中山大學 SUN YAT-SEN UNIVERSITY

电机拖动技术实验报告

院(系):智能工程学院 学号：22354189 姓名：张瑞程

日期：2024 年 12 月 实验名称：第 3 次实验-BLDC 控制实验

实验三：BLDC 控制实验

目录

实验三：BLDC 控制实验

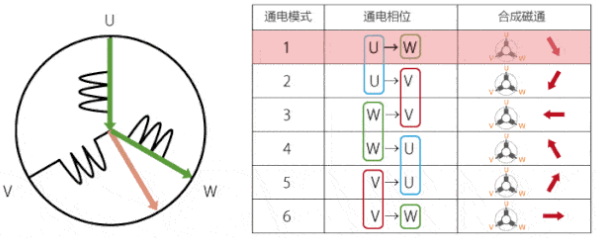
1.实验目的

1. 确定电机运行的参数，包括转速、三相电流和直流电压，并通过调整 CCS 设置的参数值来观察这些参数的变化。

2. 通过示波器观察 DAC 输出的电机转速和电流波形，并与 PMSM 电流波形进行对比。

3. 通过修改转速 PI 控制的比例和积分参数，观察其对控制性能的影响，包括响应时间、超调和稳态误差。通过 CCS 软件和示波器进行观察，并简要对比 PMSM 控制性能。

2.实验原理



BLDC 的工作状态如图所示，本次实验就是要编程实现对这一电机的控制，然后对系统和 BLDC 本体进行性能分析。在实验进行之前，我们有必要对代码部分进行深入的理解和分析：

在 main 函数里面首先调用*InitPeripherals*()和*InitVeriable*()两个函数进行初始化。

```
38      InitPeripherals(); //Initialize all the Device Peripherals
39      InitVariables(); //Initialize Variables
```

首先是*InitPeripherals*()，该函数的目的是初始化设备的所有外设，以及 AD 和 DA 转换：

```
128 void InitPeripherals(void) // Initialize all the Device Peripherals
129 {
130     ADC_Init(); //初始化ADC
131     InitialDAC();
132     scic_init();
133
134     Init_Cap1(0, 1, 0, 1); //初始化CAP1
135     Init_Cap2(0, 1, 0, 1); //初始化CAP2
136     Init_Cap3(0, 1, 0, 1); //初始化CAP3
137
138     EPWM_Enable(1); //使能EPWM引脚
139     EPWM_Single_Init(); //初始化EPWM
140     EPWM_OUTPUT.PRD = EPWM_Configure(100); //配置EPWM
141     EPWM_OUTPUT.CMP = EPWM_OUTPUT.PRD * 0.1;
142 }
```

其中，ADC_Init();：调用 ADC_Init 函数来初始化模数-数转换器（ADC）。

InitialDAC();：调用 InitialDAC 函数来初始化数模转换器（DAC）。

scic_init();：调用 scic_init 函数来初始化串行通信接口（SCIC）。

Init_Cap1(0, 1, 0, 1);、Init_Cap2(0, 1, 0, 1);、Init_Cap3(0, 1, 0, 1);：分别初始化捕获器 1（CAP1）、捕获器 2（CAP2）和捕获器 3（CAP3）。

然后调用*InitVeriable*()，对一些需要用到的参数进行初始化：

```
144  void InitVariables(void)
145  {
146  //Initialize system variables
147      MAIN_LOOP.cnt1 = 0;
148      MAIN_LOOP.cnt2 = 0;
149
150  //Insert user code
151
152      bldcm.idc = 0;
153      bldcm.pole = 4; //电机极对数
154      bldcm.speed = 0;
155      bldcm.udc = 0;
156
157      bldcm_set.control_logic = 0;
158      bldcm_set.cw_ccw = 1;
159      bldcm_set.start = 0;
160      bldcm_set.run = 0;
161
162      hall.cap_t1 = 0;
163      hall.cap_t2 = 0;
164      hall.sysclk = 150; //时钟频率150M
165      hall.pos = 0;
166
167      pid_speed.Kp = 0.1;
168      pid_speed.Ki = 0.01;
169      pid_speed.Kd = 0;
170      pid_speed.Set = 1000; //初始转速1000
171      pid_speed.Outmax = 10;
172      pid_speed.Outmin = -10;
173      pid_speed.Outpre = 0;
```

需要特别关注其中的：pid_speed，Kp，Ki,后面需要修改这些参数，观看实验现象。在进行各个参数的初始化以后，就是要执行主循环了。程序一直运行 for(;;)里面的代码执行条件判断：

```
86          if ((bldcm_set.start == 1) && (bldcm_set.run == 0)) //起动电机
87          {
88              EPWM_Enable(1);
89
90              Motor_Ctrl(&bldcm_set, &hall);
91              bldcm_set.run = 1;
92
93          }
```

```
94          if (bldcm_set.start == 0) //停止电机
95          {
96              EPWM_Enable(0);
97
98              bldcm_set.run = 0;
99              pid_speed.Outpre = 0;
100         }
```

电机控制运行在中断中，中断是每个固定时间执行一次，执行过程中主程序暂停执行。

```
6   interrupt void Cap_Isr(void);
```

每次中断执行的具体动作如下面的代码所示：

```
116  interrupt void Cap_Isr(void)
117  {
118      Motor_Ctrl(&bldcm_set, &hall); //电机运行函数
119
120      ECap1Regs.ECCLR.all = 0xFFFF; //clear interrupt flag
121      ECap2Regs.ECCLR.all = 0xFFFF;
122      ECap3Regs.ECCLR.all = 0xFFFF;
123
124      PieCtrlRegs.PIEACK.all = PIEACK_GROUP4;
125
126  }
```

其中包含了 118 行所写的电机的位置切换（每隔固定时间运行）

```
177  void BLDCAPP(void *argv, uint16_t argc)
```

BLDCAPP()为电机控制应用函数。根据霍尔传感器的读数计算电机速度，使用 PID 控制调节脉宽调制（PWM）来控制电机速度。

```
177  void BLDCAPP(void *argv, uint16_t argc)
178  {
179  //insert user code
180      hall.cap_t1 = Get_Cap1_Ts1();
181      hall.cap_t2 = Get_Cap1_Ts2();
182
183      if (bldcm_set.run == 1)
184      {
185          bldcm.speed = Motor_Speed(&hall, &bldcm);  //计算电机转速
186          pid_speed.Actual = bldcm.speed;
187          EPWM_OUTPUT.CMP = 3000 + Pid_Ctrl(&pid_speed); //PID调节
188          if (EPWM_OUTPUT.CMP > EPWM_OUTPUT.PRD)
189          {
190              EPWM_OUTPUT.CMP = EPWM_OUTPUT.PRD;
191          }
192      }
193      else
194      {
195          bldcm.speed = 0;  //计算电机转速
196      }
197
198      ADC_Ctrl(); //ADC采样
199      bldcm.udc = ((float) Get_A_Adc(0)) * 10.0 * 16 / 32768.0;
200      bldcm.idc = ((float) Get_A_Adc(1)) * 10.0 * 2 / 32768.0;
201      bldcm.iu= -((float) Get_A_Adc(2)) * 0.00152587890625 + 12.5; //外部电路正负极错误
202      bldcm.iv = -((float) Get_A_Adc(3)) * 0.00152587890625 + 12.5;
203      bldcm.iw = -((float) Get_A_Adc(4)) * 0.00152587890625 + 12.5;
```

```
204
205      WriteDAC(DA_ADD0,bldcm.udc*0.1);
206      WriteDAC(DA_ADD1,bldcm.speed*0.001);
207      WriteDAC(DA_ADD2,bldcm.idc*100);
208      WriteDAC(DA_ADD3,bldcm.iv*10);
209  }
```

