BLDC 方波速度控制

实验报告

学院 机电工程与自动化学院

姓名 吴俊达

学号 210320621

日期 2024.5.6

目录

一、BLDC 方波速度开环控制实验

1.1 程序流程图

代码里有编号，是指各项子任务：

1. LCD 初始化及显示	2. TIM3（测速定时器）
3. ADC & DMA	4. TIM2: HALL（中断读取霍尔状态）
5. 换向（Change Direction）	6. TIM8: BREAK（刹车中断）
7. 启停（START & STOP，状态机）	8. 增量式 PID 控制器计算

1) 画出程序流程图。

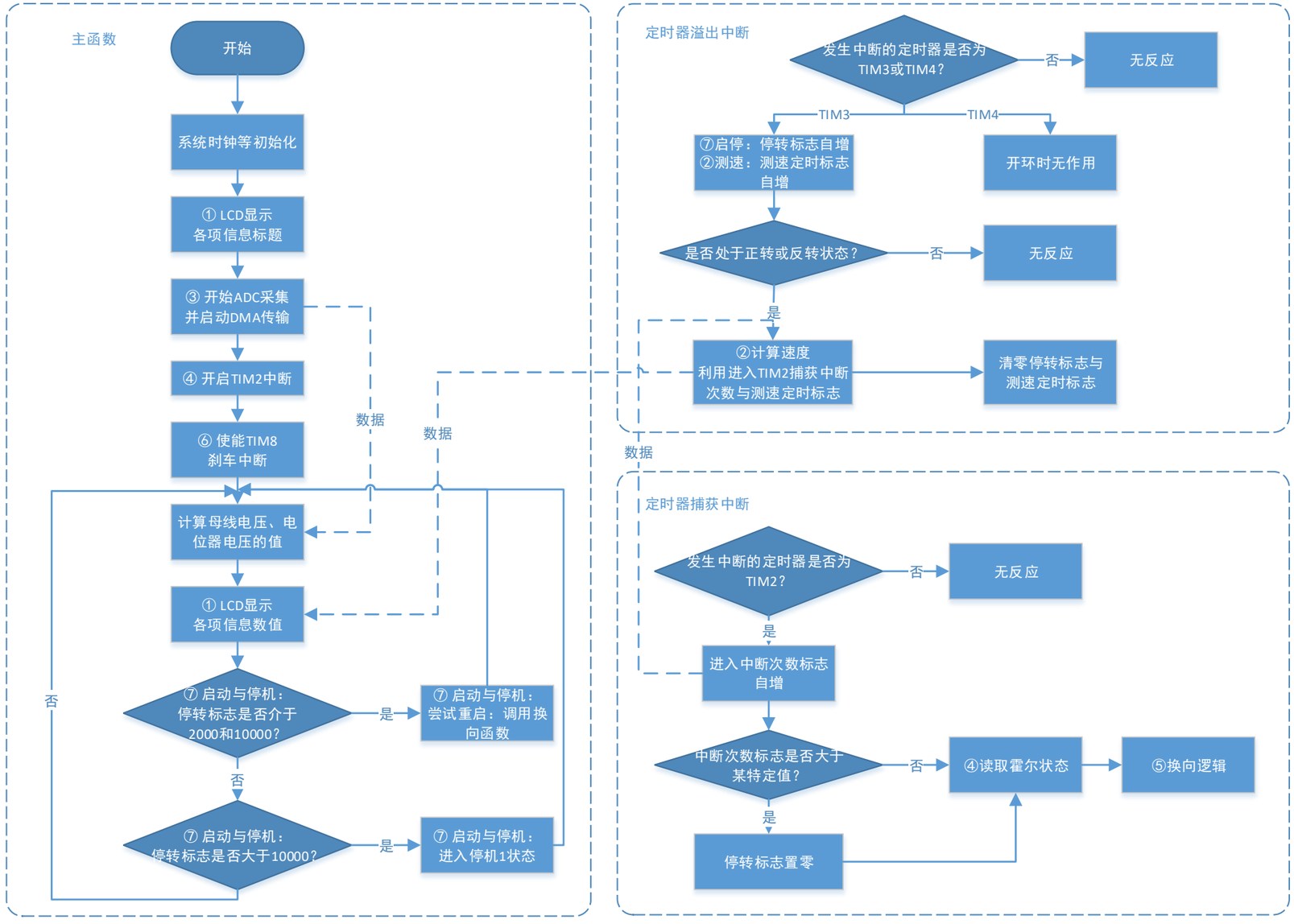


图 1：主函数、定时器溢出中断与捕获中断逻辑

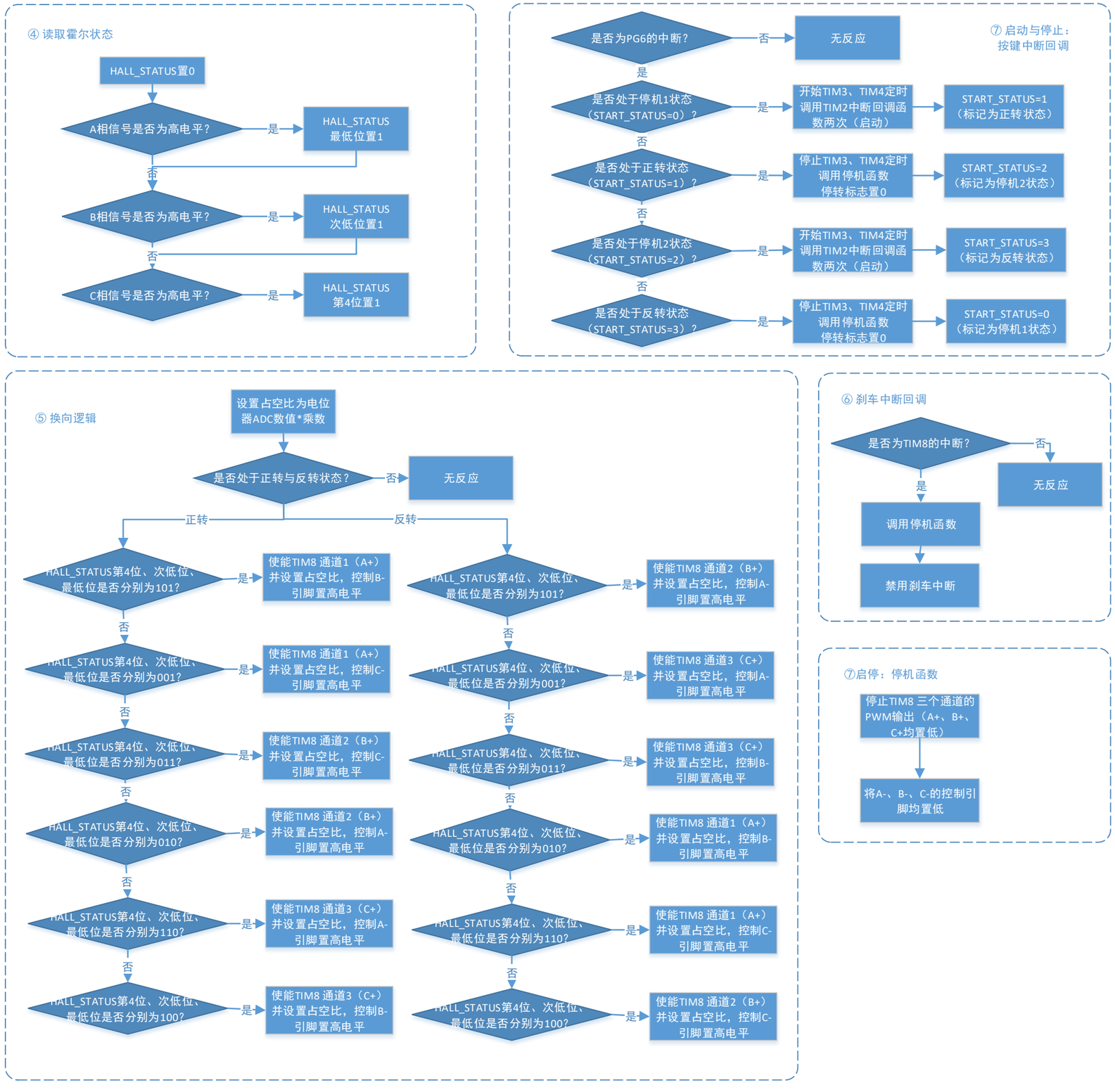


图 2：霍尔传感器读取、换向逻辑、刹车中断回调、切换状态（采用按键中断回调）、停机函数逻辑

2) 对所画流程图进行说明，尽量详细的描述程序设计。

已经在流程图中说明得很清楚，此处从略。

1.2 功能代码

我的开环和闭环代码是写在一起的，利用变量 Closed_loop 来区分，此变量置 0 时为开环，置 1 时为闭环。此处系统生成的代码已经删除。

代码里有编号，是指各项子任务：

1. LCD 初始化及显示	2. TIM3（测速定时器）
3. ADC & DMA	4. TIM2: HALL（中断读取霍尔状态）
5. 换向（Change Direction）	6. TIM8: BREAK（刹车中断）
7. 启停（START & STOP，状态机）	8. 增量式 PID 控制器计算

具体代码如下：

```
/* USER CODE BEGIN Includes */
#include "stdio.h"
#include "string.h"
#include "stm32_2.8_lcd.h" // 引入必要的头文件
/* USER CODE END Includes */

/* USER CODE BEGIN PV */
const uint8_t Closed_loop = 0; // 开环闭环标志，1 表示闭环
uint32_t ADC_VOLTAGE_32[32] = {0}; // 电位器电压接收 buffer
uint32_t BUS_VOLTAGE_32[32] = {0}; // 母线电压接收 buffer
uint32_t Duty_Cycle = 0; // 占空比，最大值为 TIM8 最大计数值
float ADC_VOLTAGE, BUS_VOLTAGE, Duty_Cycle_Display; // 用于转换与显示的中间变量
uint8_t HALL_STATUS = 0; // 霍尔传感器状态
uint8_t START_STATUS = 0; // 电机状态，0/停机 1，1/正转，2/停机 2，3/反转

uint16_t TIME_GAP = 0;  // 缓转/停转标志
uint8_t TIME_GAP_CNT = 0;  // 用于计算速度
uint8_t TIME_TURN_CNT = 0; // 用于计算速度

// PID
float Speed = 0;
float targetSpeed = 0;
// 速度环 PID 参数
#define PERIOD 10 // 控制周期
float u1, ek11, ek12; // 增量式 PID 式中三项，分别用于记录上一控制时刻 PWM、上一控制时刻偏差和上上控制时刻
```

