# 基于自适应算法和增量式PID算法的模拟直升飞机控制系统

作者：自动化96 薛荣坤（2196113513）

# 1 控制系统硬件

模拟直升机垂直升降控制系统主要由C8051F020单片机、按键和显示等模块以及直升机垂直升降模拟对象组成。

## 1.1 单片机系统

显示功能由液晶和数码管实现，液晶屏和数码管分别实现显示控制主菜单界面、当前霍尔电压的数值及变化曲线等功能，按键用于切换液晶屏显示内容、设置PID参数、增加和减少霍尔电压 设定值等功能。

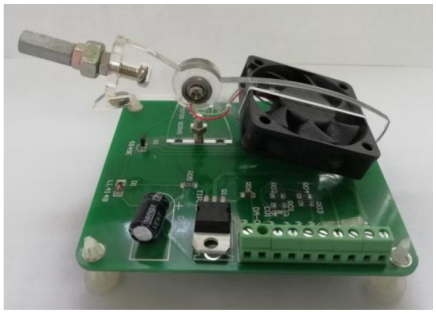
| 端子号 | 端子名称 | 功能 |
|---|---|---|
| 1 | +12V | +12V电源 |
| 2 | GND | 数字地 |
| 3 | +5V | +5V电源 |
| 4 | GND | 数字地 |
| 5 | AIN3 | 空接 |
| 6 | AIN2 | 空接 |
| 7 | AIN1 | 空接 |
| 8 | AIN0 | 霍尔传感器电路输出信号检测端子 |
| 9 | DA_OUT | 模拟量控制信号端子 |
| 10 | AGND | 模拟地 |

## 1.2 传感器系统介绍

SS49E线性霍尔传感器具有体积小，用途广泛等特点。SS49E可由永磁体或电磁铁进行操作，电源电压 控制线性输出，可根据磁场强度的不同做成线性变化。SS49E内部集成了低噪声输出电路，省去了外部滤波 器的使用。器件包含了薄膜电阻，增加了温度的稳定性和精度。SS49E的工作电压为4.5V~6V。

## 1.3 直升机模拟系统介绍

直升机垂直升降模拟对象系统原理图如图2-2所示，实物如图2-3所示，接口说明如表2-1所示。



# 2 系统模块介绍

## 2.1 ADC采样系统

ADC采样系统因为是对整个项目生命最重要的部分，这里我们想要通过对于Stm32F103与C8051F020单片机的差距，来展现更高级系统对于控制的作用。

## 2.2 STM32

ADC实现思路：

我们首先需要了解STM32F103系列所拥有的三个高精度ADC，

| 符号 | 参数 | 条件 | 最小值 | 典型值 | 最大值 | 单位 |
|---|---|---|---|---|---|---|
| $V_{DDA}$ | 供电电压 | | 2.4 | | 3.6 | V |
| $V_{REF+}$ | 正参考电压 | | 2.4 | | $V_{DDA}$ | V |
| $I_{VREF}$ | 在$V_{REF}$输入脚上的电压 | | | 160(1) | 220(1) | μA |
| $f_{ADC}$ | ADC时钟频率 | | 0.6 | | 14 | MHz |
| $f_S$(2) | 采样速率 | | 0.05 | | 1 | MHz |
| $f_{TRIG}$(2) | 外部触发频率 | $f_{ADC}$ = 14MHz | | | 823 | kHz |
| | | | | | 17 | $1/f_{ADC}$ |
| $V_{AIN}$ | 转换电压范围(3) | | 0($V_{SSA}$或$V_{REF-}$连接到地) | | $V_{REF+}$ | V |
| $R_{AIN}$(2) | 外部输入阻抗 | | 参见公式1和表59 | | | kΩ |
| $R_{ADC}$(2) | 采样开关电阻 | | | | 1 | kΩ |
| $C_{ADC}$(2) | 内部采样和保持电容 | | | | 12 | pF |
| $t_{CAL}$(2) | 校准时间 | $f_{ADC}$ = 14MHz | 5.9 | | | μs |
| | | | 83 | | | $1/f_{ADC}$ |
| $t_{lat}$(2) | 注入触发转换时延 | $f_{ADC}$ = 14MHz | | | 0.214 | μs |
| | | | | | 3(4) | $1/f_{ADC}$ |
| $t_{latr}$(2) | 常规触发转换时延 | $f_{ADC}$ = 14MHz | | | 0.143 | μs |
| | | | | | 2(4) | $1/f_{ADC}$ |

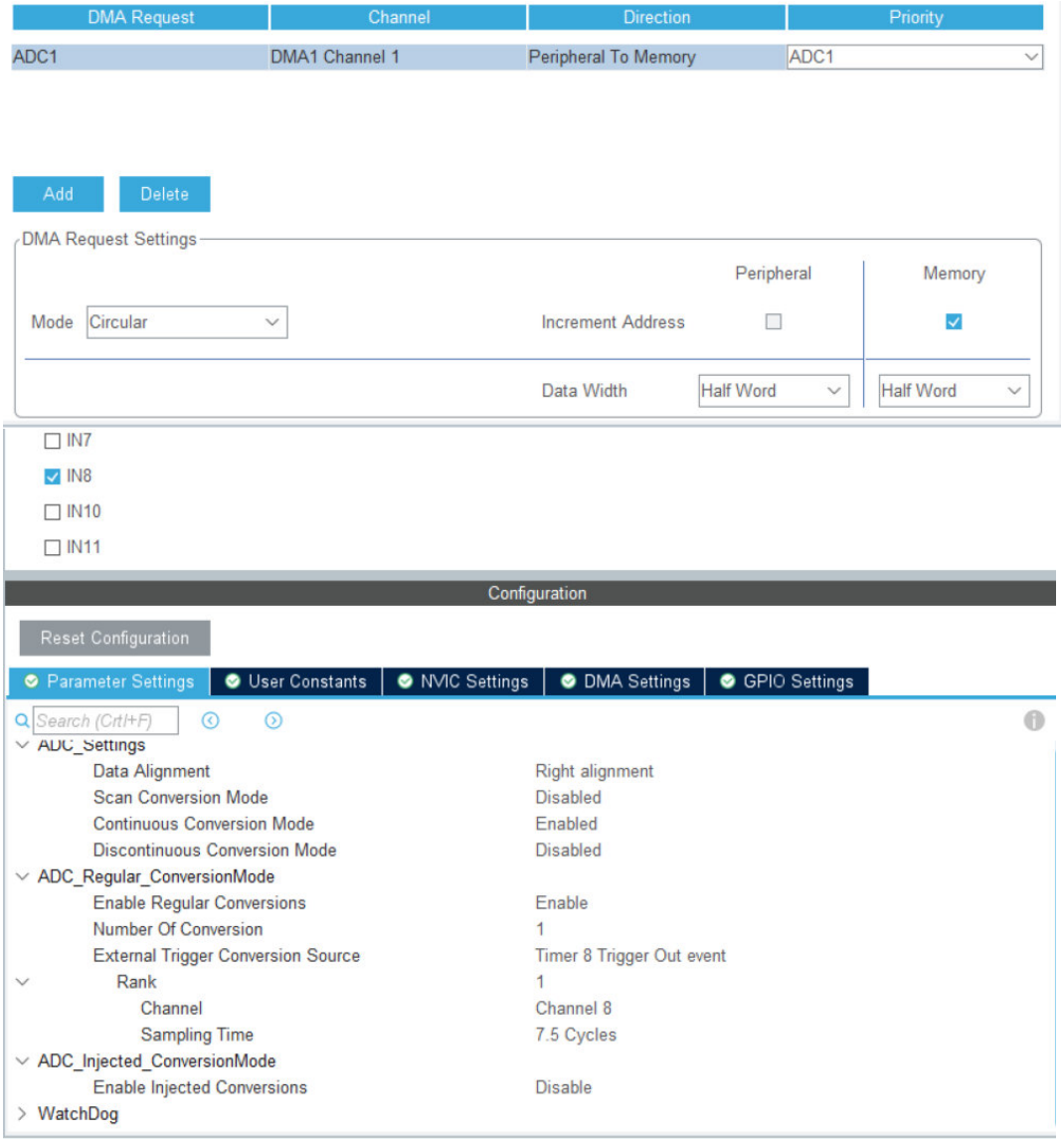
显然我们可以看到这里$f_{adc}$工作频率是14MHZ，而系统时钟是72MHZ，显然需要把分频因子设置为6。因为我们后续还希望CPU在处理ADC时执行更多操作，显然不应该使用轮询，由于我们希望由定时器中断触发ADC的采样，然后DMA源源不断的将代码从ADC的寄存器移动到指定数组。

