term3;**
15. **gamma\_values(i) = gamma;**
16. **end**
17. **figure;**
18. **plot(D\_values, gamma\_values, 'b-', 'LineWidth', 1.5);**
19. **xlabel('D');**
20. **ylabel('\gamma (degrees)');**
21. **title('Phase Margin vs D');**
22. **grid on;**
23. **% 找到 D=18 时的 gamma 值**
24. **index\_at\_18 = find(D\_values == 18);**
25. **gamma\_at\_18 = gamma\_values(index\_at\_18);**
26. **hold on;**
27. **plot(18, gamma\_at\_18, 'ro', 'MarkerFaceColor', 'r');**
28. **text(18, gamma\_at\_18, sprintf(' (D=%.1f, \\gamma=%.1f^{\\circ})', 18, gamma\_at\_18),...**
29. **'VerticalAlignment','bottom', 'HorizontalAlignment','right');**

相角裕度在设计在30°到60°之间比较合适。考虑到此系统对于快速性的要求较高，所以取D=18，则可由图得相角裕度为60°左右。

再绘制出时的精确Bode图并验算剪切频率和相位裕度：



由校正后系统的Bode图可知，校正后系统的动态性能得到了显著改善，剪切频率和相角裕度为：，和设计时的要求比较接近。的带宽也约为，远远超过输入正弦信号的频率，满足了要求。下一节将通过Simulink仿真来分析系统的阶跃响应和正弦响应。

使用的代码为：

1. **clear; clc; close all;**
2. **% 控制器参数**
3. **P = 500;**
4. **D = 18;**
5. **I = 0;**
6. **s = tf('s');**
7. **% 被控对象**
8. **G = 0.095682 / (s \* (0.071437\*s + 1) \* (0.0057668\*s + 1));**
9. **% PID 控制器**
10. **C = P + I/s + D\*s;**
11. **% 绘制波特图**
12. **figure;**
13. **margin(G \* C);**
14. **grid on;**
15. **title('开环系统带控制器的波特图');**

**设计方法二：**

使用MATLAB语言进行根轨迹设计。为给定的开环传递函数设计一个超前控制器，以满足系统的性能指标：调节时间和稳态误差。

设计步骤：

1. 根据给定的调节时间和稳态误差，计算出期望的闭环极点（包括阻尼比、自然频率等）
2. 计算原系统在该极点处的相位滞后（phi）和幅值（M）
3. 使用几何法（极点-零点配置）设计一个超前控制器，使其能在该极点处提供所需的相位补偿
4. 计算控制器的零点、极点和增益，使闭环极点落在期望位置

使用代码：

1. **% 清除工作区和命令窗口**
2. **clear; clc; close all;**
3. **sigma=0.01; % 最大允许超调量**
4. **ts=0.05;  % 期望的调节时间**
5. **K=60;  % 开环增益，是为了满足稳态误差要求**
6. **%% 根据性能指标设定期望闭环极点**
7. **% 计算阻尼比和自然频率**
8. **xi = sqrt((log(1/sigma)/pi)^2/((log(1/sigma)/pi)^2+1));**
9. **omega\_n = 3/ts/xi;**
10. **% 构造期望闭环极点**
11. **s = omega\_n\*(-xi+j\*sqrt(1-xi^2));**
12. **theta = acos(xi);**
13. **phi = 2\*pi-(angle(s)+angle(s+1/0.071437)+angle(s+1/0.0057668));**
14. **%% 设计控制器参数**
15. **M = abs(s^3+187.3968 \*s^2+2.4274e+03\*s);**
16. **eta = atan(1/(M/(K\*2.4274e+03)/sin(phi)-1/tan(phi)));**
17. **delta =pi-eta-theta;**
18. **zc = omega\_n\*sin(eta)/sin(delta);**
19. **pc = omega\_n\*sin(phi+eta)/sin(delta-phi);**
20. **Ka = K\*pc/zc/369.6961\*2.4274e+03;**
21. **% 显示结果**
22. **fprintf('zc=%.4f\n',zc);**
23. **fprintf('pc=%.4f\n',pc);**
24. **fprintf('Ka=%.4f\n',Ka);**