偏差

```
float velocity_kp= 0.0003;  // 比例系数
float velocity_ti= 0.7;  // 积分时间
float velocity_td= 0;  // 微分时间
uint16_t PWM = 0;  // 调速 PWM 值
/* USER CODE END PV */

/* USER CODE BEGIN PFP */
void ChangeDirection(void); // 换向
void StopMotor(void); // 停机
float pidController(void); // PID 计算
/* USER CODE END PFP */

int main(void)
{
  /* USER CODE BEGIN 2 */
    // 1. LCD Initialize
    STM32_LCD_Init();
    LCD_Clear(BackColor);
    LCD_SetTextColor(Red);
    LCD_DisplayStringLine(1,"PF9 VOLTAGE");
    LCD_DisplayStringLine(2,"BUS VOLTAGE");
    if(Closed_loop) LCD_DisplayStringLine(3,"PWM Value");
    else LCD_DisplayStringLine(3,"Duty Cycle");
    LCD_DisplayStringLine(4,"SPEED(*.1rps)");
    LCD_DisplayStringLine(5,"Target SPEED");
    // 3. ADC & DMA
    HAL_NVIC_DisableIRQ(DMA2_Stream1_IRQn);  // 禁用 DMA 中断
    HAL_NVIC_DisableIRQ(DMA2_Stream0_IRQn);  // 禁用 DMA 中断
    HAL_ADC_Start(&hadc1);
    HAL_ADC_Start(&hadc3);
    HAL_ADC_Start_DMA(&hadc1, &BUS_VOLTAGE_32[0], 1);  // Continuous Conversion mode, BUS VOLTAGE
    HAL_ADC_Start_DMA(&hadc3, &ADC_VOLTAGE_32[0], 1);  // Continuous Conversion mode, Potentialmeter
VOLTAGE
    // 4. TIM2: HALL
    HAL_TIM_IC_Start_IT(&htim2, TIM_CHANNEL_1);
    HAL_TIM_IC_Start_IT(&htim2, TIM_CHANNEL_2);
    HAL_TIM_IC_Start_IT(&htim2, TIM_CHANNEL_3);
    // Duty_Cycle = 2000; // 定占空比，调试时使用
    // 6. TIM8: BREAK
    __HAL_TIM_ENABLE_IT(&htim8, TIM_IT_BREAK);  // 使能刹车
  /* USER CODE END 2 */
```

```
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
        ADC_VOLTAGE = ADC_VOLTAGE_32[0] * 33 / 40.96;  // 电位器电压(*0.01V)，最大值为 330，即对应 3.3V
        BUS_VOLTAGE = BUS_VOLTAGE_32[0]; // 母线电压(*0.1V)，显示数值约二百，也就是二十多伏
//      BUS_VOLTAGE = BUS_VOLTAGE_32[0] * 3.3* 11.2732 /7.32 / 40.96;
        if(Closed_loop == 0) Duty_Cycle_Display = ADC_VOLTAGE_32[0] / 4.096;  // 开环时的占空比显示，最大值为 1000
        // 1. LCD Display
        LCD_Draw_NUM(24,120, (uint16_t) ADC_VOLTAGE);  // 显示 ADC 采集电压
        LCD_Draw_NUM(48,120, (uint16_t) BUS_VOLTAGE);  // 显示母线电压
        if(Closed_loop) LCD_Draw_NUM(72,120, (uint16_t) PWM); // 闭环 PWM 显示，最大值 8399
        else LCD_Draw_NUM(72,120, (uint16_t) Duty_Cycle_Display);  // 开环 占空比显示
        LCD_Draw_NUM(96,120, (uint16_t) Speed); // 速度显示，单位*0.1rps，例:显示 330 即为 33rps
        if(Closed_loop) LCD_Draw_NUM(120,120, (uint16_t)targetSpeed); // 目标速度显示，单位说明同上
        LCD_Draw_NUM(144,120, (uint16_t)TIME_GAP); // 停转标志，每进入 3 次转动中断重置 1 次，若长期不重置则进入下面逻辑
        if(Closed_loop) printf("%f,%f\r\n",Speed,targetSpeed); // 闭环时，发送速度与目标速度数据
        if(TIME_GAP > 2000 && TIME_GAP < 10000){ // 停转标志较大，尝试重启
            ChangeDirection();
            HAL_Delay(4);
            ChangeDirection();
        }
        else if(TIME_GAP > 10000){ // STOP
            HAL_TIM_Base_Stop_IT(&htim3);
            TIME_GAP = 0;
            START_STATUS = 2;
            StopMotor();
        }
        if(START_STATUS == 1){
            LCD_DisplayStringLine(7,"FORWARD"); // 显示正转
        }
        else if(START_STATUS == 3){
            LCD_DisplayStringLine(7,"REVERSE"); // 显示反转
        }
        else LCD_DisplayStringLine(7,"STOP   "); // 显示停止
        HAL_Delay(200); // 每秒钟约更新 5 次
```

```
  }
  /* USER CODE END 3 */
}

// 中间系统生成的时钟初始化、外部中断初始化代码略去

/* USER CODE BEGIN 4 */
// 2. TIM3 & 8. PID Controller
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
    UNUSED(htim);
    if(htim->Instance == TIM3){
        ++TIME_GAP; // 停转标志增加
        ++TIME_GAP_CNT;
        // 定时，计量进中断次数来求速度
        if((START_STATUS == 1 || START_STATUS == 3) && TIME_GAP_CNT == 50){
                Speed = 10000 / 24 / TIME_GAP_CNT * TIME_TURN_CNT; // 用于显示与 PID 计算的速度，单位
(*0.1rps)，例如 380 代表 38rps
            //else Speed = 0;
            TIME_TURN_CNT = 0;
            TIME_GAP_CNT = 0;
        }
    }
    if(htim->Instance == TIM4 && Closed_loop){
        targetSpeed = ADC_VOLTAGE_32[0] / 6; // 目标速度(*0.1rps)，最大值 4096/6
        PWM = (uint16_t) pidController(); // PID 控制器
    }
}
// 4. TIM2: HALL
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim){
    UNUSED(htim);
    if(htim->Instance == TIM2){
            ++TIME_TURN_CNT;
            if(TIME_TURN_CNT >= 3) TIME_GAP = 0; // 进入 hall 中断 3 次，停转标志置 0（正在旋转）
        HALL_STATUS = 0;
        if(HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_15)==GPIO_PIN_SET) //A
                HALL_STATUS |= 0x01;
        if(HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_3)==GPIO_PIN_SET) //B
                HALL_STATUS |= 0x04;
        if(HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_10)==GPIO_PIN_SET) //C
                HALL_STATUS |= 0x08;
        ChangeDirection();
    }
```