---

### 2.2.1 CUBEMX配置

#### 2.2.1.1 ADC1和ADC3的配置

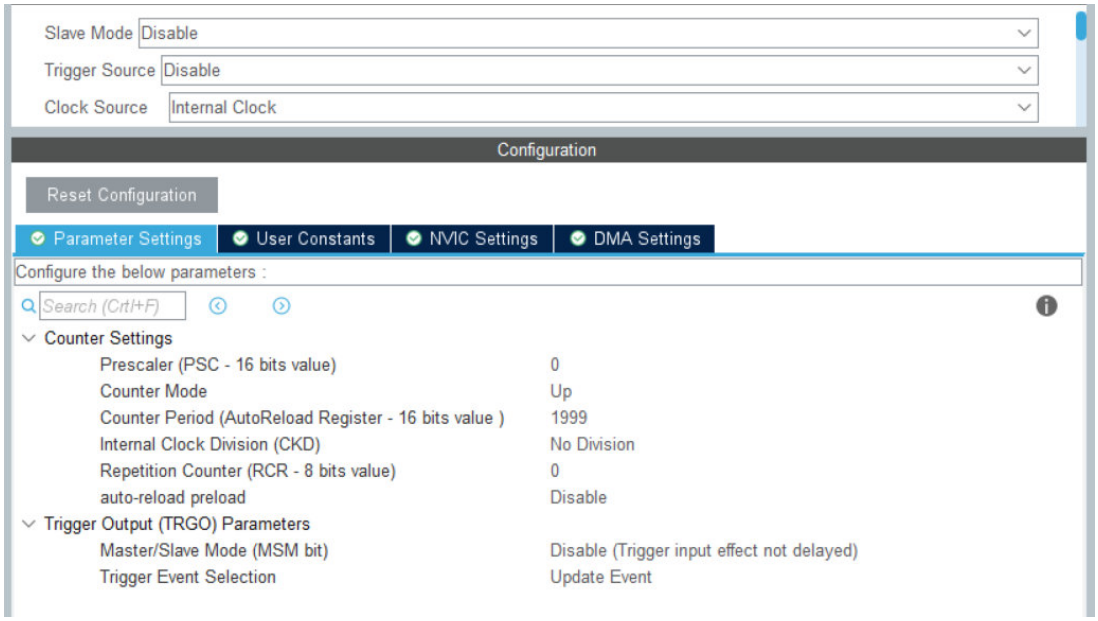
##### 2.2.1.1.1 图形初始化




以ADC3的配置举例，（ADC3的通道8，对应PF10）

- 在ADC配置中需要修改 External Trigger Convision Source为定时器8触发，Sampling Time修改为7.5Cycles，Continuous Conversion Mode（连续转换）改为ENABLE。
- 在DMA配置中，Data Width(数据宽)改为Half World. *Mode 改为循环，这至关重要，这使得DMA传送不会停止，笔者一开始默认了NORMAL，DEBug发现每次只传送一次，就结束了* ，方向改为Peripheral to Memory

#### 2.2.1.1.2 TIMEer定时器界面



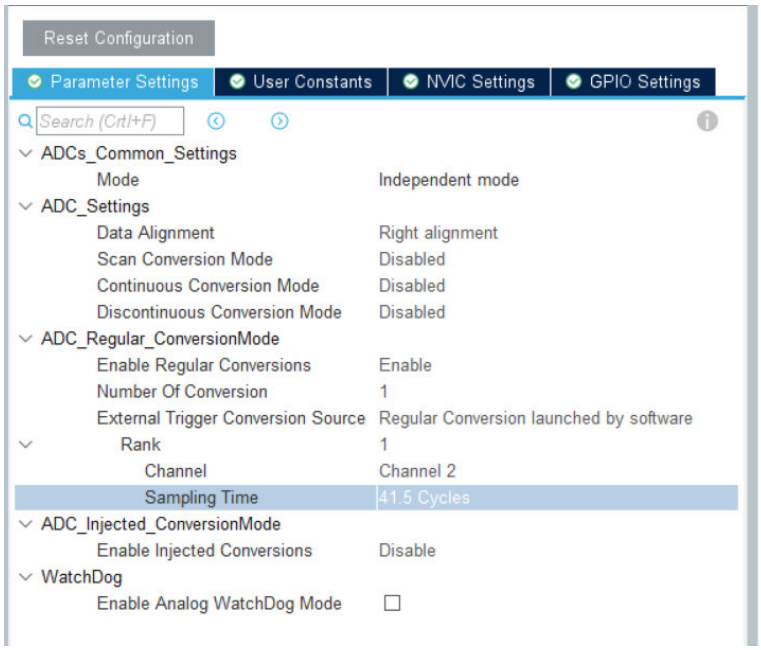

要想实现每0.x秒的精准采样，这一点是必不可少的，因为我们如果只使用DMA技术，ADC将不断以高频率采样，笔者发现这种速度太快且不可控，尤其在实际生产中，我们还需要协调3个ADC的关系，如果3个ADC都在同一个时刻采样，显然是可以很好的解决我们的问题。