计算结果：

1. **zc=24.5324**
2. **pc=71.1270**
3. **Ka=1.1422e+03**

下一节将通过Simulink仿真来分析系统的阶跃响应和正弦响应。

**设计方法三：**

**期望频率特性法**

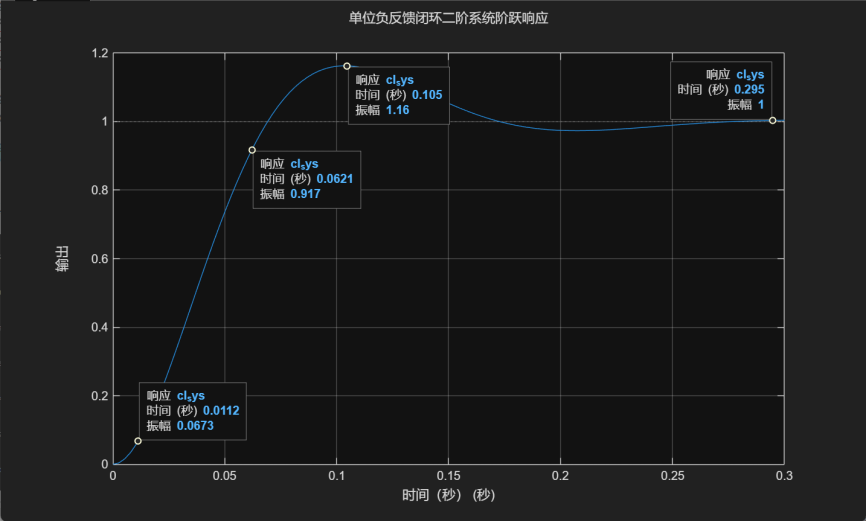
由于原系统建模成二阶系统，设计二阶的期望传函为：

所得的期望开环传递函数为：

使用下面的代码：

1. **% 二阶系统闭环阶跃响应仿真**
2. **% 定义系统参数**
3. **wn = 35;**
4. **K = wn^2;**
5. **a = 0;**
6. **zeta = 0.50;**
7. **% 构造开环传递函数 G(s) = K / [(s + a)(s^2 + 2\*zeta\*wn\*s + wn^2)]**
8. **num = K;**
9. **den = conv([1 a], [1 2\*zeta\*wn]);  % 分母 = (s + a)(s^2 + 2\*zeta\*wn\*s)**
10. **open\_sys = tf(num, den);  % 开环系统 G(s)**
11. **% 打印开环系统传递函数**
12. **fprintf('--- 开环系统传递函数 G(s) ---\n');**
13. **% 构造单位负反馈闭环系统：T(s) = G(s) / (1 + G(s))**
14. **cl\_sys = feedback(open\_sys, 1);**
15. **% 阶跃响应仿真**
16. **figure;**
17. **step(cl\_sys);**
18. **title('单位负反馈闭环二阶系统阶跃响应');**
19. **xlabel('时间 (秒)');**
20. **ylabel('输出');**
21. **grid on;**
22. **% 计算系统阶跃响应特性**
23. **info = stepinfo(cl\_sys);**
24. **Kp = dcgain(cl\_sys);**
25. **% 输出性能指标**
26. **fprintf('--- 二阶闭环系统性能指标 ---\n');**
27. **fprintf('超调量（%%%%）：%.2f\n', info.Overshoot);**
28. **fprintf('上升时间（秒）：%.4f\n', info.RiseTime);**
29. **% 打印 G(s)**
30. **open\_sys**