```
}

// 5. HALL Change Direction
void ChangeDirection(){
    if(Closed_loop) Duty_Cycle = PWM; // Close loop
    else Duty_Cycle = ADC_VOLTAGE_32[0]*2; // Open loop，最大值为 4096*乘数
    // PWM 计数器最大值 8400 时，设定乘数为 2；PWM 计数器最大值为 4200 时，设定乘数为 1
if(START_STATUS == 1){
    if((HALL_STATUS & 0x08) && !(HALL_STATUS & 0x04) && (HALL_STATUS & 0x01) ){
        HAL_TIM_PWM_Start(&htim8, TIM_CHANNEL_1);//A+
        HAL_TIM_PWM_Stop(&htim8, TIM_CHANNEL_2);//B+
        HAL_TIM_PWM_Stop(&htim8, TIM_CHANNEL_3);//C+
        __HAL_TIM_SetCompare(&htim8, TIM_CHANNEL_1, Duty_Cycle);
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, GPIO_PIN_RESET);//A-
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_SET);//B-
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_RESET);//C-
    }
    else if(!(HALL_STATUS & 0x08) && !(HALL_STATUS & 0x04) && (HALL_STATUS & 0x01) ){
        HAL_TIM_PWM_Start(&htim8, TIM_CHANNEL_1);//A+
        HAL_TIM_PWM_Stop(&htim8, TIM_CHANNEL_2);//B+
        HAL_TIM_PWM_Stop(&htim8, TIM_CHANNEL_3);//C+
        __HAL_TIM_SetCompare(&htim8, TIM_CHANNEL_1, Duty_Cycle);
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, GPIO_PIN_RESET);//A-
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_RESET);//B-
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);//C-
    }
    else if(!(HALL_STATUS & 0x08) && (HALL_STATUS & 0x04) && (HALL_STATUS & 0x01) ){
        HAL_TIM_PWM_Stop(&htim8, TIM_CHANNEL_1);//A+
        HAL_TIM_PWM_Start(&htim8, TIM_CHANNEL_2);//B+
        HAL_TIM_PWM_Stop(&htim8, TIM_CHANNEL_3);//C+
        __HAL_TIM_SetCompare(&htim8, TIM_CHANNEL_2, Duty_Cycle);
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, GPIO_PIN_RESET);//A-
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_RESET);//B-
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);//C-
    }
    else if(!(HALL_STATUS & 0x08) && (HALL_STATUS & 0x04) && !(HALL_STATUS & 0x01) ){
        HAL_TIM_PWM_Stop(&htim8, TIM_CHANNEL_1);//A+
        HAL_TIM_PWM_Start(&htim8, TIM_CHANNEL_2);//B+
        HAL_TIM_PWM_Stop(&htim8, TIM_CHANNEL_3);//C+
        __HAL_TIM_SetCompare(&htim8, TIM_CHANNEL_2, Duty_Cycle);
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, GPIO_PIN_SET);//A-
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_RESET);//B-
```

```
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_RESET);//C-
    }
    else if((HALL_STATUS & 0x08) && (HALL_STATUS & 0x04) && !(HALL_STATUS & 0x01) ){
        HAL_TIM_PWM_Stop(&htim8, TIM_CHANNEL_1);//A+
        HAL_TIM_PWM_Stop(&htim8, TIM_CHANNEL_2);//B+
        HAL_TIM_PWM_Start(&htim8, TIM_CHANNEL_3);//C+
        __HAL_TIM_SetCompare(&htim8, TIM_CHANNEL_3, Duty_Cycle);
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, GPIO_PIN_SET);//A-
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_RESET);//B-
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_RESET);//C-
    }
    else if((HALL_STATUS & 0x08) && !(HALL_STATUS & 0x04) && !(HALL_STATUS & 0x01) ){
        HAL_TIM_PWM_Stop(&htim8, TIM_CHANNEL_1);//A+
        HAL_TIM_PWM_Stop(&htim8, TIM_CHANNEL_2);//B+
        HAL_TIM_PWM_Start(&htim8, TIM_CHANNEL_3);//C+
        __HAL_TIM_SetCompare(&htim8, TIM_CHANNEL_3, Duty_Cycle);
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, GPIO_PIN_RESET);//A-
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_SET);//B-
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_RESET);//C-
    }
}

if(START_STATUS == 3){
    if((HALL_STATUS & 0x08) && !(HALL_STATUS & 0x04) && (HALL_STATUS & 0x01) ){
        HAL_TIM_PWM_Stop(&htim8, TIM_CHANNEL_1);//A+
        HAL_TIM_PWM_Start(&htim8, TIM_CHANNEL_2);//B+
        HAL_TIM_PWM_Stop(&htim8, TIM_CHANNEL_3);//C+
        __HAL_TIM_SetCompare(&htim8, TIM_CHANNEL_2, Duty_Cycle);
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, GPIO_PIN_SET);//A-
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_RESET);//B-
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_RESET);//C-
    }
    else if(!(HALL_STATUS & 0x08) && !(HALL_STATUS & 0x04) && (HALL_STATUS & 0x01) ){
        HAL_TIM_PWM_Stop(&htim8, TIM_CHANNEL_1);//A+
        HAL_TIM_PWM_Stop(&htim8, TIM_CHANNEL_2);//B+
        HAL_TIM_PWM_Start(&htim8, TIM_CHANNEL_3);//C+
        __HAL_TIM_SetCompare(&htim8, TIM_CHANNEL_3, Duty_Cycle);
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, GPIO_PIN_SET);//A-
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_RESET);//B-
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_RESET);//C-
    }
    else if(!(HALL_STATUS & 0x08) && (HALL_STATUS & 0x04) && (HALL_STATUS & 0x01) ){
```

```
        HAL_TIM_PWM_Stop(&htim8, TIM_CHANNEL_1);//A+