- 分频系数改为1999，Trigger Event Selection 改为自动更新，每次中断结束后，自己重新开始计数。

---

#### 2.2.1.2 ADC2的配置

##### 2.2.1.2.1 图形化初始化

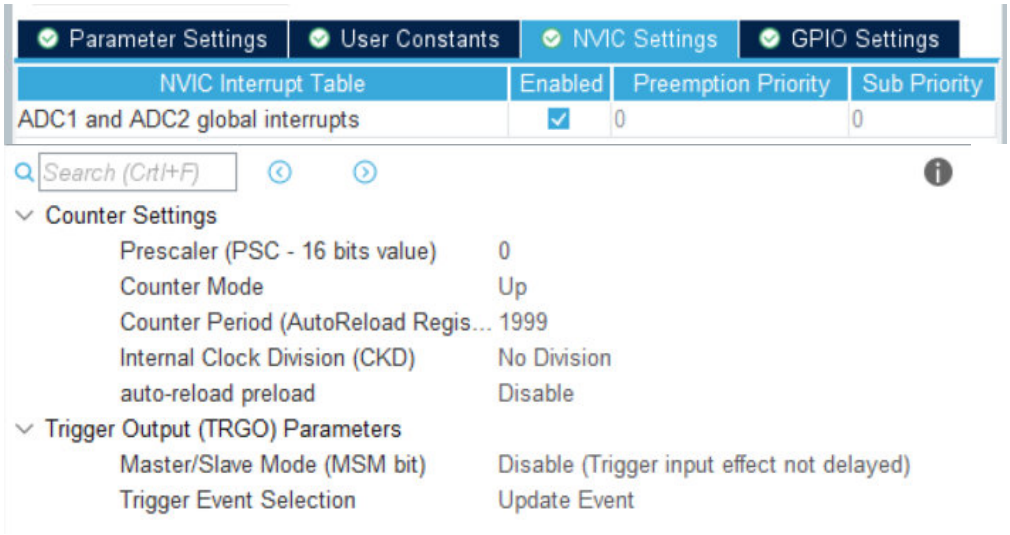



因为ADC2没有DMA系统，所以我们需要使用单独的Timer3来产生中断，这里我们用一个小技巧。我们如果直接设置ADC2由TIMER3触发，发生很奇怪的现象，始终没有中断写入。不如直接使用软件中断触发，然后写两个中断回调函数来判断，这么做本质原因就是调用HAL库之后，程序员部分失去了底层的控制。

- Continuous Conversion Mode :改为Disabled

- Samping Time (因为没有DMA，我们丧失了快速反应能力，不如直接增加Samping Time) :41.5 Cycles

#### 2.2.1.2.2 Timer定时器界面




- TIMER3的初始设置和TIMER8一样

#### 2.2.1.2.3 中断初始化


- 中断初始化只需要将值（0，0）赋值给（Preemption Priority, Sub Pority）
- 确实没有必要给ADC太高的优先级，因为我们ADC采样是不断进行的，我们应该注意串口的优先级，串口再发送数据时，发送WIFI串口优先级>>发送Debug优先级>>ADC. *个人觉得打断串口的中断是一种疯狂的行为，曾经这个错误把我折腾了一下午，甚至无法靠调试得到结果*

---

### 2.2.2 代码源码（不感兴趣的可以跳过）

#### 2.2.2.1 ADC.c 与ADC.h

*代码主体由CUBEMX组成，我们需要理解后，对指定部分进行更改，我们在后面进行了详细注释，尤其是修改部分*

（ADC1 ADC3）（ ADC2）分别有两种配置方式

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file    adc.c
  * @brief   This file provides code for the configuration
  *          of the ADC instances.
  ******************************************************************************
  * @attention
```

```
  *
  * Copyright (c) 2022 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "adc.h"

/* USER CODE BEGIN 0 */
__IO uint16_t ADC_ConvertedValue1;
__IO uint16_t ADC_ConvertedValue2;
__IO uint16_t ADC_ConvertedValue3;
#include "tim.h"
/* USER CODE END 0 */

ADC_HandleTypeDef hadc1;
ADC_HandleTypeDef hadc2;
ADC_HandleTypeDef hadc3;
DMA_HandleTypeDef hdma_adc1;
DMA_HandleTypeDef hdma_adc3;

/* ADC1 init function */
void MX_ADC1_Init(void)
{

  /* USER CODE BEGIN ADC1_Init 0 */
  __HAL_RCC_DMA1_CLK_ENABLE();
  /* USER CODE END ADC1_Init 0 */

  ADC_ChannelConfTypeDef sConfig = {0};

  /* USER CODE BEGIN ADC1_Init 1 */

  /* USER CODE END ADC1_Init 1 */
  /** Common config
  */
  hadc1.Instance = ADC1;
  hadc1.Init.ScanConvMode = ADC_SCAN_DISABLE;
  hadc1.Init.ContinuousConvMode = ENABLE;
```

```
  hadc1.Init.DiscontinuousConvMode = DISABLE;
  hadc1.Init.ExternalTrigConv = ADC_EXTERNALTRIGCONV_T8_TRGO;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.NbrOfConversion = 1;
  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    Error_Handler();
  }
  /** Enable or disable the remapping of ADC1_ETRGREG:
  * ADC1 External Event regular conversion is connected to TIM8 TRGO
  */
  __HAL_AFIO_REMAP_ADC1_ETRGREG_ENABLE();
  /** Configure Regular Channel
  */
  sConfig.Channel = ADC_CHANNEL_4;
  sConfig.Rank = ADC_REGULAR_RANK_1;
  sConfig.SamplingTime = ADC_SAMPLETIME_7CYCLES_5;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN ADC1_Init 2 */
  //HAL_ADC_Start_DMA(&hadc1, (uint32_t*)&ADC_ConvertedValue1, 1);
  /* USER CODE END ADC1_Init 2 */
}
/* ADC2 init function */
void MX_ADC2_Init(void)
{

  /* USER CODE BEGIN ADC2_Init 0 */
  /* USER CODE END ADC2_Init 0 */

  ADC_ChannelConfTypeDef sConfig = {0};

  /* USER CODE BEGIN ADC2_Init 1 */

  /* USER CODE END ADC2_Init 1 */
  /** Common config
  */
  hadc2.Instance = ADC2;
  hadc2.Init.ScanConvMode = ADC_SCAN_DISABLE;
  hadc2.Init.ContinuousConvMode = DISABLE;
  hadc2.Init.DiscontinuousConvMode = DISABLE;
  hadc2.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc2.Init.DataAlign = ADC_DATAALIGN_RIGHT;
```

```
  hadc2.Init.NbrOfConversion = 1;
  if (HAL_ADC_Init(&hadc2) != HAL_OK)
  {
    Error_Handler();
  }
  /** Configure Regular Channel
  */
  sConfig.Channel = ADC_CHANNEL_2;
  sConfig.Rank = ADC_REGULAR_RANK_1;
  sConfig.SamplingTime = ADC_SAMPLETIME_41CYCLES_5;
  if (HAL_ADC_ConfigChannel(&hadc2, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN ADC2_Init 2 */

  /* USER CODE END ADC2_Init 2 */

}
/* ADC3 init function */
void MX_ADC3_Init(void)
{

  /* USER CODE BEGIN ADC3_Init 0 */
  __HAL_RCC_DMA2_CLK_ENABLE();
  /* USER CODE END ADC3_Init 0 */

  ADC_ChannelConfTypeDef sConfig = {0};

  /* USER CODE BEGIN ADC3_Init 1 */

  /* USER CODE END ADC3_Init 1 */
  /** Common config
  */
  hadc3.Instance = ADC3;
  hadc3.Init.ScanConvMode = ADC_SCAN_DISABLE;
  hadc3.Init.ContinuousConvMode = ENABLE;
  hadc3.Init.DiscontinuousConvMode = DISABLE;
  hadc3.Init.ExternalTrigConv = ADC_EXTERNALTRIGCONV_T8_TRGO;
  hadc3.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc3.Init.NbrOfConversion = 1;
  if (HAL_ADC_Init(&hadc3) != HAL_OK)
  {
    Error_Handler();
  }
```