分析得期望系统的阶跃响应超调量为16.29%，调节时间为0.0468s，满足要求，验证了期望频率特性法的有效性。图如下：



用期望开环传函除以原系统开环传函，即可得到校正环节的传函（为了物理可实现性，增加了一个很远的极点：，可得：



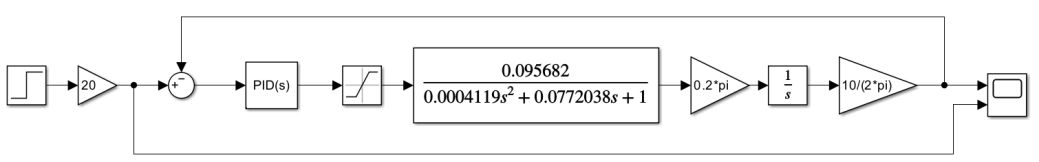
下一节将通过Simulink仿真来分析系统的阶跃响应和正弦响应。

## 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。记录仿真验证过程。

**设计方法一：**

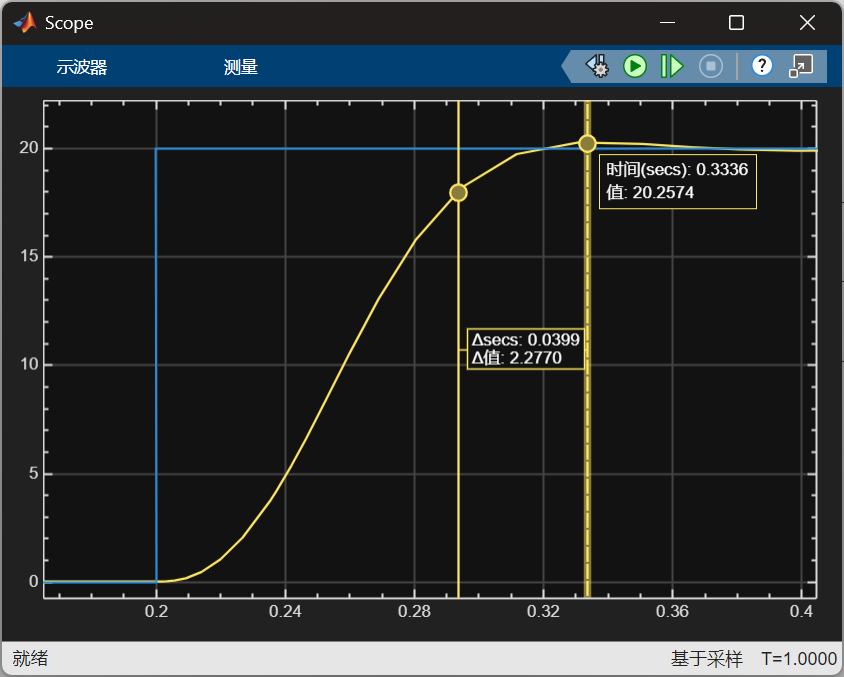
Simulink**仿真框图**：



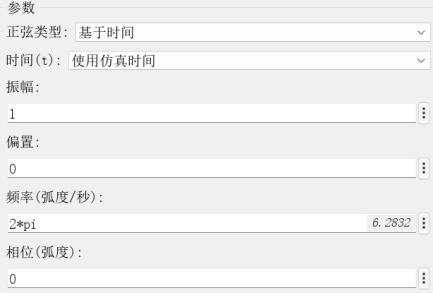


**阶跃输入**结果：

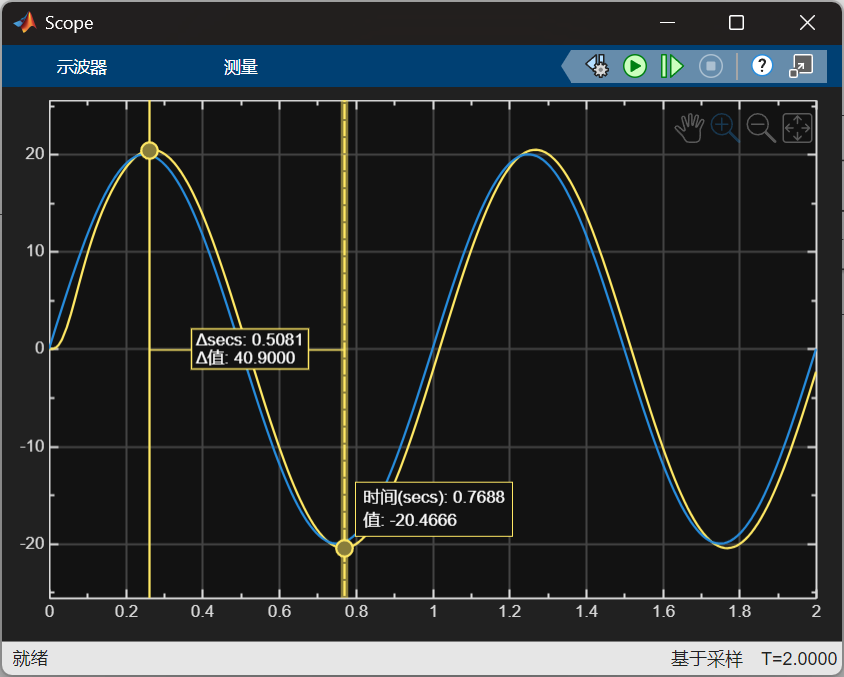
在0.2s时有一个幅值为20的阶跃输入，如下图，可见校正后系统的超调量为，调节时间约为，符合对系统的动态特性要求。



**正弦信号输入**结果：

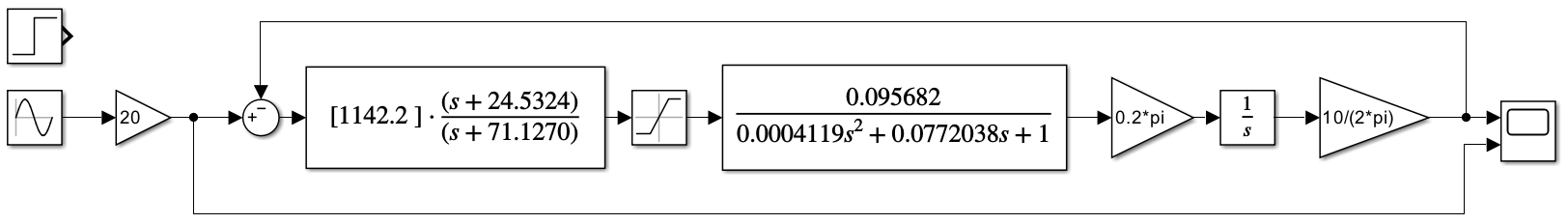


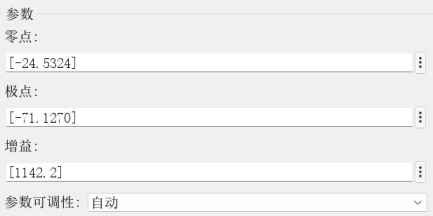
输入1Hz的正弦信号，输入输出曲线如下，可见输出没有衰减，说明校正后系统带宽满足要求；延迟也较低，矫正环节起到了应有的作用。