        HAL_TIM_PWM_Stop(&htim8, TIM_CHANNEL_2);//B+
        HAL_TIM_PWM_Start(&htim8, TIM_CHANNEL_3);//C+
        __HAL_TIM_SetCompare(&htim8, TIM_CHANNEL_3, Duty_Cycle);
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, GPIO_PIN_RESET);//A-
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_SET);//B-
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_RESET);//C-
    }
    else if(!(HALL_STATUS & 0x08) && (HALL_STATUS & 0x04) && !(HALL_STATUS & 0x01) ){
        HAL_TIM_PWM_Start(&htim8, TIM_CHANNEL_1);//A+
        HAL_TIM_PWM_Stop(&htim8, TIM_CHANNEL_2);//B+
        HAL_TIM_PWM_Stop(&htim8, TIM_CHANNEL_3);//C+
        __HAL_TIM_SetCompare(&htim8, TIM_CHANNEL_1, Duty_Cycle);
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, GPIO_PIN_RESET);//A-
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_SET);//B-
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_RESET);//C-
    }
    else if((HALL_STATUS & 0x08) && (HALL_STATUS & 0x04) && !(HALL_STATUS & 0x01) ){
        HAL_TIM_PWM_Start(&htim8, TIM_CHANNEL_1);//A+
        HAL_TIM_PWM_Stop(&htim8, TIM_CHANNEL_2);//B+
        HAL_TIM_PWM_Stop(&htim8, TIM_CHANNEL_3);//C+
        __HAL_TIM_SetCompare(&htim8, TIM_CHANNEL_1, Duty_Cycle);
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, GPIO_PIN_RESET);//A-
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_RESET);//B-
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);//C-
    }
    else if((HALL_STATUS & 0x08) && !(HALL_STATUS & 0x04) && !(HALL_STATUS & 0x01) ){
        HAL_TIM_PWM_Stop(&htim8, TIM_CHANNEL_1);//A+
        HAL_TIM_PWM_Start(&htim8, TIM_CHANNEL_2);//B+
        HAL_TIM_PWM_Stop(&htim8, TIM_CHANNEL_3);//C+
        __HAL_TIM_SetCompare(&htim8, TIM_CHANNEL_2, Duty_Cycle);
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, GPIO_PIN_RESET);//A-
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_RESET);//B-
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);//C-
    }
}
}

void StopMotor(void){
        HAL_TIM_PWM_Stop(&htim8, TIM_CHANNEL_1);//A+
        HAL_TIM_PWM_Stop(&htim8, TIM_CHANNEL_2);//B+
        HAL_TIM_PWM_Stop(&htim8, TIM_CHANNEL_3);//C+
```

```
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, GPIO_PIN_RESET);//A-
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_RESET);//B-
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_RESET);//C-
}

// 6. TIM8: BREAK
void HAL_TIMEx_BreakCallback(TIM_HandleTypeDef *htim){
    if(htim->Instance==TIM8){
        StopMotor();
        __HAL_TIM_DISABLE_IT(&htim8, TIM_IT_BREAK);
    }
}

// 7. START AND STOP
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin){
    if(GPIO_Pin== GPIO_PIN_10){
        for(uint32_t i = 0; i<500000; ++i); // 软件消抖
        // HAL_Delay(10);
        if(HAL_GPIO_ReadPin(GPIOG,GPIO_PIN_10)==GPIO_PIN_RESET){
            if(START_STATUS == 0){ // 停转 1，准备正转
                // 2. TIM3
                HAL_TIM_Base_Start_IT(&htim3);
                // 8. PWM
                HAL_TIM_Base_Start_IT(&htim4);
                // START
                START_STATUS = 1;
                HAL_TIM_IC_CaptureCallback(&htim2);
                // HAL_Delay(4);
                HAL_TIM_IC_CaptureCallback(&htim2);
            }
            else if(START_STATUS == 1){ // 正转时按下按钮，进入停车 2
                HAL_TIM_Base_Stop_IT(&htim3);
                HAL_TIM_Base_Stop_IT(&htim4);
                TIME_GAP = 0;
                StopMotor();
                START_STATUS = 2;
            }
            else if(START_STATUS == 2){ // 停车 2 按下按钮，准备反转
                // 2. TIM3
                HAL_TIM_Base_Start_IT(&htim3);
                // 8. PWM
                HAL_TIM_Base_Start_IT(&htim4);
```

```
                // START
                START_STATUS = 3;
                HAL_TIM_IC_CaptureCallback(&htim2);
                // HAL_Delay(4);
                HAL_TIM_IC_CaptureCallback(&htim2);
            }
            else { // 反转按下按钮，进入停车 1
                HAL_TIM_Base_Stop_IT(&htim3);
                HAL_TIM_Base_Stop_IT(&htim4);
                TIME_GAP = 0;
                StopMotor();
                START_STATUS = 0;
            }
        }
    }
}

// 8. Incremental PID Controller
float pidController()
{
  float q0 = velocity_kp * (1 + PERIOD / velocity_ti + velocity_td / PERIOD);
  float q1 = -velocity_kp * (1 + 2 * velocity_td / PERIOD);
  float q2 = velocity_kp * velocity_td / PERIOD;
  float ek10;
  ek10 = targetSpeed - Speed;
  u1 = u1 + q0 * ek10 + q1 * ek11 + q2 * ek12;
  if (u1 > 8399)
    u1 = 8399;
  if (u1 <= 0)
    u1 = 0;
  ek12 = ek11;
  ek11 = ek10;
  return u1;
}

/* USER CODE END 4 */
```