```
  /** Configure Regular Channel
  */
  sConfig.Channel = ADC_CHANNEL_8;
  sConfig.Rank = ADC_REGULAR_RANK_1;
  sConfig.SamplingTime = ADC_SAMPLETIME_7CYCLES_5;
  if (HAL_ADC_ConfigChannel(&hadc3, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN ADC3_Init 2 */
  //HAL_ADC_Start_DMA(&hadc3, (uint32_t*)&ADC_ConvertedValue3, 1);
  /* USER CODE END ADC3_Init 2 */

}

void HAL_ADC_MspInit(ADC_HandleTypeDef* adcHandle)
{

  GPIO_InitTypeDef GPIO_InitStruct = {0};
  if(adcHandle->Instance==ADC1)
  {
  /* USER CODE BEGIN ADC1_MspInit 0 */

  /* USER CODE END ADC1_MspInit 0 */
    /* ADC1 clock enable */
    __HAL_RCC_ADC1_CLK_ENABLE();

    __HAL_RCC_GPIOA_CLK_ENABLE();
    /**ADC1 GPIO Configuration
    PA4     ------> ADC1_IN4
    */
    GPIO_InitStruct.Pin = GPIO_PIN_4;
    GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

    /* ADC1 DMA Init */
    /* ADC1 Init */
    hdma_adc1.Instance = DMA1_Channel1;
    hdma_adc1.Init.Direction = DMA_PERIPH_TO_MEMORY;
    hdma_adc1.Init.PeriphInc = DMA_PINC_DISABLE;
    hdma_adc1.Init.MemInc = DMA_MINC_ENABLE;
    hdma_adc1.Init.PeriphDataAlignment = DMA_PDATAALIGN_HALFWORD;
    hdma_adc1.Init.MemDataAlignment = DMA_MDATAALIGN_HALFWORD;
    hdma_adc1.Init.Mode = DMA_CIRCULAR;
    hdma_adc1.Init.Priority = DMA_PRIORITY_MEDIUM;
```

```
  if (HAL_DMA_Init(&hdma_adc1) != HAL_OK)
  {
    Error_Handler();
  }

  __HAL_LINKDMA(adcHandle,DMA_Handle,hdma_adc1);

  /* ADC1 interrupt Init */
  HAL_NVIC_SetPriority(ADC1_2_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(ADC1_2_IRQn);
/* USER CODE BEGIN ADC1_MspInit 1 */

/* USER CODE END ADC1_MspInit 1 */
}
else if(adcHandle->Instance==ADC2)
{
/* USER CODE BEGIN ADC2_MspInit 0 */

/* USER CODE END ADC2_MspInit 0 */
  /* ADC2 clock enable */
  __HAL_RCC_ADC2_CLK_ENABLE();

  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  /**ADC2 GPIO Configuration
  PC2     ------> ADC2_IN12
  PA2     ------> ADC2_IN2
  */
  GPIO_InitStruct.Pin = GPIO_PIN_2;
  GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

  GPIO_InitStruct.Pin = GPIO_PIN_2;
  GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /* ADC2 interrupt Init */
  HAL_NVIC_SetPriority(ADC1_2_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(ADC1_2_IRQn);
/* USER CODE BEGIN ADC2_MspInit 1 */

/* USER CODE END ADC2_MspInit 1 */
}
else if(adcHandle->Instance==ADC3)
{
```

```
  /* USER CODE BEGIN ADC3_MspInit 0 */

  /* USER CODE END ADC3_MspInit 0 */
    /* ADC3 clock enable */
    __HAL_RCC_ADC3_CLK_ENABLE();

    __HAL_RCC_GPIOF_CLK_ENABLE();
    /**ADC3 GPIO Configuration
    PF10     ------> ADC3_IN8
    */
    GPIO_InitStruct.Pin = GPIO_PIN_10;
    GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
    HAL_GPIO_Init(GPIOF, &GPIO_InitStruct);

    /* ADC3 DMA Init */
    /* ADC3 Init */
    hdma_adc3.Instance = DMA2_Channel5;
    hdma_adc3.Init.Direction = DMA_PERIPH_TO_MEMORY;
    hdma_adc3.Init.PeriphInc = DMA_PINC_DISABLE;
    hdma_adc3.Init.MemInc = DMA_MINC_ENABLE;
    hdma_adc3.Init.PeriphDataAlignment = DMA_PDATAALIGN_HALFWORD;
    hdma_adc3.Init.MemDataAlignment = DMA_MDATAALIGN_HALFWORD;
    hdma_adc3.Init.Mode = DMA_CIRCULAR;
    hdma_adc3.Init.Priority = DMA_PRIORITY_MEDIUM;
    if (HAL_DMA_Init(&hdma_adc3) != HAL_OK)
    {
      Error_Handler();
    }

    __HAL_LINKDMA(adcHandle,DMA_Handle,hdma_adc3);

  /* USER CODE BEGIN ADC3_MspInit 1 */

  /* USER CODE END ADC3_MspInit 1 */
  }
}

void HAL_ADC_MspDeInit(ADC_HandleTypeDef* adcHandle)
{

  if(adcHandle->Instance==ADC1)
  {
  /* USER CODE BEGIN ADC1_MspDeInit 0 */

  /* USER CODE END ADC1_MspDeInit 0 */
```

```
  /* Peripheral clock disable */
  __HAL_RCC_ADC1_CLK_DISABLE();

  /**ADC1 GPIO Configuration
  PA4     ------> ADC1_IN4
  */
  HAL_GPIO_DeInit(GPIOA, GPIO_PIN_4);

  /* ADC1 DMA DeInit */
  HAL_DMA_DeInit(adcHandle->DMA_Handle);

  /* ADC1 interrupt Deinit */
/* USER CODE BEGIN ADC1:ADC1_2_IRQn disable */
  /**
  * Uncomment the line below to disable the "ADC1_2_IRQn" interrupt
  * Be aware, disabling shared interrupt may affect other IPs
  */
  /* HAL_NVIC_DisableIRQ(ADC1_2_IRQn); */
/* USER CODE END ADC1:ADC1_2_IRQn disable */

/* USER CODE BEGIN ADC1_MspDeInit 1 */

/* USER CODE END ADC1_MspDeInit 1 */
}
else if(adcHandle->Instance==ADC2)
{
/* USER CODE BEGIN ADC2_MspDeInit 0 */

/* USER CODE END ADC2_MspDeInit 0 */
  /* Peripheral clock disable */
  __HAL_RCC_ADC2_CLK_DISABLE();

  /**ADC2 GPIO Configuration
  PC2     ------> ADC2_IN12
  PA2     ------> ADC2_IN2
  */
  HAL_GPIO_DeInit(GPIOC, GPIO_PIN_2);

  HAL_GPIO_DeInit(GPIOA, GPIO_PIN_2);

  /* ADC2 interrupt Deinit */
/* USER CODE BEGIN ADC2:ADC1_2_IRQn disable */
  /**
  * Uncomment the line below to disable the "ADC1_2_IRQn" interrupt
  * Be aware, disabling shared interrupt may affect other IPs
```

```
    */
    /* HAL_NVIC_DisableIRQ(ADC1_2_IRQn); */
  /* USER CODE END ADC2:ADC1_2_IRQn disable */

  /* USER CODE BEGIN ADC2_MspDeInit 1 */

  /* USER CODE END ADC2_MspDeInit 1 */
  }
  else if(adcHandle->Instance==ADC3)
  {
  /* USER CODE BEGIN ADC3_MspDeInit 0 */