由于 DAC 的输出范围是-10V 到 10V，因此需要对 BLDC 电机的电压、电流和转速等信号进行相应的缩放，以确保能够正确地输出到 DAC 上。在使用 CCS（Code Composer Studio）观察值或波形时，例如三相电流，由于采样频率的限制，我们可能无法直接获取到完整的方波形。为了在示波器上观察到清晰的 BLDC 电机的方波驱动波形，需要将这些信号输出到相应的 DAC 通道上：DA_ADD0—通道 1；DA_ADD1—通道 2；DA_ADD2—通道 3；DA_ADD3—通道 4。

需要特别注意的是：

a. 电机运行的数据存在 bldcm 结构里面，包括电机的电流、电压和转速等信息。这些数据反映了电机在实时运行中的状态。

```
152     bldcm.idc = 0;
153     bldcm.pole = 4; //电机极对数
154     bldcm.speed = 0;
155     bldcm.udc = 0;
```

b. 控制器参数都存 pid_Speed 结构里面（Set 是参考值，Kp,Ki 是 PI 控制器参数）

```
167     pid_speed.Kp = 0.1;
168     pid_speed.Ki = 0.01;
169     pid_speed.Kd = 0;
170     pid_speed.Set = 1000; //初始转速1000
171     pid_speed.Outmax = 10;
172     pid_speed.Outmin = -10;
173     pid_speed.Outpre = 0;
```

3.实验设备

（1）硬件：BOX28335 实验平台，仿真器 HDSP-XDS200ISO，相应的配套电源。

（2）软件：安装了 Windows 10 和 CCS 软件的 PC。

4.实验步骤及结果展示

首先先进行基本的运行，测试设备连接情况，将永磁同步电机的三相电线与位置传感器线连接到试验箱，参考上图连接。

然后运行红框中的程序测试连接，当全部 success 以后就可以开始进行后续的实验了。

```
Scan tests: 5, skipped: 0, failed: 0
Do a test using 0xAACC3355.
Scan tests: 6, skipped: 0, failed: 0
All of the values were scanned correctly.

The JTAG DR Integrity scan-test has succeeded.

[End: Texas Instruments XDS100v2 USB Emulator]
```

点击“小锤子”图标编译程序

点击“小瓢虫”图标下载永磁同步电机程序 TestBoxBLDC 并点击运行按钮

按照图示，按顺序选择 BLDC,点击启动按钮，设置转速，实现电机转动。然后将各个变量添加进 Expression 中进行观察如下图,注意要电机箭头所示的地方,开启“连续刷新”功能。

Expression	Type	Value	Address
bldcm.speed	unsigned int	0	0x00009751@Data
pid_speed	struct pid_para	{Kp=0.100000001,Ki=0.00999999978,Kd=...	0x000093D6@Data
bldcm.iu	float	-0.0381469727	0x00009754@Data
bldcm.iv	float	-0.0350952148	0x00009756@Data
bldcm.iw	float	0.0411987305	0x00009758@Data
bldcm.udc	float	23.5205078	0x00009752@Data

在 Expression 中选择变量，右击选择 Graph，就会出现上图，就可以实时观察数据。

Task1

任务要求：“启动电机，通过 CCS 的 Expression 和 Graph 观察电机运行，包括电机转速，三相电流，直流电压等参数，（通过 CCS Expression 或屏幕 设置转速 PI_Speed 里面 SetPoint 参数的值，观察上述参数的值与波形）。记录并分析实验结果。”

完成步骤：启动电机以后，就直接可以在电机的屏幕上面观察到基本的转速等数据：

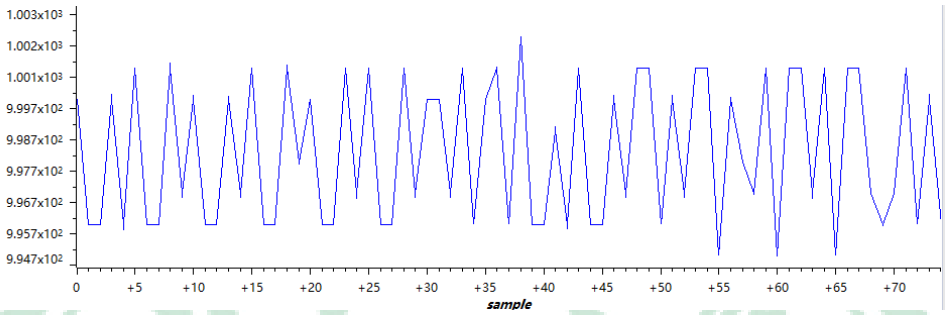
但是试验箱显示器只能观察，无法记录，还是要在 Expression 中的 graph 中进行分析。我们在 CCS 的 Expression 界面加入 Speed、udc、idc 等变量，观察它们的数值变化：在 Bldc 未启动时候的状态:

Expression	Type	Value	Address
bldcm.speed	unsigned int	0	0x00009751@Data
pid_speed	struct pid_para	{Kp=0.100000001,Ki=0.00999999978,Kd=...	0x000093D6@Data
bldcm.iu	float	-0.0381469727	0x00009754@Data
bldcm.iv	float	-0.0350952148	0x00009756@Data
bldcm.iw	float	0.0411987305	0x00009758@Data
bldcm.udc	float	23.5205078	0x00009752@Data