在 stm32f4xx_hal_msp.c 中的代码：

```
/* USER CODE BEGIN Includes */
#include "stdio.h"
extern UART_HandleTypeDef huart1;
```

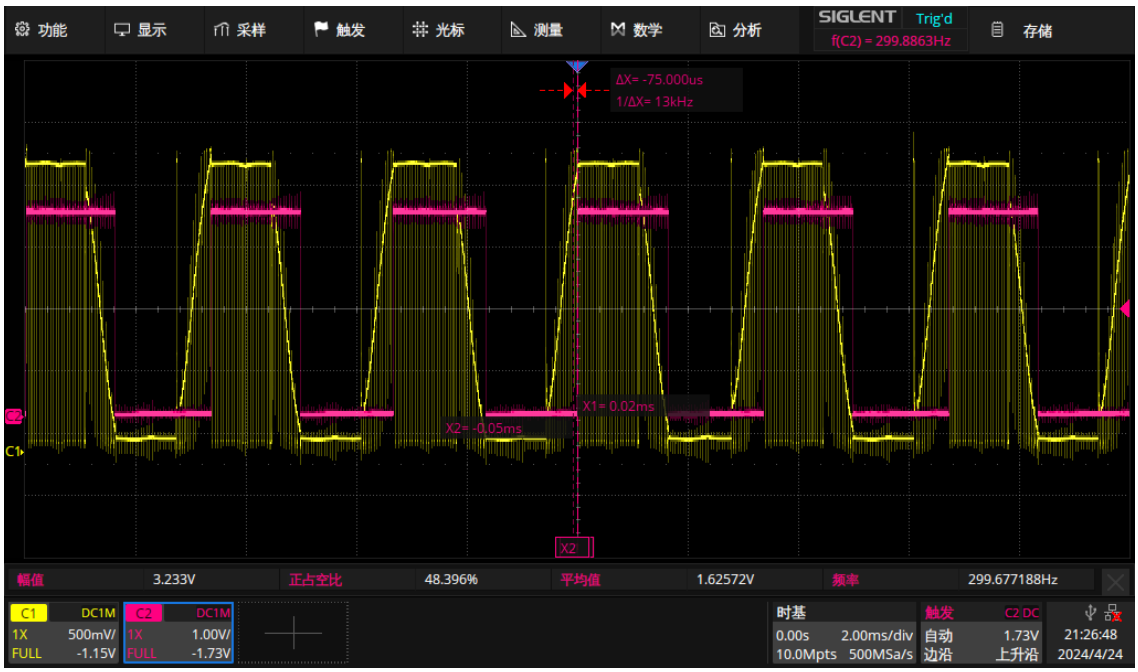
```
/* USER CODE END Includes */

/* USER CODE BEGIN 0 */
int fputc(int ch, FILE *f){
    HAL_UART_Transmit(&huart1, (uint8_t * )&ch, 1, 0xffff);
    return ch;
}
int fgetc(FILE *f){
    uint8_t ch = 0;
    HAL_UART_Receive(&huart1, (uint8_t * )&ch, 1, 0xffff);
    return ch;
}
/* USER CODE END 0 */
```

1.3 波形测量

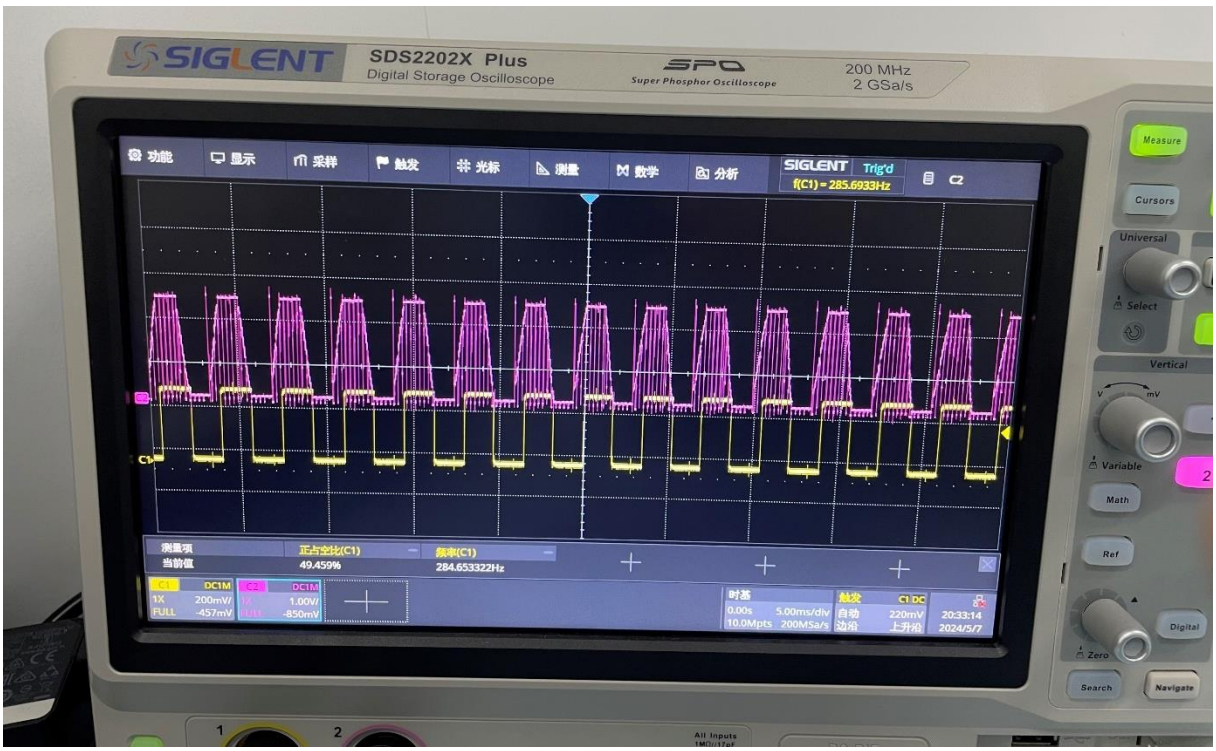
将不同 PWM 载波频率下的反电动势和 HALL 信号波形记录在实验报告中，并思考 PWM 载波频率对电机转动的影响。