  /* USER CODE END ADC3_MspDeInit 0 */
    /* Peripheral clock disable */
    __HAL_RCC_ADC3_CLK_DISABLE();

    /**ADC3 GPIO Configuration
    PF10     ------> ADC3_IN8
    */
    HAL_GPIO_DeInit(GPIOF, GPIO_PIN_10);

    /* ADC3 DMA DeInit */
    HAL_DMA_DeInit(adcHandle->DMA_Handle);
  /* USER CODE BEGIN ADC3_MspDeInit 1 */

  /* USER CODE END ADC3_MspDeInit 1 */
  }
}

/* USER CODE BEGIN 1 */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)    //定时器中断回调
{
    HAL_ADC_Start_IT(&hadc2);
}

void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* AdcHandle){
    HAL_ADC_Stop_IT(&hadc2);
    HAL_TIM_Base_Stop_IT(&htim3);
    ADC_ConvertedValue2=HAL_ADC_GetValue(&hadc2);
    HAL_TIM_Base_Start_IT(&htim3);
}
```

#### 2.2.2.2 ADC3中断函数的说明

这里注意，一定要在中断里面关闭时钟，然后处理完再打开，这是一个好习惯，因为谁都不知道会在里面处理多久，如果不关闭，下一次中断来临，可能会造成程序异常。*血的教训*

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)     //定时器中断回调
{
    HAL_ADC_Start_IT(&hadc2); //定时器中断里面开启ADC中断转换，1ms开启一次采集
}

void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* AdcHandle){
    HAL_ADC_Stop_IT(&hadc2);    //关闭adc2
    HAL_TIM_Base_Stop_IT(&htim3); //关闭时钟Timer2
    ADC_ConvertedValue2=HAL_ADC_GetValue(&hadc2);//传递值
    HAL_TIM_Base_Start_IT(&htim3);  //打开时钟Timer2
}
```

#### 2.2.2.3 ADC1,2初始化时的说明

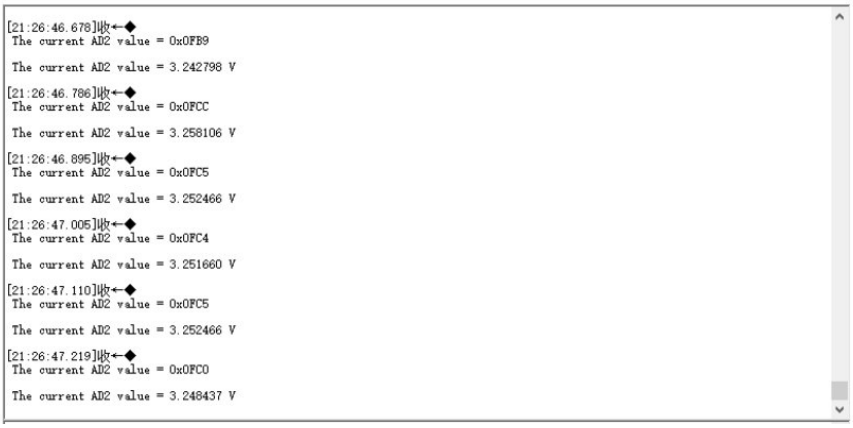
为了使用DMA传输，我们需要在第一句使能DMA时钟，但是注意我们不用HAL库生成的DMA_init,其存在一定的时间逻辑混乱。

```
void MX_ADC1_Init(void)
{

  /* USER CODE BEGIN ADC1_Init 0 */
  __HAL_RCC_DMA1_CLK_ENABLE();
  /* USER CODE END ADC1_Init 0 */

  ADC_ChannelConfTypeDef sConfig = {0};
  ...
  }


  void MX_ADC3_Init(void)
{

  /* USER CODE BEGIN ADC3_Init 0 */
  __HAL_RCC_DMA2_CLK_ENABLE();
  /* USER CODE END ADC3_Init 0 */

  ADC_ChannelConfTypeDef sConfig = {0};

  /* USER CODE BEGIN ADC3_Init 1 */
  ...}
```

### 2.2.3 测试结果

#### 2.2.3.1 ADC1


#### 2.2.3.2 ADC2




#### 2.2.3.3 ADC3


### 2.2.4 硬件说明（ADC1,2,3通用）

#### 2.2.4.1 ADC硬件连接

*由于ADC采样只能采集0-3.3v之间的电压，所以需要硬件设计中考虑对传感器的分压，建议采用2个贴片电阻将5V分压至3.3.v,请务必考虑ADC的输入电阻$R_{AIN}$*

公式1：最大$R_{AIN}$公式

$$R_{AIN} < \frac{T_S}{f_{ADC} \times C_{ADC} \times \ln(2^{N+2})} - R_{ADC}$$

上述公式(公式1)用于决定最大的外部阻抗，使得误差可以小于1/4 LSB。其中N=12(表示12位分辨率)。

表59 $f_{ADC}$=14MHz(1)时的最大$R_{AIN}$

| $T_S$(周期) | $t_S$(μs) | 最大$R_{AIN}$(kΩ) |
|---|---|---|
| 1.5 | 0.11 | 1.2 |
| 7.5 | 0.54 | 10 |
| 13.5 | 0.96 | 19 |
| 28.5 | 2.04 | 41 |
| 41.5 | 2.96 | 60 |
| 55.5 | 3.96 | 80 |
| 71.5 | 5.11 | 104 |
| 239.5 | 17.1 | 350 |

1. 由设计保证，不在生产中测试。

*本嵌入式，均采用$T_s$=55.5*

## 2.3 51ADC

### 2.3.1 DA 转换

DAC 的输出方式有很多，如果选择了通过定时器 2 溢出来更新 DAC 的输出，由于 C8051F02 单片机的DAC0 为 12 位精度的数模转换器，当用 16 位来存储时，就产生了不同的数据对齐方式。设置 DAC0 时， 首先在初始化部分设置 DAC0 的更新方式和对齐模式，随后为 DAC0 选择参考电压(一般为内部参考电压)， 随后将数据寄存器清 0 就完成了初始化。DAC0 的输出控制，在定时器 2 中断溢出时，把需要输出的电压所 对应的值送给高、低数据寄存器即可。

### 2.3.2 AD 转换

ADC0 是 12 位精度的模数转换器，和 DAC0 一样，也需要一个定时器来控制时序。不同的是 ADC0 不是靠定时器溢出的中断来开始转换，而是在定时器溢出的时候不触发中断，并且在转换结束的时候触发自 己的中断来让微处理器读取数据。定时器 3 的初始化与其它定时器的初始化略有不同，由于不需要定时器 溢出中断，在初始化时，应关闭定时器中断并立刻开始计时。