启动以后的状态：

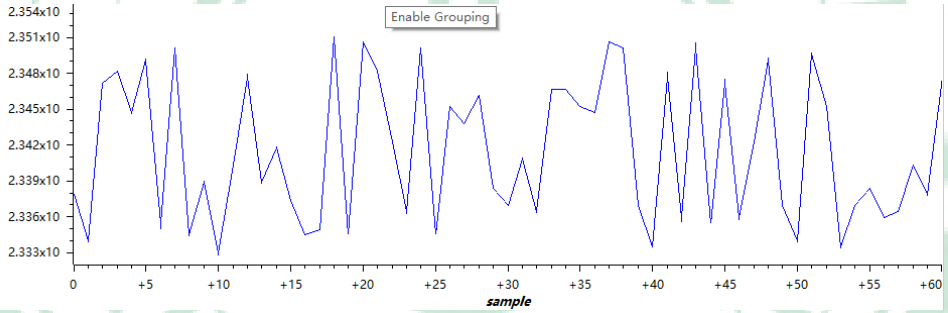
Expression	Type	Value	Address
bldcm.speed	unsigned int	1001	0x00009751@Data
pid_speed	struct pid_para	{Kp=0.100000001,Ki=0.00999999978,Kd=...	0x000093D6@Data
bldcm.iu	float	-0.0579833984	0x00009754@Data
bldcm.iv	float	-0.0244140625	0x00009756@Data
bldcm.iw	float	0.0350952148	0x00009758@Data
bldcm.udc	float	23.515625	0x00009752@Data
pid_speed.Set	float	1000.0	0x000093DC@Data
Add new expression			

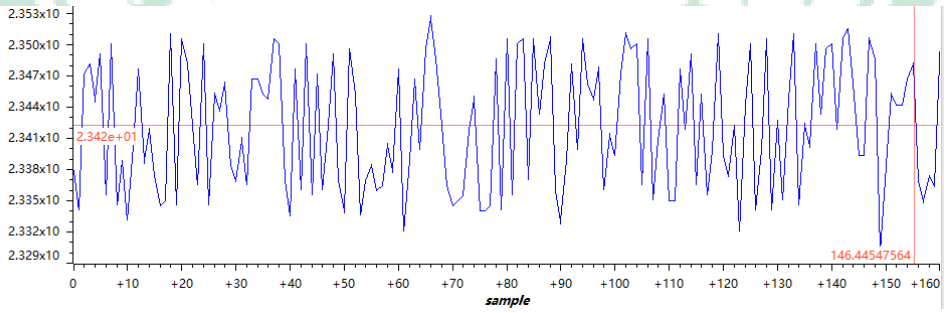
转速变化曲线



在实验过程中，我们将电机的设定转速调整为 1000 转。通过观察记录的数据图表，可以明显看到实际运行时的转速在 1000 转附近有轻微的波动（波动幅度大概 0.5%），偏差处于一个可接受的误差范围内，从而证实了电机的运行状态较为稳定，所采用的控制系统能够有效地维持预定的转速。

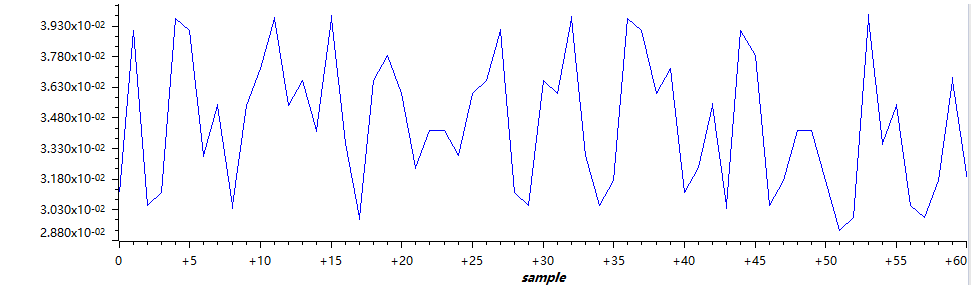
直流电压 udc

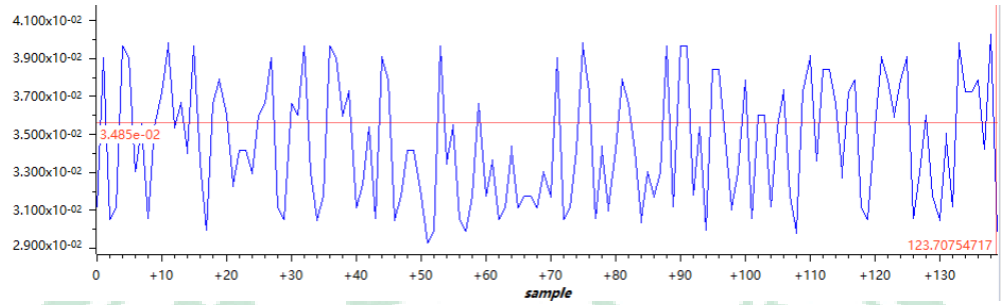




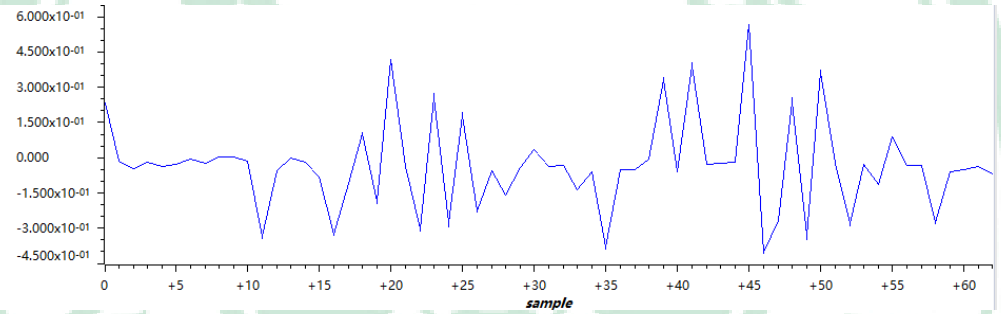
在实验过程中，我们注意到电压值围绕 23.42 伏特这个中心点，在合理的范围内呈现周期性波动，波动幅度大约为 0.5%。这种微小的波动表明系统运行稳定，电压控制精确，且在可接受的误差范围内。