1) 调整PWM载波频率，测量所得波形如下：

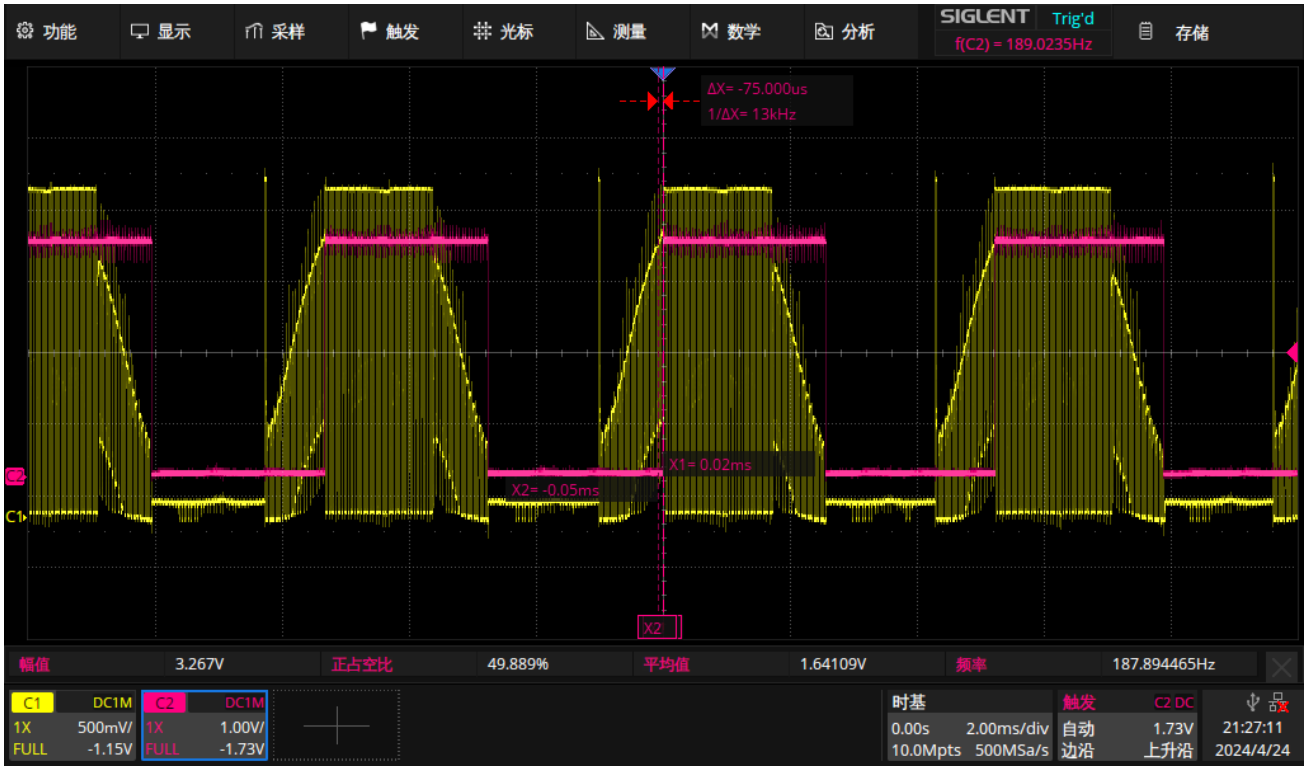
载波频率 f_1＝10kHz



载波频率 f_2＝2kHz



载波频率 f_3＝15kHz



2) 说说载波频率对电机转动影响。

载波频率过高：由于电机是感性元件，其电流变化存在过渡过程，因此若控制频率过高，则其电流尚未到达稳态时，控制模态就已改变，可能使电机转矩变小，甚至不能启动。

载波频率过低：电机控制不平顺（有段落感），且有啸叫声。

1.4 实验总结

阐述一下自己在开发过程中遇到的主要问题，及最终解决方法。

1. 卡死在 DMA 读取数据里（接收数据正确，但电机控制信号未发出），后来加入禁用 DMA 全局中断的两行代码，即

```
HAL_NVIC_DisableIRQ(DMA2_Stream1_IRQn);  // 禁用 DMA 中断
HAL_NVIC_DisableIRQ(DMA2_Stream0_IRQn);  // 禁用 DMA 中断
```

方解决问题。

2. 在按钮的外部中断中调用 `HAL_Delay(4);`，程序卡死，后改用软件中断

```
for(uint32_t i = 0; i<500000; ++i); // 软件消抖
```