(1)ADC0 初始化

ADC0 的初始化设置相比于 DAC0 复杂了一些。除了设置采用定时器 3 溢出启动、数据对齐格式和参 考电压之外，还需要设置 ADC0 的采样通道、ADC0 的 SAR 时钟频率、ADC0 的增益，并且使能 ADC0 转 换结束的中断请求。这些参数可以根据需要自行修改。从 ADC0 中读取转换的结果时，在中断处理程序中，首先需要将中断触发为手动清零，随后利用缓冲 的思想，将 ADC0 的各个通道数值读入 buffer 中，不断变换通道，就可以从不同通道读取数值，这样可以 避免时序对于程序的影响。ADC0 的采样频率如果大于程序采用的频率，则 buffer 会不断被覆盖，这样可以 保证用到的数据都是最新的采样值，虽然浪费了 ADC0 的能力，但是不会导致程序出现问题。然而，当 ADC0 的采样频率低于系统利用采样值的频率时，就需要系统进行等待，以防止同一采样值的重复读取，可以采 用下述方法:如果 buffer 经过了读取，则为 buffer 赋一个不可能采到的值(如 0xffff)，判断只要读取的值 出现异常，主程序就进行等待，直到获得正常的数值。

(2)ADC0 数据处理

ADC0 获取的数据处理时，从 ADC0 获取的 12 位数据需要转化为 10 进制数据。同时，由于物理环境的影响，ADC0 读取的数值并不会那么准确，甚至连线性都不能保证。计算时需要对 ADC0 的采样值进行 修正，修正的办法就是假设 ADC0 是线性的，ADC0 的输入端分别接入的是 1V、5V 电压，获取 ADC0对应的数值并做归一化处理，ADC0 就可以较为准确的读取输入电压了。例如在模拟直升机垂直升降控制系统中，需要根据外部放大电路，对 ADC0 的输入电压进行适当缩小，并转化为 12 位二进制数送给 ADC0 的数据寄存器。

#### 2.3.3 代码

```
void Do(void) {
    if ((timer1_value & 0x0007) == 0x0001) {
        if (channel == 1)    {
            // 从 ADC0 AIN1 取得 10 位 16 进制数 vadc
            vadc = ADC_ValueReturn(channel);
            // 将 vadc 转化为 10 进制数进行计算
            vadc_dec = (unsigned long int)vadc * (unsigned long int)vref / 4096;
            PID_contrl(&tmp_pid,vadc_dec);
            vdac_dec=vadc_dec+tmp_pid.result;
            // 将 10 进制数 vdac_dec 转化为 16 进制数
            vdac = (unsigned long int)vdac_dec * 4096 / (unsigned long int)vref;
            // 从 ADC0 输出 10 位 16 进制数 vdac
            DAC0_Output(vdac);
          //调节目标霍尔电压值
          if(KEY_FLAG==1){
            tmp_pid.setpoint-=100; //用于控制目标电压
            KEY_FLAG=0;}
          else{
            if(KEY_FLAG==2){
            tmp_pid.setpoint+=100; //用于控制目标电压
            KEY_FLAG=0;}
          }
      }
    }
  }
```

## 2.4 控制算法

### 2.4.1 自适应控制算法

#### 2.4.1.1 自适应控制算法的原理

自适应过程是一个不断逼近目标的过程。它所遵循的途径以数学模型表示，称为自适应算法。通常采用基于梯度的算法，其中最小均方误差算法尤为常用。自适应算法可以用硬件(处理电路)或软件（程序控制）两种办法实现。前者依据算法的数学模型设计电路，后者则将算法的数学模型编制成程序并用计算机实现。算法有很多种，它的选择很重要，它决定处理系统的性能质量和可行性。