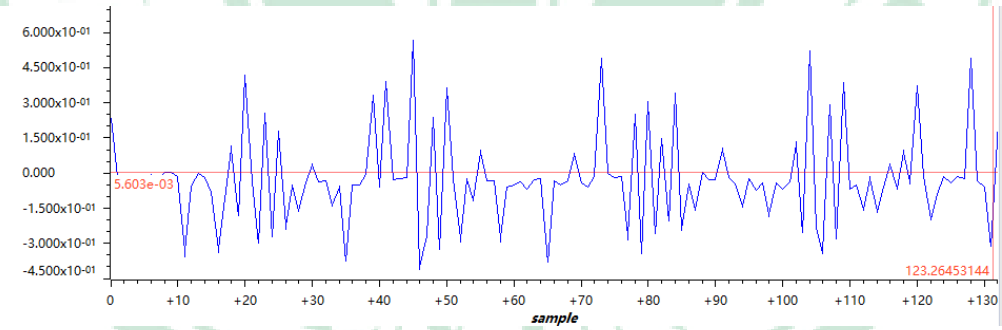
直流电流 idc



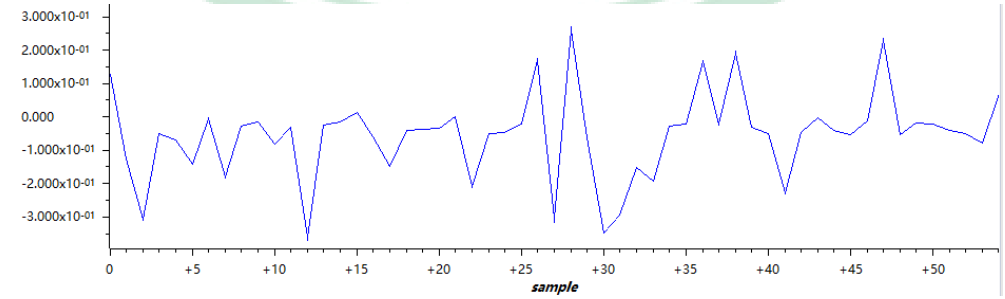


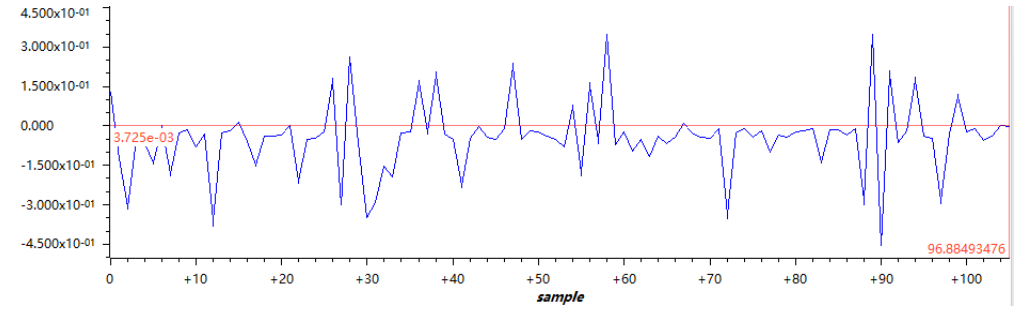
三相电流 iu



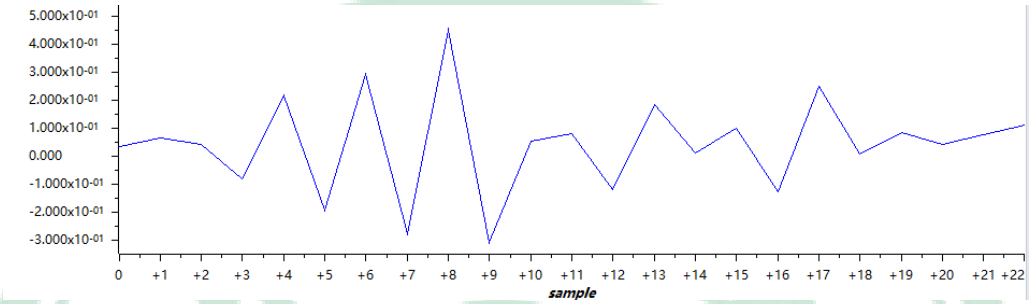


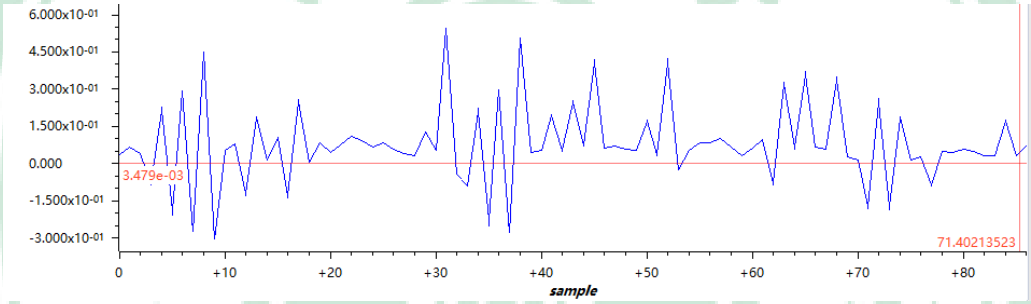
三相电流 iv





三相电流 iw





三相电流在实验中展现出典型的周期性波动特征，每一相电流均呈现出规律性的变化。值得注意的是，这三相电流之间保持着精确的相位差，具体为每相之间相差 120 度（即三分之一个圆周），这是三相系统正常运行的典型标志。然而，在观察中发现，Iv 和 Iw 两相电流的波动中心并非完美地对齐在零值基准线上，而是存在一定的偏差。这种偏差可能源于测量设备的精度限制或系统内部的微小不平衡，尽管如此，整体系统的运行仍保持稳定，且这种偏差对系统的整体性能影响有限。

Task2

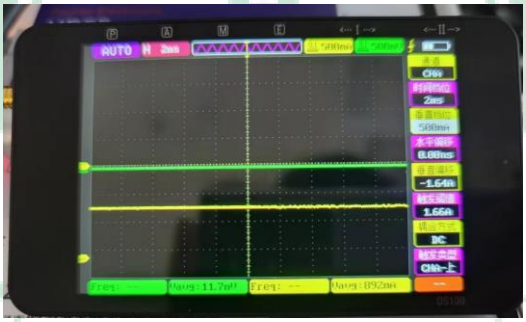
任务要求：通过示波器观察电机(bldcm 里面)的电机的转速，电流等波形，这里需要将电机的转速,三相电流,电压等波形输出到 DAC 通道(当前代码已经包含部分输出到 DAC 通道，电流等，其他没有加的需要自己手动添加/替换，添加相应的代码 WriteDAC(DA_ADDxx, xxxx)；xx 表示通道与 xxxx 表示要输出的变量 e.g 包括转速，三相电流，电压等。记录不同转速下的实验结果，包括转速，三相电流，电压等；对比上次实验 PMSM 三相电流波形，记录并分析结果

完成步骤：逐一进行分析，首先是**转速 speed**

```
206     WriteDAC(DA_ADD1,bldcm.speed*0.001);
```

Expression	Type	Value	Address
bldcm.speed	unsigned int	718	0x00009751@Data
pid_speed	struct pid_para	{Kp=0.100000001,Ki=0.00999999978,Kd=...	0x000093D6@Data
bldcm.iu	float	-0.0106811523	0x00009754@Data
bldcm.iv	float	-0.242614746	0x00009756@Data
bldcm.iw	float	0.0747680664	0x00009758@Data
bldcm.udc	float	23.4179688	0x00009752@Data
pid_speed.Set	float	700.0	0x000093DC@Data
bldcm.idc	float	0.0378417969	0x0000975A@Data