实现，并将代码的优化层级从-O3 改为-O1 防止该语句被优化。

二、BLDC 方波速度闭环控制实验

2.1 程序流程图

1) 画出程序流程图。（橙色表示和开环时**不同之处**）

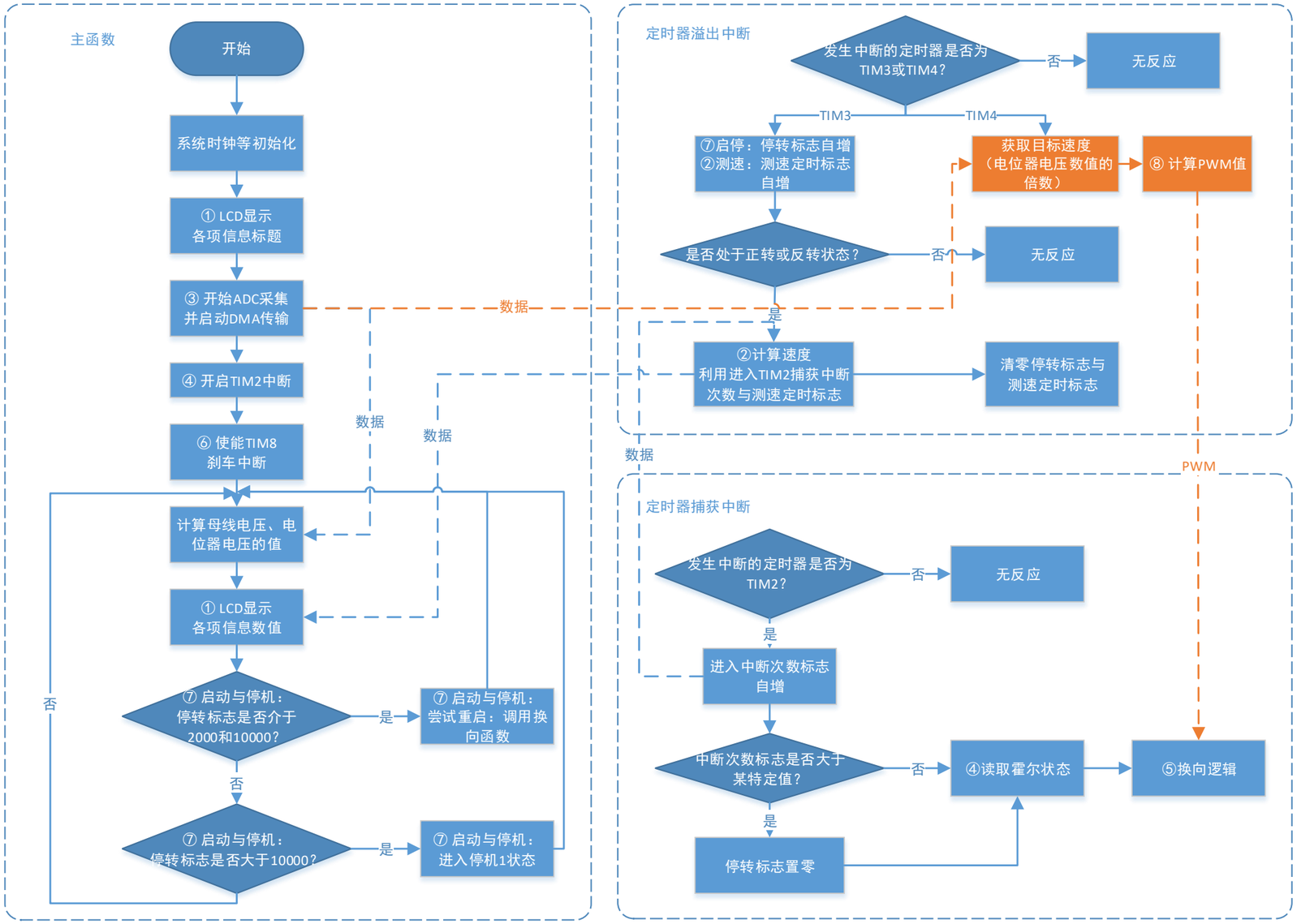


图 1：主函数、定时器溢出中断与捕获中断逻辑

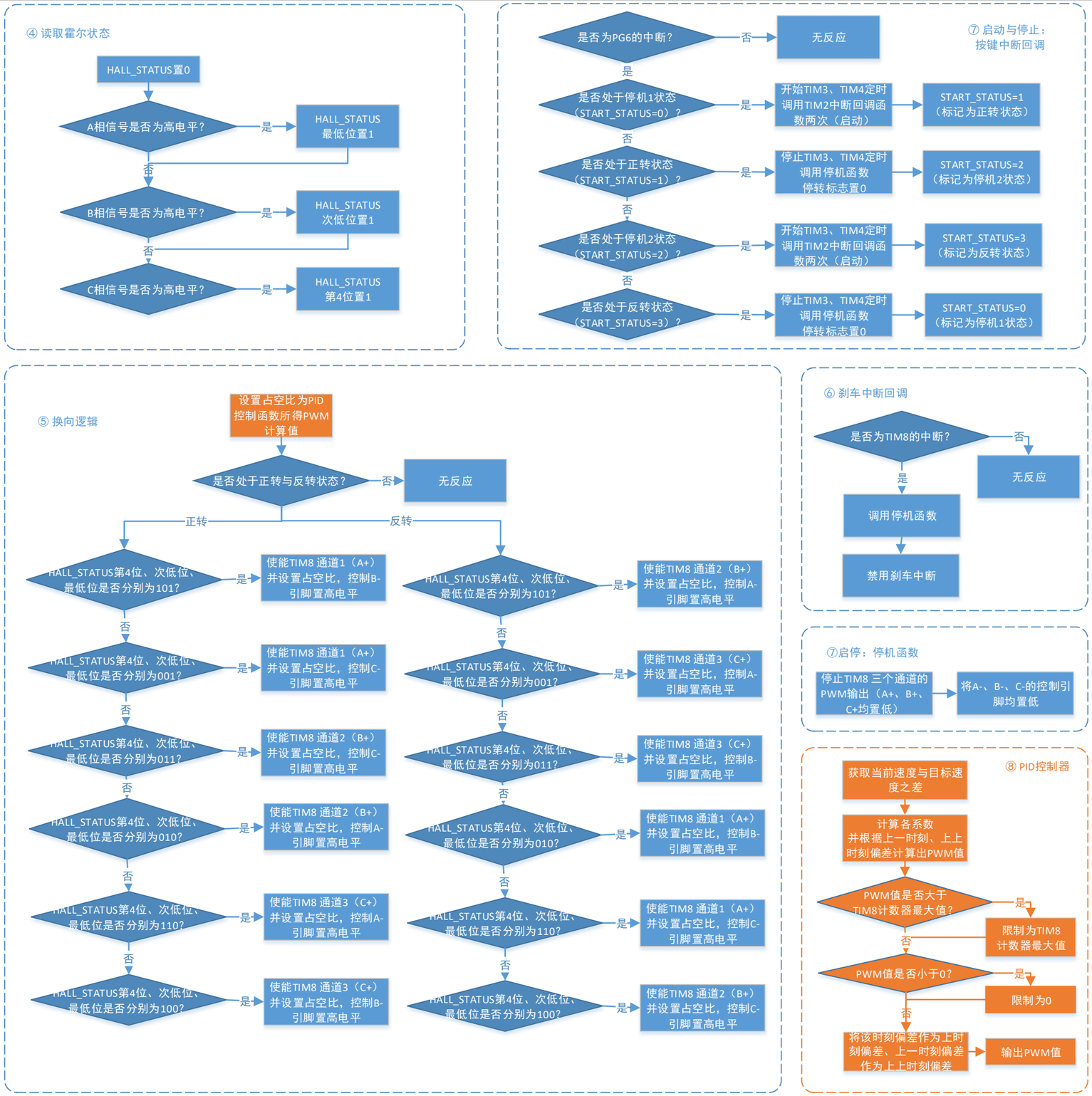


图 2：霍尔传感器读取、换向逻辑、刹车中断回调、切换状态（采用按键中断回调）、停机函数、增量式 PID 控制函数逻辑

2) 对所画流程图进行说明，尽量详细的描述程序设计。

已经在流程图中说明得很清楚，此处从略。

2.2 功能代码

我的开环和闭环代码是写在一起的，利用变量 Closed_loop 来区分，此变量置 0 时为开环，置 1 时为闭环。故代码详见上面的 1.2 节，此处不再重新书写。