#### 2.4.1.2 自适应控制算法的代码

```
void FuzzyPID();  //初始化PID参数
float FuzzyPIDcontroller(float e_max, float e_min, float ec_max, float ec_min, float
kp_max, float kp_min, float error, float error_c, float ki_max, float ki_min,float kd_max,
float kd_min,float error_pre, float error_ppre);  //模糊PID控制实现函数
float Quantization(float maximum, float minimum, float x);  //误差 error 和误差变化 error_c
映射到论域中的函数
```

```
void Get_grad_membership(float error, float error_c); ////计算输入e与de/dt隶属度
void GetSumGrad();// 获取输出增量 △kp、△ki、△kd 的总隶属度
void GetOUT();  // 计算输出增量 △kp、△ki、△kd 对应论域值
float Inverse_quantization(float maximum, float minimum, float qvalues);  //去模糊化


const int  num_area = 8; //划分区域个数
float e_membership_values[7] = {-3,-2,-1,0,1,2,3}; //输入e的隶属值
float ec_membership_values[7] = { -3,-2,-1,0,1,2,3 };//输入de/dt的隶属值
float kp_menbership_values[7] = { -3,-2,-1,0,1,2,3 };//输出增量kp的隶属值
float ki_menbership_values[7] = { -3,-2,-1,0,1,2,3 }; //输出增量ki的隶属值
float kd_menbership_values[7] = { -3,-2,-1,0,1,2,3 };  //输出增量kd的隶属值

float kp;                       //PID参数kp
float ki;                       //PID参数ki
float kd;                       //PID参数kd
float qdetail_kp;               //增量kp对应论域中的值
float qdetail_ki;               //增量ki对应论域中的值
float qdetail_kd;               //增量kd对应论域中的值
float detail_kp;                //输出增量kp
float detail_ki;                //输出增量ki
float detail_kd;                //输出增量kd
float qerror;                    //输入e对应论域中的值
float qerror_c;                  //输入de/dt对应论域中的值
float e_gradmembership[2];      //输入e的隶属度
float ec_gradmembership[2];     //输入de/dt的隶属度
int e_grad_index[2];            //输入e隶属度在规则表的索引
int ec_grad_index[2];           //输入de/dt隶属度在规则表的索引
float KpgradSums[7] = { 0,0,0,0,0,0,0 };   //输出增量kp总的隶属度
float KigradSums[7] = { 0,0,0,0,0,0,0 };   //输出增量ki总的隶属度
float KdgradSums[7] = { 0,0,0,0,0,0,0 };   //输出增量kd总的隶属度

float e_max = 150;      //误差最大值
float e_min = -150;    //误差最小值
float ec_max = 300;     //误差变化最大值
float ec_min = -300;    //误差变化最小值
float kp_max = 50;       //比例系数 kp 上限值
float kp_min = -50;    //比例系数 kp 下限值
float ki_max = 0.1;     //积分系数 ki 上限值
float ki_min = -0.1;    //积分系数 ki 下限值
float kd_max = 0.01;    //微分系数 kd 上限值
float kd_min = -0.01;   //微分系数 kd 下限值
float error;        //误差值
float error_c;      //误差变化值
float error_pre = 0;    //上一次误差值
```

```
float error_ppre = 0;   //上上次误差值


int  Kp_rule_list[7][7] = { {PB,PB,PM,PM,PS,ZO,ZO},        //kp规则表
             {PB,PB,PM,PS,PS,ZO,NS},
             {PM,PM,PM,PS,ZO,NS,NS},
             {PM,PM,PS,ZO,NS,NM,NM},
             {PS,PS,ZO,NS,NS,NM,NM},
             {PS,ZO,NS,NM,NM,NM,NB},
             {ZO,ZO,NM,NM,NM,NB,NB} };

int  Ki_rule_list[7][7] = { {NB,NB,NM,NM,NS,ZO,ZO},     //ki规则表
            {NB,NB,NM,NS,NS,ZO,ZO},
            {NB,NM,NS,NS,ZO,PS,PS},
            {NM,NM,NS,ZO,PS,PM,PM},
            {NM,NS,ZO,PS,PS,PM,PB},
            {ZO,ZO,PS,PS,PM,PB,PB},
            {ZO,ZO,PS,PM,PM,PB,PB} };

  int  Kd_rule_list[7][7] = { {PS,NS,NB,NB,NB,NM,PS},     //kd规则表
             {PS,NS,NB,NM,NM,NS,ZO},
             {ZO,NS,NM,NM,NS,NS,ZO},
             {ZO,NS,NS,NS,NS,NS,ZO},
             {ZO,ZO,ZO,ZO,ZO,ZO,ZO},
                          {PB,NS,PS,PS,PS,PS,PB},
                          {PB,PM,PM,PM,PS,PS,PB} };

void FuzzyPID()  //参数初始化
{
  kp = 0;
  ki = 0;
  kd = 0;
  qdetail_kp = 0;
  qdetail_ki = 0;
  qdetail_kd = 0;
}

//模糊PID控制实现函数
float FuzzyPIDcontroller(float e_max, float e_min, float ec_max, float ec_min, float
kp_max, float kp_min, float error, float error_c,float ki_max,float ki_min,float
kd_max,float kd_min,float error_pre,float error_ppre)
{
  float output;
  qerror = Quantization(e_max, e_min, error);    //将 误差 error 映射到论域中
  qerror_c = Quantization(ec_max, ec_min, error_c);   //将误差变化 error_c 映射到论域中
```

```
  Get_grad_membership(qerror, qerror_c);  //计算误差 error 和误差变化 error_c 的隶属度
  GetSumGrad(); //计算输出增量 △kp、△ki、△kd 的总隶属度
  GetOUT();   // 计算输出增量 △kp、△ki、△kd 对应论域值
  detail_kp = Inverse_quantization(kp_max, kp_min, qdetail_kp);    //去模糊化得到增量 △kp
  detail_ki = Inverse_quantization(ki_max, ki_min, qdetail_ki);    //去模糊化得到增量 △ki
  detail_kd = Inverse_quantization(kd_max, kd_min, qdetail_kd);    //去模糊化得到增量 △kd
  qdetail_kd = 0;
  qdetail_ki = 0;
  qdetail_kp = 0;
  kp = kp + detail_kp;    //得到最终的 kp 值
  ki = ki + detail_ki;    //得到最终的 ki 值
  kd = kd + detail_kd;    //得到最终的 kd 值
  if (kp < 0){
    kp = 0;}
  if (ki < 0){
    ki = 0;}
  if (kd < 0){
    kd = 0;}
  detail_kp = 0;
  detail_ki = 0;
  detail_kd = 0;
  output = kp*(error - error_pre) + ki * error + kd * (error - 2 * error_pre + error_ppre);
  //计算最终的输出
  return output;
}


///区间映射函数
float Quantization(float maximum,float minimum,float x)
{
  float qvalues= 6.0 *(x-minimum)/(maximum - minimum)-3;
  return qvalues;
}

//输入e与de/dt隶属度计算函数
void Get_grad_membership(float error,float error_c)
{
  int i;
  if (error > e_membership_values[0] && error < e_membership_values[6])
  {
    for ( i = 0; i < num_area - 2; i++)
    {
      if (error >= e_membership_values[i] && error <= e_membership_values[i + 1])
      {
```

```
        e_gradmembership[0] = -(error - e_membership_values[i + 1]) /
(e_membership_values[i + 1] - e_membership_values[i]);
        e_gradmembership[1] = 1+(error - e_membership_values[i + 1]) /
(e_membership_values[i + 1] - e_membership_values[i]);
        e_grad_index[0] = i;
        e_grad_index[1] = i + 1;
        break;
      }
    }
  }
  else
  {
    if (error <= e_membership_values[0])
    {
      e_gradmembership[0] = 1;
      e_gradmembership[1] = 0;
      e_grad_index[0] = 0;
      e_grad_index[1] = -1;
    }
    else if (error >= e_membership_values[6])
    {
      e_gradmembership[0] = 1;
      e_gradmembership[1] = 0;
      e_grad_index[0] = 6;
      e_grad_index[1] = -1;
    }
  }

  if (error_c > ec_membership_values[0] && error_c < ec_membership_values[6])
  {
    for ( i = 0; i < num_area - 2; i++)
    {
      if (error_c >= ec_membership_values[i] && error_c <= ec_membership_values[i + 1])
      {
        ec_gradmembership[0] = -(error_c - ec_membership_values[i + 1]) /
(ec_membership_values[i + 1] - ec_membership_values[i]);
        ec_gradmembership[1] = 1 + (error_c - ec_membership_values[i + 1]) /
(ec_membership_values[i + 1] - ec_membership_values[i]);
        ec_grad_index[0] = i;
        ec_grad_index[1] = i + 1;
        break;
      }
    }
  }
  else
```

```
  {
    if (error_c <= ec_membership_values[0])
    {
      ec_gradmembership[0] = 1;
      ec_gradmembership[1] = 0;
      ec_grad_index[0] = 0;
      ec_grad_index[1] = -1;
    }
    else if (error_c >= ec_membership_values[6])
    {
      ec_gradmembership[0] = 1;
      ec_gradmembership[1] = 0;
      ec_grad_index[0] = 6;
      ec_grad_index[1] = -1;
    }
  }

}

// 获取输出增量kp,ki,kd的总隶属度
void GetSumGrad()
{
  int i;
  int j;

    // 初始化 Kp、Ki、Kd 总的隶属度值为 0

  for ( i = 0; i <= num_area - 1; i++)
  {
    KpgradSums[i] = 0;
    KigradSums[i] = 0;
            KdgradSums[i] = 0;
  }

      for ( i = 0; i < 2; i++)
     {
            if (e_grad_index[i] == -1)
            {
                  continue;
          }
            for ( j = 0; j < 2; j++)
          {
                  if (ec_grad_index[j] != -1)
                  {
                      int indexKp = Kp_rule_list[e_grad_index[i]][ec_grad_index[j]] + 3;
```

```
                    int indexKi = Ki_rule_list[e_grad_index[i]][ec_grad_index[j]] + 3;
                    int indexKd = Kd_rule_list[e_grad_index[i]][ec_grad_index[j]] + 3;
                    KpgradSums[indexKp]= KpgradSums[indexKp] + (e_gradmembership[i] *
ec_gradmembership[j]);
                    KigradSums[indexKi] = KigradSums[indexKi] + (e_gradmembership[i] *
ec_gradmembership[j]);
                    KdgradSums[indexKd] = KdgradSums[indexKd] + (e_gradmembership[i] *
ec_gradmembership[j]);
                }
                else
                {
                    continue;
                }
            }
    }
}

// 计算输出增量kp,kd,ki对应论域值
void GetOUT()
{
  int i;
  for ( i = 0; i < num_area - 1; i++)
  {
    qdetail_kp += kp_menbership_values[i] * KpgradSums[i];
    qdetail_ki += ki_menbership_values[i] * KigradSums[i];
    qdetail_kd += kd_menbership_values[i] * KdgradSums[i];
  }
}

//反区间映射函数
float Inverse_quantization(float maximum, float minimum, float qvalues)
{
  float x = (maximum - minimum) *(qvalues + 3)/6 + minimum;
  return x;
}
```