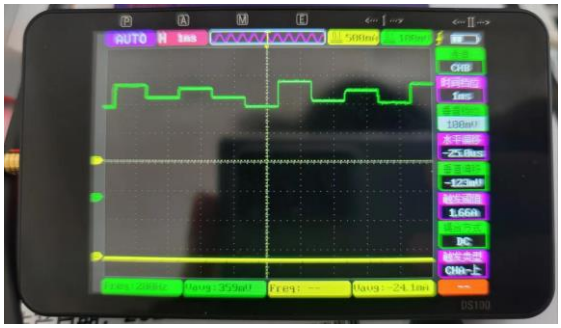
Expression	Type	Value	Address
bldcm.speed	unsigned int	884	0x00009751@Data
pid_speed	struct pid_para	{Kp=0.100000001,Ki=0.00999999978,Kd=...	0x000093D6@Data
bldcm.iu	float	-0.0427246094	0x00009754@Data
bldcm.iv	float	-0.0350952148	0x00009756@Data
bldcm.iw	float	0.0625610352	0x00009758@Data
bldcm.udc	float	23.4912109	0x00009752@Data
pid_speed.Set	float	900.0	0x000093DC@Data
bldcm.idc	float	0.0335693359	0x0000975A@Data
Add new expression			



idc（直流电流）

```
207     WriteDAC(DA_ADD2,bldcm.idc*100);
```

Expression	Type	Value	Address
bldcm.speed	unsigned int	994	0x00009751@Data
pid_speed	struct pid_para	{Kp=0.100000001,Ki=0.00999999978,Kd=...	0x000093D6@Data
bldcm.iu	float	-0.0701904297	0x00009754@Data
bldcm.iv	float	-0.0305175781	0x00009756@Data
bldcm.iw	float	0.115966797	0x00009758@Data
bldcm.udc	float	23.5009766	0x00009752@Data
pid_speed.Set	float	1000.0	0x000093DC@Data
bldcm.idc	float	0.0305175781	0x0000975A@Data



udc（直流电压）

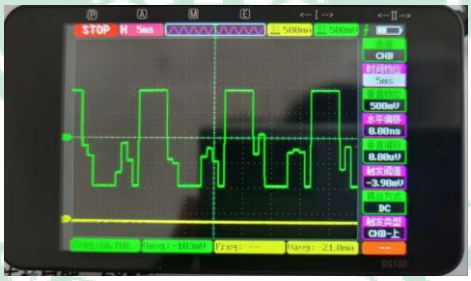
```
205     WriteDAC(DA_ADD0,bldcm.udc*0.1);
```

Expression	Type	Value	Address
bldcm.udc	float	23.5302734	0x00009752@Data
Add new expression			

iv（v 相电流）

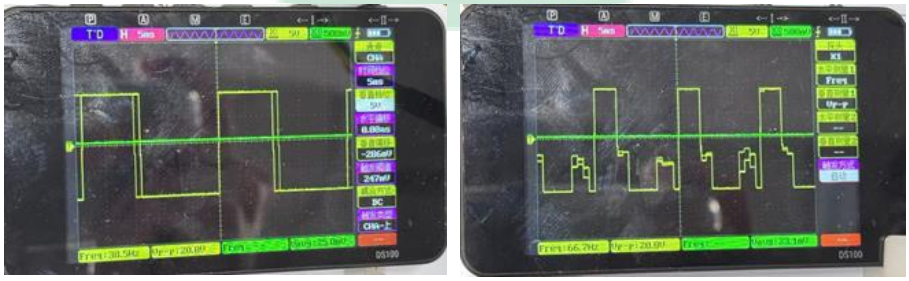
```
161     WriteDAC(DA_ADD2, pmsm.iv * 1);
```

Expression	Type	Value	Address
bldcm.speed	unsigned int	994	0x00009751@Data
pid_speed	struct pid_para	{Kp=0.100000001,Ki=0.00999999978,Kd=...	0x000093D6@Data
bldcm.iu	float	0.184631348	0x00009754@Data
bldcm.iv	float	-0.0457763672	0x00009756@Data
bldcm.iw	float	0.0457763672	0x00009758@Data
bldcm.udc	float	23.4863281	0x00009752@Data
pid_speed.Set	float	1000.0	0x000093DC@Data
bldcm.idc	float	0.0335693359	0x0000975A@Data
Add new expression			



与 pmsm 的对比：

我们同时对比了永磁同步电机的输出波形，并从参数设置和电机原理层面对二者进行了对比分析。



上图左侧为 pmsm 输出电流波形，右侧为 bldc 输出电流波形可以观察到，pmsm 输出结果更为稳定。对于此现象，我们进行了如下分析：

参数方面：PMSM 通常具有更小的比例增益和较大的积分增益（如 Kp=0.04, Ki=0.5），这使得 PMSM 在稳定性方面通常优于 BLDC。PMSM 的这种参数设置有助于减少由于输入变化引起的剧烈系统反应，从而提高系统的整体稳定性。 相比之下，BLDC 具有相对较高的比例增益和较低的积分增益，这可能使 BLDC 在快速响应方面表现更佳，但在稳定性方面略逊于 PMSM。

原理方面：当比较永磁同步电机（PMSM）和无刷直流电机（BLDC）时，重要的是要考虑它们的基本工作原理和特性，这些因素对于选择合适的 PID 参数和预期的电机性能至关重要。

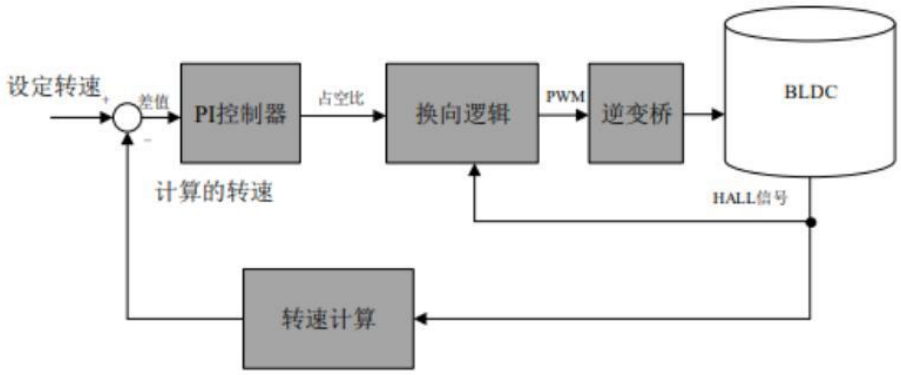
首先，PMSM 和 BLDC 的主要区别在于它们的构造和电磁波形。PMSM 通常采用**正弦波驱动**，这使得其运行更平滑，噪音和振动较小，提供更高的效率和精确度。由于 PMSM 的这些特性，它们通常需要更小的比例增益和更大的积分增益来保持稳定性和准确性。较小的比例增益有助于防止系统对输入变化的剧烈反应，而较大的积分增益则有利于长期稳定和精确的速度或位置控制。另一方面，BLDC 电机使用的是**方波控制**，这使得它们在特定应用中更为简单和成本效益高。BLDC 电机通常有更高的比例增益和较低的积分增益，这使它们在快速动态响应方面表现更佳，但可能在长期稳定性和精确控制方面不如 PMSM。BLDC 电机的这种特性使其非常适合于需要快速启动和停止的应用，例如在电动工具和某些类型的风扇中。