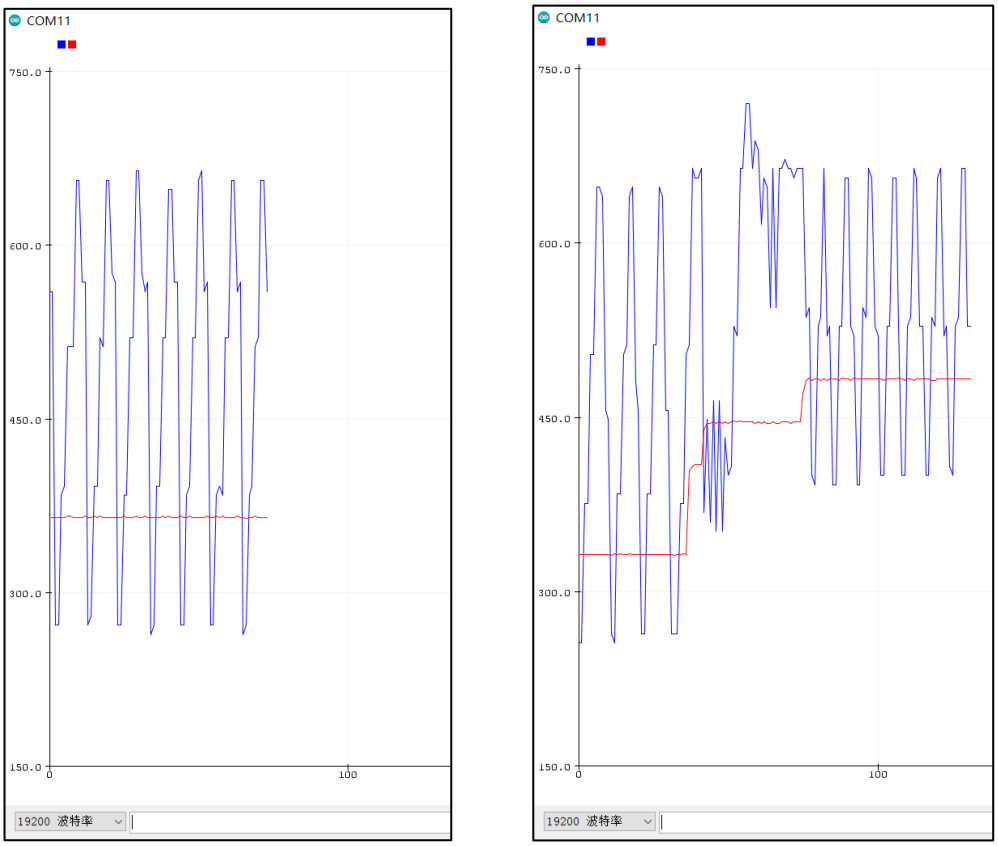
2.3 PID 参数调试

调节 PID 参数与采样周期 Ts，记录不同参数配置下，曲线的变化。选择最好的一组曲线对应的参数作为调参结果。

1) 调整PID参数与采样周期Ts，测量所得波形如下：

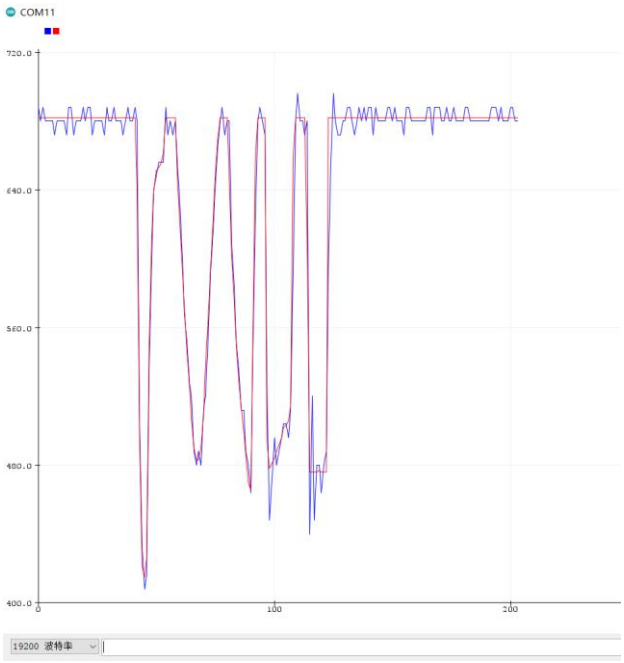
使用的是增量式PID，参数有：K_p, T_i, T_d。

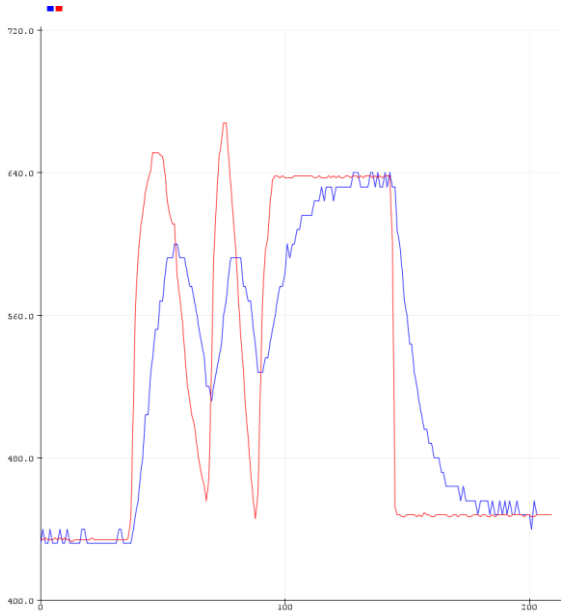
波形中：红线：目标速度（×0.1rps）；蓝线：实测转速（×0.1rps）



左图： K_p=0.01, T_i=0.1, T_d=0, T_s=10ms;　右图：K_p=0.005, T_i=0.1, T_d=0, T_s=10ms;

先将积分时间 T_i 取为 0.1 左右（使积分作用尽量小），T_d 取为 0，然后将 K_p 调整为 0.01 左右，发现出现严重的振荡现象。然后逐渐减小 K_p，发现振荡逐渐减小。最后，K_p 取为 0.0002 左右效果较好，如下图。





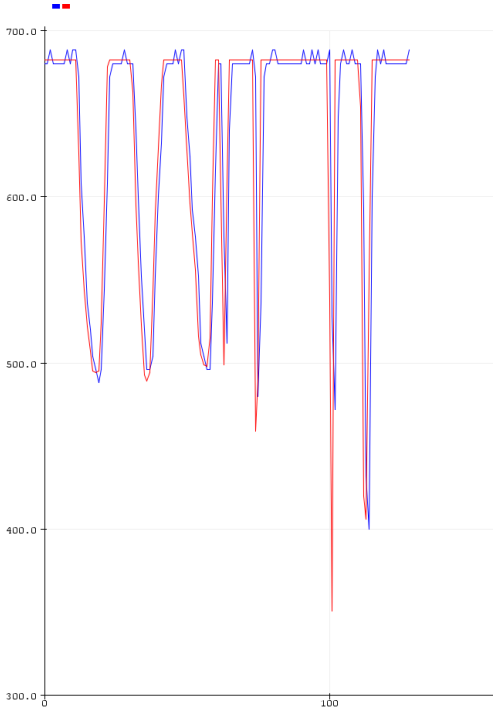
左图：K_p=0.0002, T_i=0.1, T_d=0, T_s=10ms;　右图：K_p=0.00005, T_i=1, T_d=0, T_s=10ms

但是，在目标速度骤减时，速度降低过快，可能导致电机直接停机。于是，尝试了**较大的积分时间，**系统快速性降低，但缓解了电机因减速过快而直接停机的问题，电机重新启动的性能也较好。于是适当增大比例系数以改善快速性。我还尝试了积分时间过大且增益过小的情形，发现系统的动态响应性能差；如上右图。

2)　最终结果

K_p=0.0003, T_i=0.5, T_d=0, T_s=10ms，跟踪效果如下图

波形中：红线：目标速度（×0.1rps）；蓝线：实测转速（×0.1rps）



3) 简述参数调试过程

见上，与图形一同说明了。

2.4 实验总结

阐述一下自己在开发过程中遇到的主要问题，及最终解决方法。

1. 每更改一次参数就烧录一次，调参效率低。

解决方法：打开 debug 中的 watch 视窗，输入三个参数变量名，即可实时更改其值，最后在代码中更改即可。