**设计方法二：**

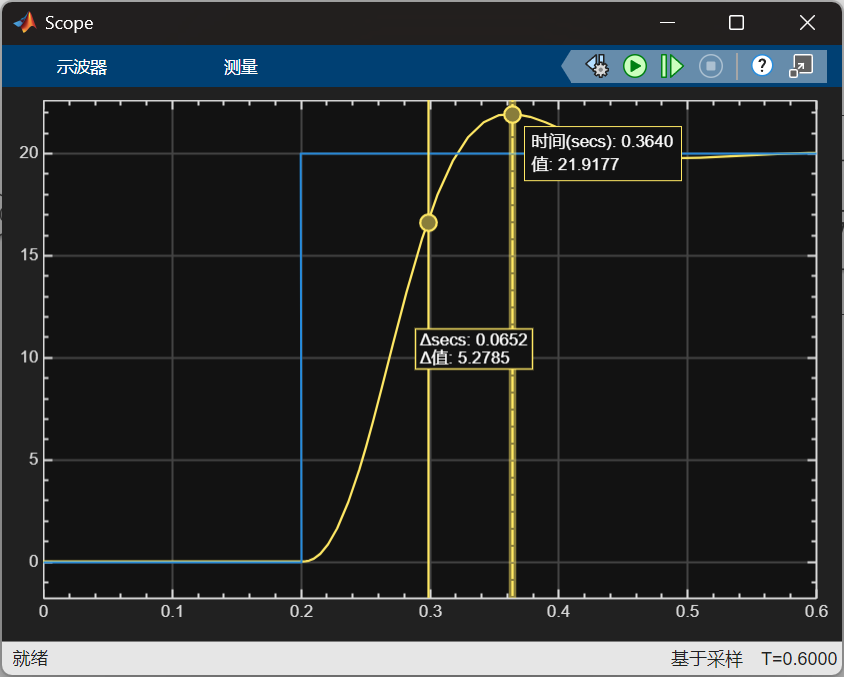
Simulink**仿真框图**：



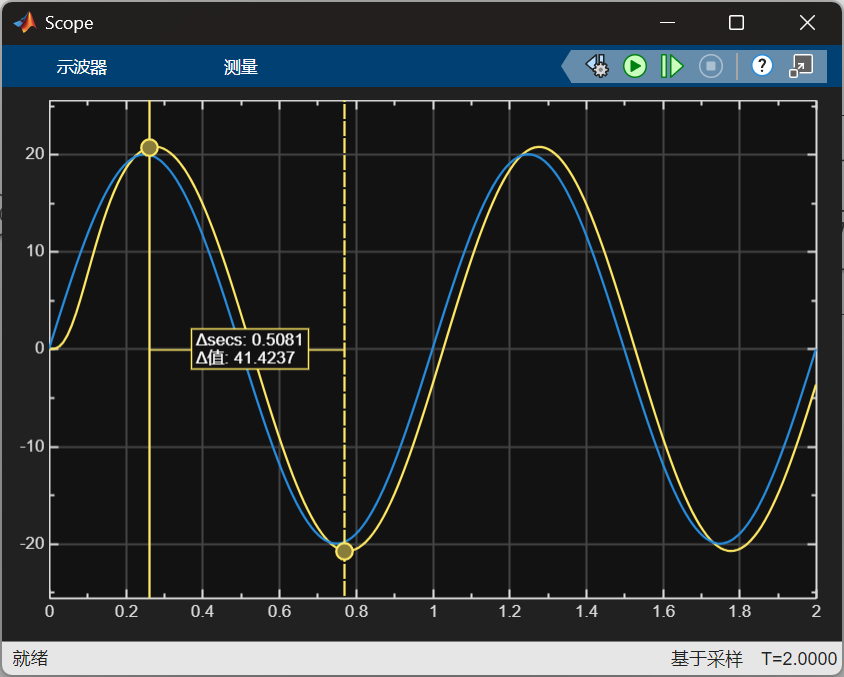


**阶跃输入**结果：

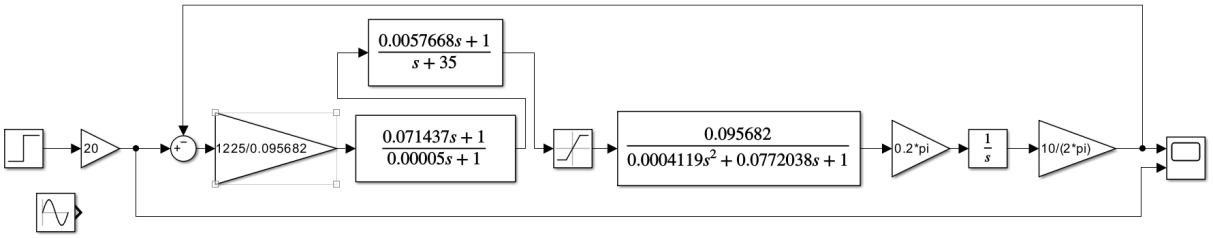
在1s时有一个幅值为20的阶跃输入，如下图，可见校正后系统的超调量为，符合要求



输入1Hz的正弦信号，输入输出曲线如下，可见输出没有衰减，说明校正后系统带宽满足要求；延迟也较低，矫正环节起到了应有的作用。

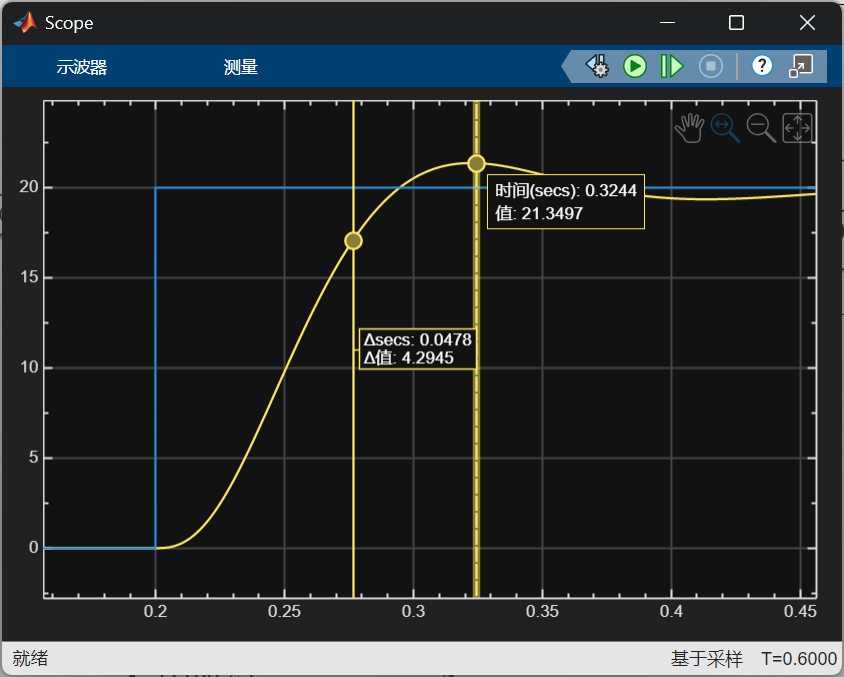


**设计方法三：**

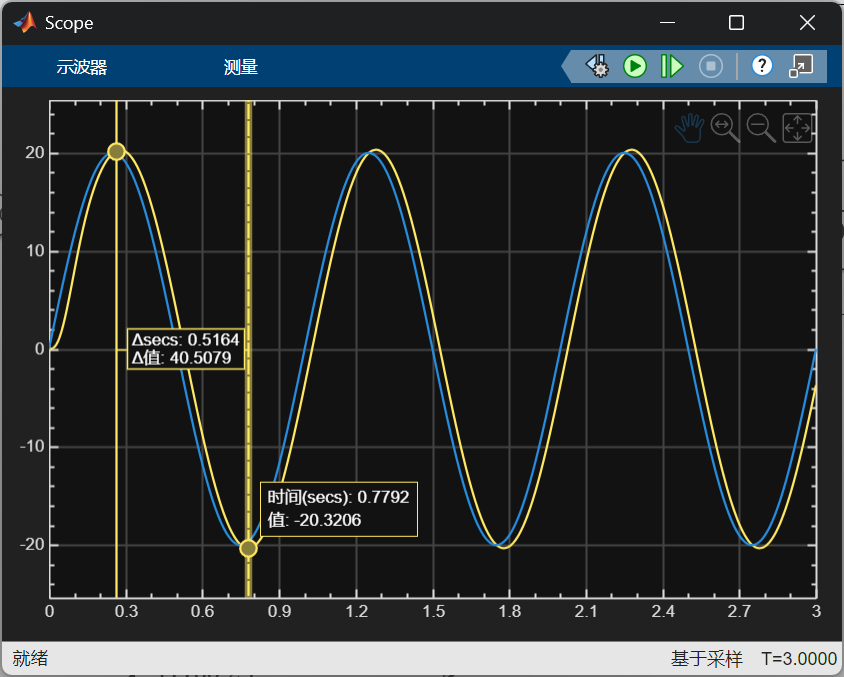


**阶跃输入**结果：

在1s时有一个幅值为20的阶跃输入，如下图，可见校正后系统的超调量为，调节时间约为，符合要求



输入1Hz的正弦信号，输入输出曲线如下，可见输出没有衰减，说明校正后系统带宽满足要求；延迟也较低，矫正环节起到了应有的作用。



**总结：**

设计方法一二三得出的控制器校正后，系统的动态性能都有显著提升，但是方法二不如设计方法一得出的控制器。

## 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。

**① 设计方法一：**

转换为离散化控制器的代码为：

1. **s = zpk('s');**
2. **% 设置控制器**
3. **ks = 500 + 18 \* 100 / (1 + 100 / s);**
4. **% 设置控制器周期 Ts**
5. **Ts = 0.01;**
6. **% 使用 c2d 进行双线性变换**
7. **kz = c2d(ks, Ts, 'tustin');**
8. **% 使用 tf 转变为离散的传递函数**
9. **kz\_tf = tf(kz);**
10. **kz\_tf**

结果为：

1. **kz\_tf =**
3. **1700 z - 1367**
4. **-------------**
5. **z - 0.3333**
7. **采样时间: 0.01 seconds**
8. **离散时间传递函数**