#### 2.4.1.3 自适应控制算法的不足

在实际代码中，由于系统已经进行了优化，但是仍然存在只能在相当狭窄的死区内活动，也就是说由于算力的问题造成了会忽然无法实现控制。

#### 2.4.1.4 控制算法的其他说明

##### 2.4.1.4.1 自适应控制PID结构体数组

```
typedef struct{
  int setpoint; //设定值
  long sumerror;  //误差的总和
  float P;      //p
  float I;      //I
  float D;  //D
  int lasterror;//当前误差
  int preverror;//前一次误差
  int result;//本次PID结果
}PID;
```

##### 2.4.1.4.2 自适应控制PID 计算输出电压代码

```
void PID_contrl(PID* ptr,int nowpoint){
  int tmp_error;
  int tmp_common;
  tmp_error = ptr->setpoint - nowpoint;
  ptr->sumerror+=tmp_error;
  tmp_common=tmp_error-ptr->lasterror;
  ptr->result=FuzzyPIDcontroller(2000, -2000,500, -500, 50, -50, tmp_error, tmp_common,
0.1,-0.1,0.01, -0.01,ptr->lasterror, ptr->preverror);
  ptr->preverror=ptr->lasterror;
  ptr->lasterror = tmp_error;
}
```

### 2.4.2 增量式 PID 控制算法

*：该算法为验收时的代码，但是非发送的代码

#### 2.4.2.1 增量式PID算法的原理

和位置式PID控制不同，增量式PID控制将当前时刻的控制量和上一时刻的控制量做差，以差值为新的控制量，是一种递推式的算法。

#### 2.4.2.2 增量式PID算法的代码

$$u[n-1]=K_p\left\{e[n-1]+\frac{T}{T_i}\sum_{i=0}^{n-1}e[i]+\frac{T_d}{T}\{e[n-1]-e[n-2]\}\right\}$$

```
extern float Kp
extern float T
extern float Ti
extern float Td
void PID_contrl(PID* ptr,int nowpoint){
```

```
  int tmp_error;
  int tmp_common;
  tmp_error = ptr->setpoint - nowpoint;
  tmp_common=tmp_error-ptr->lasterror;
  ptr->result=Kp*(ptr->lasterror+(T/Ti)*ptr->sumerror+Td/T* (ptr->preverror-ptr-
>lasterror) ;
  ptr->sumerror+=tmp_error;
  ptr->preverror=ptr->lasterror;
  ptr->lasterror = tmp_error;
}
```

#### 2.4.2.3 增量式PID算法的不足

增量式PID对于不同的器件效果完全不同，需要人工自己调节PID参数，相当的麻烦。

## 2.5 LED显示模块

这部分只需要修改实验1例程即可得到结果。

```
void my_LedDispNum(unsigned int num1,unsigned int num2,unsigned int num3)
{
  unsigned char temp1[4];
  unsigned char temp2[4];
  unsigned char temp3[4];

  temp1[0] = num1%10;
  temp1[1] = num1%100/10;
  temp1[2] = num1%1000/100;
  temp1[3] = num1/1000;

  temp2[0] = num2%10;
  temp2[1] = num2%100/10;
  temp2[2] = num2%1000/100;
  temp2[3] = num2/1000;

  temp3[0] = num3%10;
  temp3[1] = num3%100/10;
  temp3[2] = num3%1000/100;
  temp3[3] = num3/1000;

  select(4);display(temp1[0]); Delay1(500); P7 = 0xff;
  select(3);display(temp1[1]); Delay1(500); P7 = 0xff;
  select(2);display(temp1[2]); Delay1(500); P7 = 0xff;
  select(1);display(temp1[3]); P7 = P7 & ~0x80;if(temp1[3] == 0) P7 = 0xff; Delay1(500); P7
= 0xff;

  select(8);display(temp2[0]); Delay1(500); P7 = 0xff;
```

```
  select(7);display(temp2[1]); Delay1(500); P7 = 0xff;
  select(6);display(temp2[2]); Delay1(500); P7 = 0xff;
  select(5);display(temp2[3]); P7 = P7 & ~0x80;Delay1(500); P7 = 0xff; // Delay(500);

  select(12);display(temp3[0]); Delay1(500); P7 = 0xff;
  select(11);display(temp3[1]); Delay1(500); P7 = 0xff;
  select(10);display(temp3[2]); Delay1(500); P7 = 0xff;
  select(9) ;display(temp3[3]); P7 = P7 & ~0x80;Delay1(500); P7 = 0xff; // Delay(500);
}
```

## 2.6 画点函数

这是我在网上找到的一段规范代码，就是用于将x依次画出

```
void LcdShowPoint(unsigned char x)
{
  unsigned char i;
  unsigned char col=x/16;
  unsigned char off=x%16;
  unsigned char row1=wave[x]/128;
  unsigned char datah1=0;
  unsigned char datal1=0;
  for(i=0;i<8;i++)
  {
    if(i<=off&&wave[col*16+i]/128==row1) datah1|=0x80>>i;
    if(i+8<=off&&wave[col*16+8+i]/128==row1) datal1|=0x80>>i;
  }
  WriteCommand(0x34);
  WriteCommand(0x80+31-row1);
  WriteCommand(0x80+col);
  WriteCommand(0x30);
  WriteData(datah1);
  WriteData(datal1);
  WriteCommand(0x32);
  WriteCommand(0x36);
}
```

# 3 程序顺序分析

由于逻辑完全是一条直线的方式，这里不再使用逻辑框图

## 3.1 初始化

需要初始化时钟和中断，这里额外提出，我们这段代码很多同学都不会。应当理解IF的意义。

```
int main(void) {
    int i=0,j=0;
    Device_Init();
    PID_init(&tmp_pid);
}


void INT_Init(void) {
  IT1 = 0;
  //enable INT1
  EX1 = 1;
  //enable all interrupt
  EA = 1;
  INT1_Type1;     // 设定 TCON 中断标志位 2, INT1 中断为边缘触发
  Enable_INT1;    // 设定 IE 标志位 2, 允许 INT1 中断请求
}
```

## 3.2 while循环控制

```
while (1)
{
  my_LedDispNum(vadc_dec,tmp_pid.setpoint,vdac_dec);//打印点
  PID_display();//PID屏幕初始化
  vdac=0;
  DAC0_Output(vdac);
  KEY_FLAG=0;
  LcdInit();

  if(KEY_FLAG==1)
      { //屏幕退出控制
        LcdClear();
        ImageShow(blank);
        KEY_FLAG=0;
       }
    while(KEY_FLAG!=3)
        {
      my_LedDispNum(vadc_dec,tmp_pid.setpoint,vdac_dec);//打印
```

```
Do();//实现控制，函数在上面adc部分已介绍
i+=1;
if(i%5==0)
{   //每隔五个点打印一次
  wave[j]=vadc_dec;
  LcdShowPoint(j);
  i=0;
  j+=1;
  if(j==128)//屏幕到达边界
  {
    ImageShow(blank);
    j=0;
  }
}
}}
```

# 4 实验总结

## 4.1 实验结果

使用增量式PID时可以很快得到很好的结果，但是自适应算法明显可以发现由于算力的缺失，无法实现我们想要的功能。

## 4.2 实验回顾

之前在大创控制小车就有使用PID算法的经验，不难发现在这次实验中，第一次接触到直升机模拟器，感觉十分有意思了，在其中也遇到按键移植后无法使用的问题，但是最终都被我逐个克服，感谢老师在这个过程中的帮助，这片说明文档已经和项目一起发布在gitee以及我自己搭建的个人网站上