综合来看，选择 PMSM 还是 BLDC 电机取决于应用的具体需求。如果应用需要高效率、高精度和平滑运行，PMSM 可能是更好的选择。相比之下，如果应用需要快速响应和高性价比，BLDC 电机可能更加合适。对于任何类型的电机，正确的 PID 参数调整是实现期望性能的关键，这需要综合考虑电机的特性、应用的需求和系统的整体设计。

Task3

任务要求：在 Expression 里分别修改 PI_speed 中转速 PI 控制的比列与积分参数 kp 和 ki，通过 CCS Graph 观察对转速 speed 控制性能的影响（响应时间，超调，稳态误差等），即设置转速到不同的参考值（即阶跃响应）。（请通过 CCS 软件或示波器观察）记录实验结果，并分析结果(定性分析控制参数对上述控制性能的影响，不需要定量数据的结果)，并简要对比 PMSM 控制性能。(不要修改主函数代码里面变量 PI_Speed 的 kp ki 的值，仅在 Expression 中修改 kp ki 的值，这样程序重启后不影响原来控制器参数与控制运行)

完成步骤：在进行实验前，我们首先查找资料学习了解了 BLDC 控制系统的原理。下图展示了无刷直流电机（BLDC）从设定目标转速到电机实际运行的控制流程。系统通过计算设定转速与实际转速间的差值，利用 PI 控制器生成占空比信号，经换向逻辑和逆变桥驱动电机，同时霍尔传感器提供反馈信号以计算实际转速，形成一个闭环控制系统，确保电机转速达到设定值。



实验的时候，利用 CSS 自带的 graph 功能我们可以绘制出对应的阶跃响应曲线。由于直接从图形界面上观察数据的细微变化存在一定的局限性，为了能够更精确地计算诸如超调量、调节时间等关键参数，我专门编写了一个 MATLAB 函数，对导出的 CSV 文件中的数据进行深入分析和处理。通过这一自动化的分析过程，我们能够获得更为准确和可靠的结果，从而为实验的评估和结论提供坚实的数据支持。

我们在在 graph 图像上右方点击选择 data，将数据以 csv 格式导出，并进一步输 matlab 程序进行系统性能分析。

下图展示了辅助分析函数 Fun_Step_Performance 代码，用于分析和计算给定时间序列和输出序列的标准阶跃响应的性能指标，包括稳态值、上升时间、调整时间和超调量，并根据需要绘制响应曲线。

```
function [ys,tr,ts,tm,ov] = Fun_Step_Performance(t,y,drawflag)
% [ys,tr,ts,ov] = Fun_Step_Performance(t,y) 标准阶跃响应的性能指标求解
% 本程序适用于标准阶跃响应曲线，末尾时间必须已经接近稳态值
% t-y 为阶跃响应的时间-输出配对序列，可由[y,t] = step(sys)求得
% drawflag 为时候作图标志，不输入或输入非 0 值时，默认作图，输入 0 时不做图
% ys 稳态值
% tr 上升时间，默认为 0-90%的上升时间
% ts 调整时间，默认为 2%的调整时间
% tm 为峰值时间
% ov 超调量 %
% e.g.
% sys = tf(1,[1 2*0.5*1 1]);
% [y,t] = step(sys,15);
% [ys,tr,ts,tm,ov] = Fun_Step_Performance(t,y,1);
    if nargin == 2
        drawflag = 1; % 默认绘图
    end
    ys = y(end); % 稳态增益
    [ym, ind] = max(y); % 最大输出
    ov = 100*(ym-ys)/ys; % 超调量
    tm = t(ind); % 峰值时间
    ind2 = length(t);
    delta = 0.02; % 调整时间默认范围 2%
    while t(ind2) > 0
        if abs(y(ind2)-ys) >= delta*ys
```

```
                break
            end
            ind2 = ind2-1;
        end
        ts = t(ind2); % 调整时间
        ind3 = 1;
        while y(ind3) < 0.9*ys
            ind3 = ind3 + 1;
        end
        tr = t(ind3); % 上升时间
        if drawflag ~= 0
            figure
            plot(t,y,'r',LineWidth=2)
            hold on

            plot([tr tr],[0 0.9*ys],'k:')
            plot([tm tm],[0 ym],'k:')
            plot([ts ts],[0 (1-delta)*ys],'k:')
            plot([t(1) t(end)],[ys ys],'k-.')
            xlabel('时间/s')
            ylabel('输出')
            title('阶跃响应曲线')
            ylim([900,1600])
            text(1.1*tr,0.8*ys,['上升时间:  ' num2str(tr) 's'])
            text(1.1*ts,0.75*ys,['调整时间:  ' num2str(ts) 's'])
            if abs(tm-t(end)) > 0.1*tm
                text(1.1 * tm, 1 * ym, ['峰值时间:  ', num2str(tm), 's，超调量:
            else
                text(0.55*tm,0.8*ym,['峰值时间:  ' num2str(tm) 's，超调量:  '
                num2str(ov) '%'])
            end
        text(0.7*t(end),0.95*ys,['稳态值:  ' num2str(ys)])
        disp('%% 阶跃响应指标结果:  ')
        disp(['上升时间:  ' num2str(tr) 's'])
        disp(['调整时间:  ' num2str(ts) 's'])
        disp(['峰值时间:  ' num2str(tm) 's，超调量:  ' num2str(ov) '%'])
        disp(['稳态值:  ' num2str(ys)])
        disp('%% 阶跃响应指标结果显示结束')
    end
```

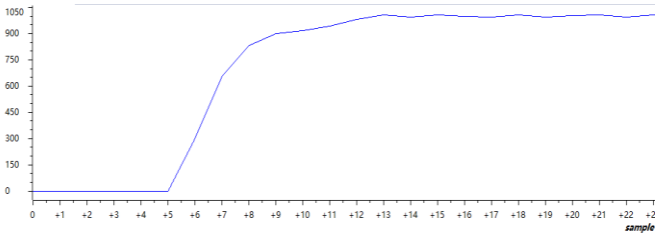
随后我们读取刚才导出的 cvs 文件，调用辅助函数，对系统性能进行分析：

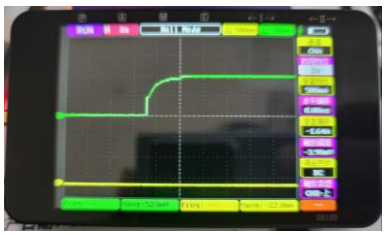
```
% Load data from CSV file
filename = 'speed5.csv';
opts = detectImportOptions(filename);
opts.VariableNamingRule = 'preserve'; % Preserve the original variable name
data = readtable(filename, opts);

% Extract data for 'Sample:float' and 'Data:float'
t = data.('Sample:float');
y = data.('Data:float');

% Set the 'drawflag' parameter
drawflag = 1; % or 0 depending on whether you want to draw plots

% Call the custom function
[ys, tr, ts, tm, ov] = Fun_Step_Performance(t, y, drawflag);
```