从离散控制器推导控制规律：

给定离散控制器的传递函数为：

将分子和分母分别表示为的多项式：

将分母乘到等号左边：

展开并取Z逆变换：





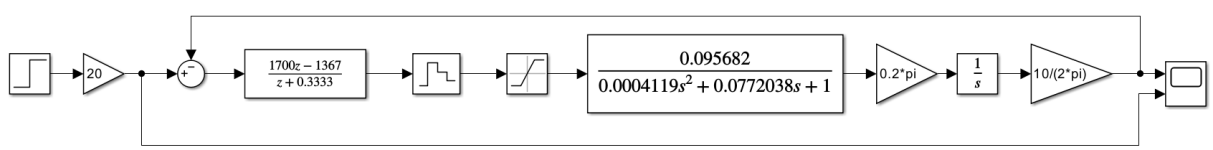
即：

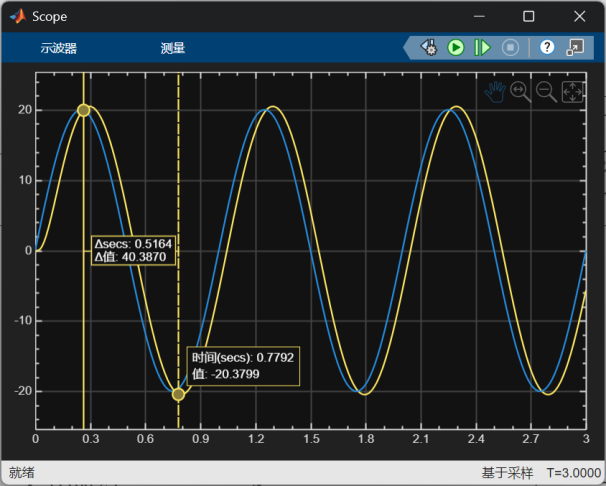
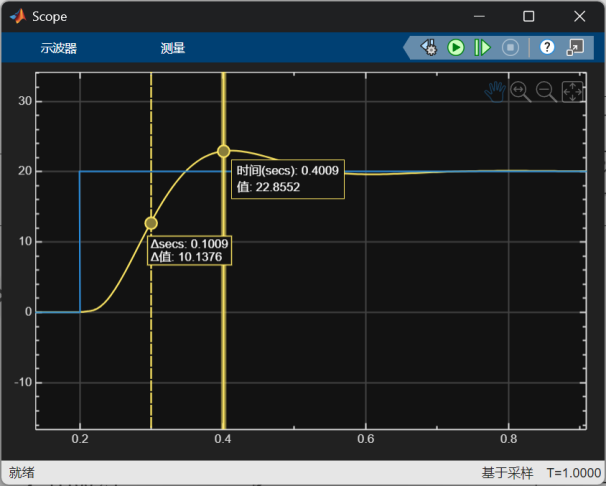


最终控制律表达式：



对于离散控制器的效果进行仿真：





效果不错，说明离散化后的控制器仍能有效控制系统。

1. 将控制器源代码粘贴在下方，并写好注释。
2. **pos = RegulatedEncoderVal \* 0.005f; *// 获取当前的位置信息***
3. **error = ref - pos; *// 计算误差***
4. ***// 计算控制量，PD 控制器***
5. **control = (float)(1700 \* error - 1367 \* last\_error + 0.3333 \* last\_control);**
6. **send\_data\_2\_matlab(time2, ref, pos); *// 发送数据到 matlab***
7. ***// 更新上次的控制量和误差***
8. **last\_control = control;**
9. **last\_error = error;**
10. **control\_int = (int16\_t)control; *// 转换为整数类型***
11. ***// 发送控制量到电机***
12. **if (control\_int > 4997) control\_int = 4997;**
13. **if (control\_int < -4997) control\_int = -4997;**
14. **MC\_ProgramTorqueRampMotor1(control\_int, 0);**

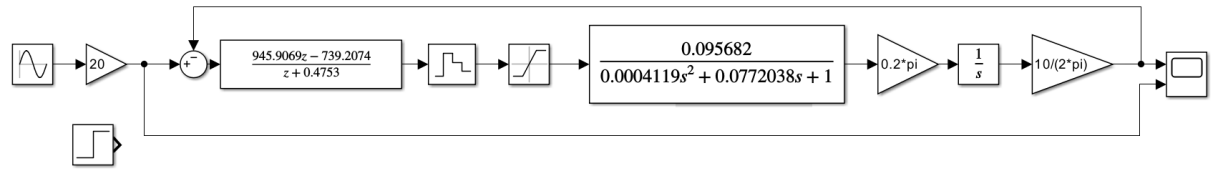
**② 设计方法二：**

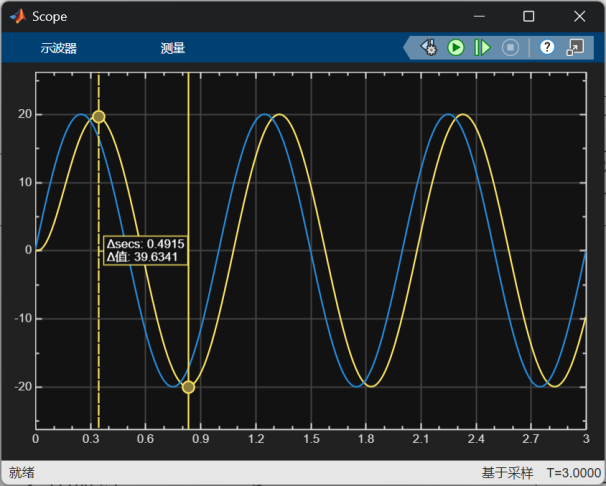
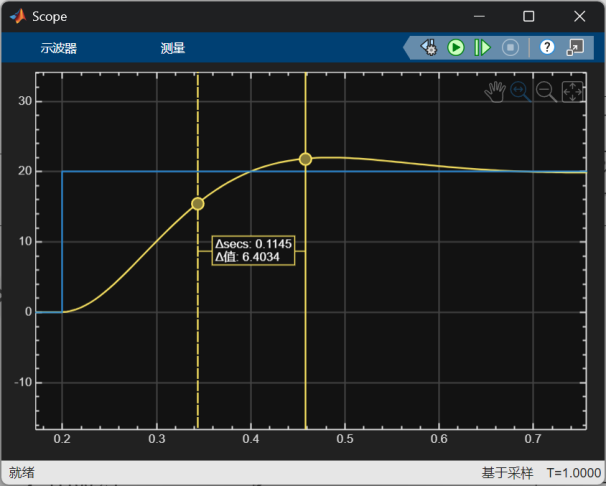
结果为：

1. **kz\_tf =**
3. **945.9069 z - 739.2074**
4. **----------------------**
5. **z - 0.4753**
7. **采样时间: 0.01 seconds**
8. **离散时间传递函数**

最终控制律表达式（同法一，这里只给出结果）：







**② 设计方法三：**

结果为：

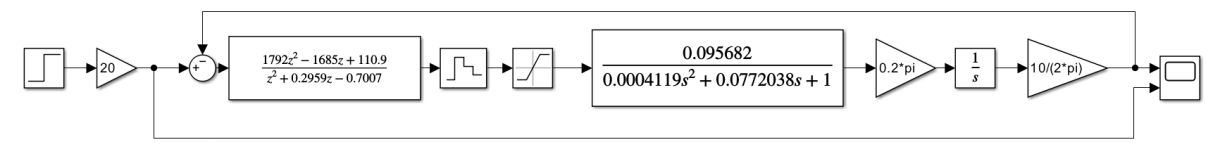
1. **G\_disc\_tf =**
3. **1792 z^2 - 1685 z + 110.9**
4. **-------------------------**
5. **z^2 + 0.2959 z - 0.7007**
7. **采样时间: 0.01 seconds**
8. **离散时间传递函数**

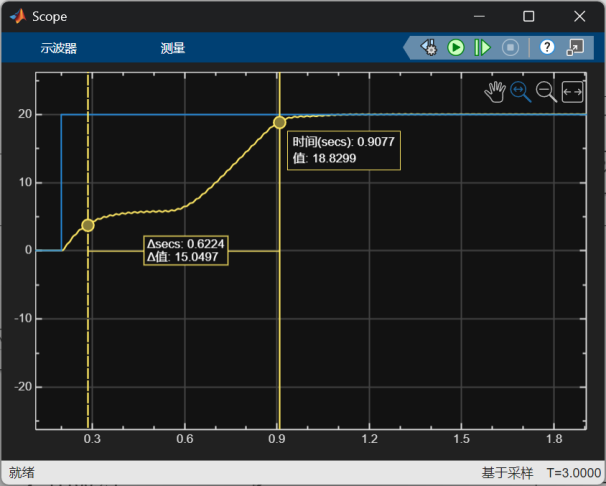
最终控制律表达式（同法一，这里只给出结果）：





对于离散控制器的效果进行仿真：





可以看出控制快速性很差，说明将控制器离散化会对控制器的性能有很大的影响。因此我们不适用方法三获得的控制器进行实机测试，只使用前两种方法的结果。

## 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。

下面是方法一：PD控制器的阶跃响应测试源代码

1. **void step\_input\_test(void) {**
2. **static uint32\_t sys\_tick\_step\_test = 0;**
3. **static float time\_step\_test = 0;**
4. **float ini\_pos = 0;**
5. **sys\_tick\_step\_test = HAL\_GetTick();**
6. **time\_step\_test = 0.001 \* sys\_tick\_step\_test;**
7. **if (step\_init\_flag == 0) {**
8. **ini\_pos = RegulatedEncoderVal \* 0.005f;**
9. **ref = ini\_pos + 20;**
10. **step\_init\_flag = 1;**
11. **}**
12. **if (START\_STATUS == 4) {**
13. **if (sys\_tick\_step\_test % 10 == 0) {**
14. **pos = RegulatedEncoderVal \* 0.005f;  *// 获取当前的位置信息***
15. **error = ref - pos;                   *// 计算误差***
16. ***// 计算控制量，PD 控制器***
17. **control = (float)(1700 \* error - 1367 \* last\_error + 0.3333 \* last\_control);**
18. **send\_data\_2\_matlab(time\_step\_test, ref, pos);  *// 发送数据到 matlab***
19. ***// 更新上次的控制量和误差***
20. **last\_control = control;**
21. **last\_error = error;**
22. **control\_int = (int16\_t)control;  *// 转换为整数类型***
23. ***// 发送控制量到电机***
24. **if (control\_int > 4997) control\_int = 4997;**
25. **if (control\_int < -4997) control\_int = -4997;**
26. **MC\_ProgramTorqueRampMotor1(control\_int, 0);**
27. **}**
28. **}**
29. **}**
30. 将扫频跟随测试源代码粘贴在下方，并写好注释。