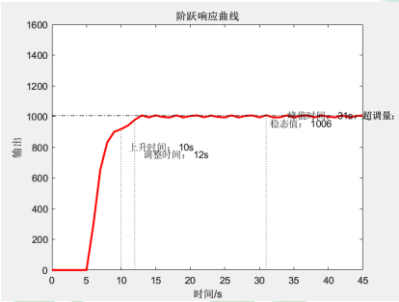
以下我们展示了 4 组不同的 PI 参数对模型性能的影响：

第一组数据

pid_speed.Kp	float	0.100000001
pid_speed.Ki	float	0.00999999978
pid_speed.Kd	float	0.0



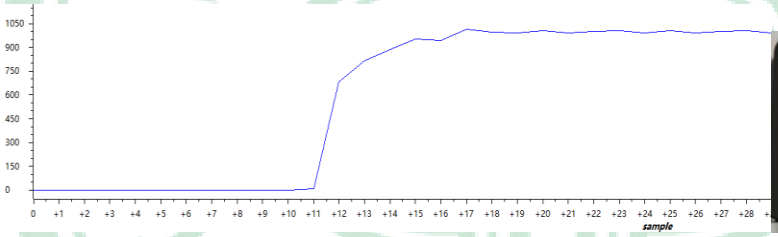


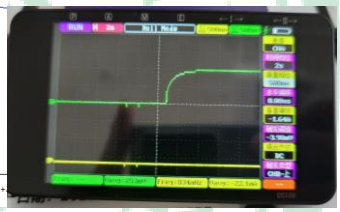


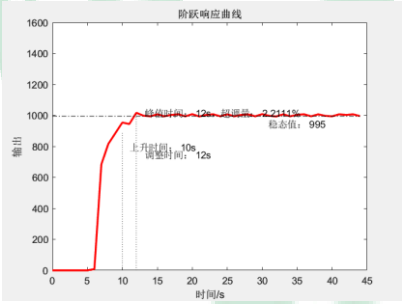

```
>> main
%% 阶跃响应指标结果：
上升时间：  10s
调整时间：  12s
峰值时间：  31s，超调量：  0.19881%
稳态值：  1006
%% 阶跃响应指标结果显示结束
```

第二组数据

pid_speed.Kp	float	0.200000003
pid_speed.Ki	float	0.00999999978
pid_speed.Kd	float	0.0



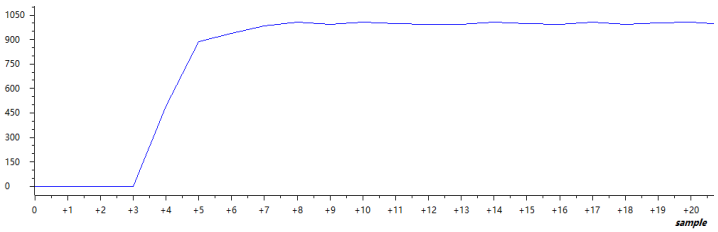


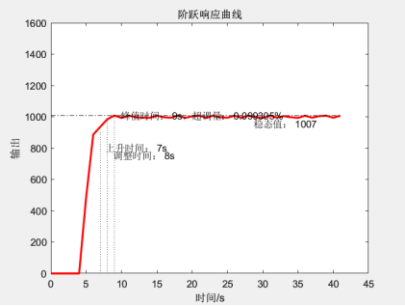



```
>> main
%% 阶跃响应指标结果：
上升时间：  10s
调整时间：  12s
峰值时间：  12s，超调量：  2.2111%
稳态值：  995
%% 阶跃响应指标结果显示结束
```

第三组数据

pid_speed.Kp	float	0.100000001
pid_speed.Ki	float	0.0199999996
pid_speed.Kd	float	0.0






```
>> main
%% 阶跃响应指标结果：
上升时间：  7s
调整时间：  8s
峰值时间：  9s，超调量：  0.099305%
稳态值：  1007
%% 阶跃响应指标结果显示结束
```

动态性能指标表

参数	稳态值	超调量	峰值时间	调节时间	上升时间
K_p=0.1; K;=0.01	1000	0.19%	15s	12s	10s
K=0.2; K;=0.01	1000	2.21%	12s	12s	10s
K=0.1 Kj=0.02	1000	0.09%	9s	8s	7s

随后我们对以上结果进行了原理分析：在 PI 控制中，比例和积分环节都对系统性能产生着一定的影响。

<1> 比例积分环节的作用

比例调节作用：按比例反应系统的偏差,系统一旦出现了偏差,比例调节立即产生 调节作用用以减少偏差。比例作用大,可以加快调节,能迅速反应误差,从而减小稳态误差。但是,比例控制不能消除稳态误差。过大的比例,使系统的稳定性下降,甚至造成系统的不稳定。

积分调节作用：使系统消除稳态误差,提高无差度。积分控制的作用是，只要系统有误差存在,积分调节就进行,积分控制器就不断地积累,输出控制量,直至无差,积分调节停止,积分调节输出一常值。因而，只要有足够的时间，积分控制将能完全消除误差，使系统误差为零，从而消除稳态误差。积分作用的强弱取决于积分时间常数 Ti,Ti 越小,积分作用就越强，积分作用太强会使系统超调加大，甚至使系统出现振荡，反之 Ti 大则积分作用弱。加入积分调节可使系统稳定性下降,动态响应变慢。

<2> 比例积分系数对控制的效果

比例系数（K_p）的影响：在数据中，K_p 的变化对系统性能产生了显著影响。比例控制反映了系统对当前误差的即时响应。较高的 K_p（如 0.2）会增加系统的响应速度，但也可能导致过大的超调量，如在您的数据中从 0.19%上升到 2.21%。这表明 BLDC 系统在较高比例增益下更敏感，可能更快地响应速度变化，但也更容易超调。

积分系数（K_i）的影响：积分控制旨在消除长期的稳态误差。从数据中可以看出，增加 K_i（从 0.01 到 0.02）有助于减少超调量和提高调节速度，这表明在 BLDC 系统中，适当增加积分控制有助于提高长期的稳定性和准确性。

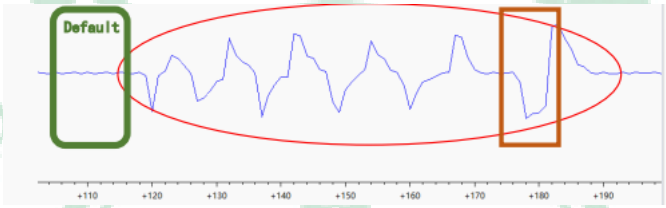
Task4

任务要求：尝试手动添加不同负载（手动，放在电机轴上增加阻力），通过 CCS Graph 观察转速 speed 的 变化，通过示波器观察至少两相电流 iu/iv/iw，记录并分析实验现象。（提示：增加负载，由运动学模型知，加速度 dω/dt 小于零，导致转速降低）

完成步骤：这个实验是要探究负载对电机的一些影响，在正常启动电机以后，尝试添加负载，观察电流、转速的变化，并记录实验现象；观察电流与施加负载的关系。

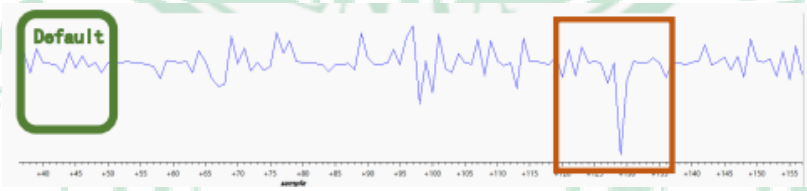
添加负载，通过搓电机轮盘的方式来添加负载，使得转矩产生变化的方式进行。

观察实验现象，则是通过 Expression 中的 graph 功能 Speed：连搓六次，可以观察到转速图中，原本稳定在 1000 左右的值出现突变。

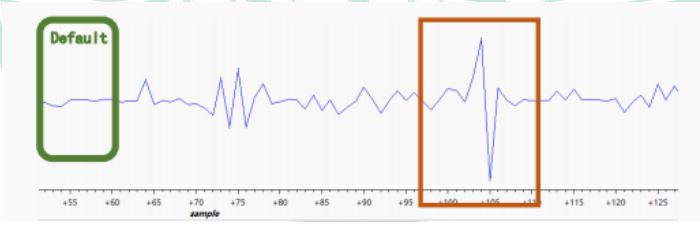


观察下面的电流变化图，可以发现添加负载之后电流变大了，虽然还是在波动，但波动的范围相较于添加负载前有显著上升。

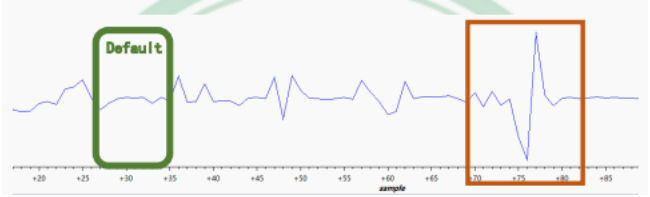
三相电流 iu：



三相电流 iv：

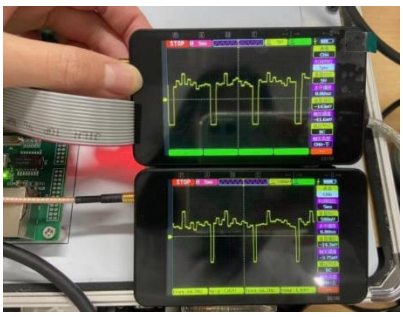
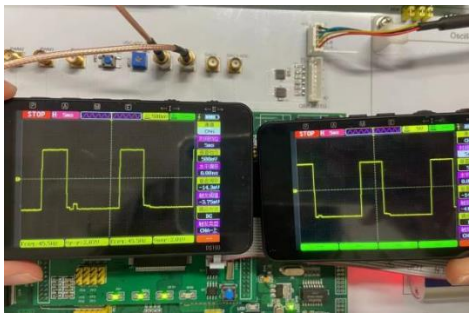


三相电流 iw：



通过手动添加不同负载到电机轴上增加阻力，观察了转速和电流的变化。当负载增

加后，发现三相电流（iu、iv、iw）的波峰值都增加了。

随后我们使用示波器输出了两项电流（iw 和 iv），主要观察到三点明显的现象：1.电流波动明显变大，出现大幅度超调；2.电流频率降低；3.电流占空比降低。

查询资料，这种现象的出现有几个关键因素：

电机负载与电流关系：电机负载增加意味着电机需要克服更大的阻力。为了维持转速，电机需要产生更大的扭矩。在电机中，扭矩与电流成正比，因此，为了产生更大的扭矩以克服增加的负载，电机必须增加流经其绕组的电流，同时使用更小的占空比。这就是 iu、iv、iw 波峰值增加、占空比降低的主要原因。

电机的动态响应：当负载突然增加时，电机的转速会短暂下降。根据运动学模型，加速度 dω/dt 小于零，导致转速降低，而三相电流的频率与电机转速相一致，因此负载的施加会导致电流频率降低。电机控制系统（PID 控制器）会检测到这种转速下降，并增加电流以恢复到设定的转速。这个过程中，电流会有超调的增以补偿负载增加带来的转速下降。

能量守恒和功率需求：从能量守恒的角度看，电机在负载增加时，需要更多的能量来维持相同的输出功率。这额外的能量主要以电流的形式提供，导致电流增加。当电机负载增加时，为了保持或恢复期望的转速，电机的控制系统会自动增加流过电机绕组的电流，这导致了 iu、iv、iw 的波峰值增加。这种现象是电机对负载变化的自然响应，反映了电机控制系统的动态调节能力。

5.实验心得

在本次 BLDC 电机控制实验中，我获得了宝贵的实践经验和深刻的理论认识。实验不仅加深了我对电机控制原理的理解，还提高了我使用 CCS 开发环境进行程序编写、调试和运行的能力。

实验前，我对 BLDC 电机的控制理论有了一定的了解，但通过实际操作，我更加直观地感受到了理论在实际中的应用。例如，通过调整 PID 控制器的参数，我观察到了电机性能的显著变化，这让我深刻认识到了理论对于实践的指导意义以及参数调整对于系统性能的重要影响。

在实验过程中，我学会了如何使用仿真器和 CCS 软件，这些工具对于电机控制系统的开发至关重要。我通过不断尝试和调整，逐渐掌握了如何通过软件来监控和调整电机的运行状

态。此外，我也提高了我的编程能力，特别是在使用 MATLAB 处理数据和编写分析函数的过程。

通过对实验数据的记录和分析，我锻炼了自己的数据分析能力。使用 MATLAB 进行数据处理和图形化展示，让我更加直观地理解了电机运行的特性。例如，通过分析转速与电压的关系，我验证了理论预期，并发现了一些有趣的现象，如负载变化对电流和转矩的影响。

实验中遇到了不少挑战，如电机无法自行启动、参数调整导致的系统不稳定等。面对这些问题，我学会了如何通过查阅资料、分析数据和调整代码来解决。这个过程提高了我的问题解决能力，也让我意识到了在工程实践中遇到问题并寻找解决方案的重要性。

总的来说，这次实验是一次非常宝贵的学习经历。我不仅加深了对 BLDC 电机控制的理解，还提升了自己的实践操作能力和问题解决能力。我相信这些经验和技能将在我的未来学习和工作中发挥重要作用。