下面是方法一：PD控制器的扫频跟随测试源代码

1. **void sine\_wave\_input\_test(void) {**
2. **static my\_sweep\_t sweep = {0};**
3. **static uint32\_t last\_sys\_tick\_sine\_wave\_test = 0;**
4. **static uint32\_t sys\_tick\_sine\_wave\_test = 0;**
5. **static float time\_sine\_wave\_test = 0;**
6. **static float ref\_signal = 0;       *// 参考信号***
7. **static float ref\_signal\_init = 0;  *// 参考信号的初始值***
8. **sys\_tick\_sine\_wave\_test = HAL\_GetTick();**
9. **time\_sine\_wave\_test = 0.001 \* sys\_tick\_sine\_wave\_test;**
10. **if (last\_sys\_tick\_sine\_wave\_test != sys\_tick\_sine\_wave\_test) {**
11. **last\_sys\_tick\_sine\_wave\_test = sys\_tick\_sine\_wave\_test;**
12. **if (sys\_tick\_sine\_wave\_test % 10 == 0) {**
13. **if (sine\_wave\_init\_flag == 0) {**
14. ***// 初始化扫频配置***
15. **init\_my\_sweep(&sweep, sys\_tick\_sine\_wave\_test, 10000, 1, 1.001, 20);**
16. **ref\_signal\_init = RegulatedEncoderVal \* 0.005f;**
17. **sine\_wave\_init\_flag = 1;**
18. **}**
19. ***// 获取正弦扫频信号***
20. **pos = RegulatedEncoderVal \* 0.005f;**
21. **ref\_signal = ref\_signal\_init + run\_my\_sweep(&sweep, sys\_tick\_sine\_wave\_test);**
22. **error = ref\_signal - pos;**
23. ***// 计算控制量，PD 控制器***
24. **control = (1700 \* error - 1367 \* last\_error + 0.3333 \* last\_control);**
25. ***// 发送数据到 matlab***
26. **send\_data\_2\_matlab(time\_sine\_wave\_test, ref\_signal, pos);**
27. **last\_control = control;**
28. **last\_error = error;**
29. **control\_int = (int16\_t)control;**
30. ***// 信号限幅***
31. **if (control\_int > 4997) control\_int = 4997;**
32. **if (control\_int < -4997) control\_int = -4997;**
33. ***// 发送控制量到电机***
34. **MC\_ProgramTorqueRampMotor1(control\_int, 0);**
35. **}**
36. **}**
37. **}**
38. 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：

**方法一：**

校正后系统的**阶跃**响应曲线如下所示，阶跃输入为20，即2.0cm，上升时间约为0.07s，没有超调，稳态误差为104.5-104.05=0.45mm，小于给定要求1mm。系统的动态特性均满足要求，说明矫正环节达到了应有的效果。



**方法二：**

校正后系统的**阶跃**响应曲线如下所示，阶跃输入为20，即2.0cm，上升时间约为0.07s，超调量小于10%，稳态误差为0.30mm，均满足要求，说明矫正环节达到了应有的效果。



1. 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;

**方法一：**

校正后系统的**正弦**响应曲线如下所示，阶跃输入为20，即2.0cm，输入为1Hz，输入最大值为，输出最大值为，计算衰减：，满足对系统扫频跟随要求，说明矫正环节达到了应有的效果。



**方法二：**



分析同上，满足对系统扫频跟随要求，说明矫正环节达到了应有的效果。

1. 写出经过调试后，最终的控制器传递函数。

经过仿真和实机上现象的对比，最终选择方法二的结果作为系统的控制器，传函：



其中采样时间为0.01s。

这个控制器控制下的系统，满足了在阶跃响应曲线和扫频跟随曲线上的性能指标要求。

## 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？

答：本次控制器的设计基于参数辨识所得出的二阶模型进行。尽管该模型在一定程度上能够反映系统的动态特性，但在实际应用中仍存在若干局限性。首先，在建模过程中对系统进行了简化处理，忽略了某些高阶动态和非线性因素，这可能导致理论模型与真实系统之间存在一定偏差。其次，参数辨识本身存在误差，尤其是在实验条件受限或噪声干扰较大的情况下，这种误差可能被放大并影响控制器的性能。此外，控制器设计时未充分考虑执行器的物理限制，如输出幅值和响应速度的约束，这在大信号输入或快速响应需求下可能会导致控制效果下降，甚至引发积分饱和（Windup）等现象。

虽然实机测试结果表明当前控制器基本满足了系统性能要求，但这也从侧面反映出我们在鲁棒性和适应性方面的设计仍有提升空间。为了增强控制器在不确定环境下的稳定性与可靠性，未来可以从以下几个方面进行改进：一是引入鲁棒控制策略，以提高系统对外部扰动和模型不确定性的抵抗能力；二是采用自适应控制机制，使控制器能够根据实时工况自动调整参数；三是针对执行器的幅值限制问题，引入 Anti-Windup 补偿机制，防止积分作用在饱和状态下持续累积，从而避免系统响应失真和恢复延迟；四是结合在线辨识技术，实现控制器参数的在线更新，以应对系统随时间变化而产生的不确定性。

## 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。

答：在设计过程中，我们尝试了多种控制器设计方案。虽然这些控制器在连续域仿真中表现良好，但在实机测试中效果存在差异。造成这一现象的原因可能包括模型参数辨识误差、模型简化带来的偏差、控制器离散化过程中的精度损失，以及执行器的物理限制等。这也说明在实际测试前应对系统模型进行更为全面和细致的评估。为改善控制效果，我们可以通过提高控制器的控制频率，或采用更精确的离散化方法来优化控制器的设计。整个过程也加深了我们对控制系统设计流程的理解，积累了宝贵的经验，为今后类似项目的开展提供了重要参考。