ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

ĐẠI HỌC BÁCH KHOA

KHOA ĐIỆN – ĐIỆN TỬ

BỘ MÔN ĐIỀU KHIỂN TỰ ĐỘNG

BÁO CÁO TỔNG KẾT BÀI TẬP LỚN 2

MÔN HỌC ĐO LƯỜNG CÔNG NGHIỆP

ĐỀ TÀI: THIẾT KẾ MẠCH ĐO VÀ HIỂN THỊ VỊ TRÍ ĐỘNG CƠ SỬ DỤNG ENCODER

GVHD: Th.S Nguyễn Đức Hoàng

Danh sách thành viên:

1. Phạm Bình Nguyên – 1811113

2.Thái Quang Nguyên – 1813294

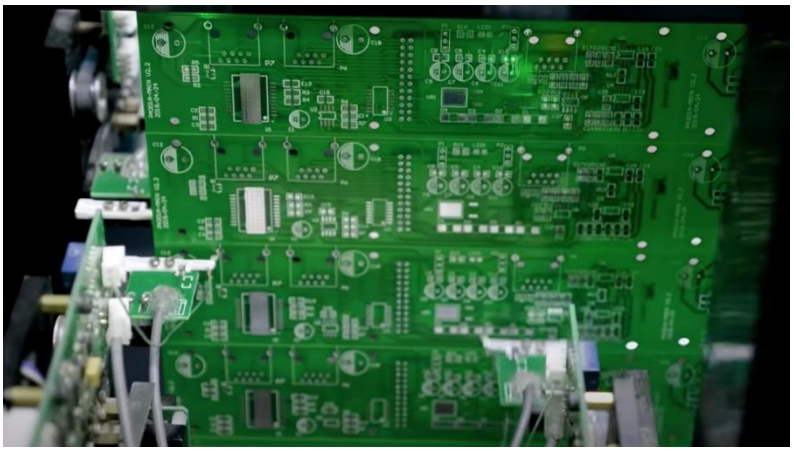
TPHCM, Ngày 26/12/2020

MỤC LỤC

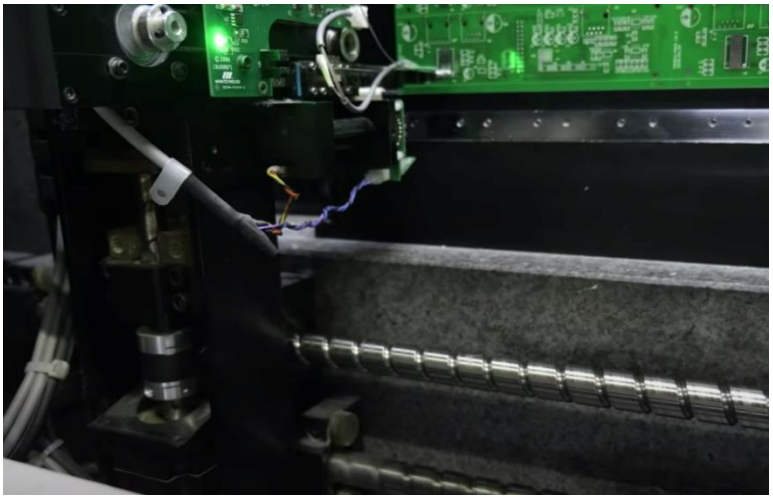
CHƯƠNG 1: TỔNG QUAN

1. Mở đầu

Động cơ điện được sử dụng phổ biến và rộng rãi trong hầu hết mọi mặt của cuộc sống, từ máy phát điện quốc gia, các cánh đồng năng lượng gió cho đến máy bơm nước của từng hộ gia đình, động cơ cho xe máy điện…. Tuy nhiên trong công nghiệp, đặc biệt là công nghiệp chế tạo, lại cần rất nhiều đến độ chính xác của tốc độ lẫn vị trí của động cơ. Lấy ví dụ như trong dây chuyền sản xuất PCB sẽ có công đoạn kiểm tra, đo thông mạch các mối nối trong một mạch đã được hoàn thiện. Điều này cần đến 2 cánh tay robot được điều khiển theo cả 3 trục xyz để chấm vào các pad đồng. Để làm được việc này đòi hỏi độ chính xác về vị trí của các động cơ là cực kì cao, việc điều khiển vòng hở động cơ rất khó có thể đạt được điều này. Vì vậy ta cần một cơ cấu cảm biến có thể ghi nhận lại được giá trị của vị trí động cơ để đưa vào luật điều khiển.



Hình 1.1. Quy trình đo thông mạch PCB



Hình 1.2. Kiểm tra lỗi mạch PCB

Tương tự, trong một dây chuyền sản xuất nhất định sẽ cần đến cơ cấu luân chuyển sản phẩm, thường thấy nhất là thông qua băng chuyền. Ở các nhà máy, thông thường các băng chuyền được nối dài liên tiếp nhau và chạy cùng tốc độ, đồng nghĩa với việc sử dụng một lúc nhiều đồng cơ để quay băng chuyền và di chuyển tải trên nó một cách đồng bộ. Ở đây đồng bộ ám chỉ về mặt tốc độ, các động cơ phải quay đồng thời và không được để việc động cơ này làm tải động cơ quay nhanh hơn xảy ra. Vì thế, các nhà máy đặt ra bài toán nhiều động cơ có thể hoạt động cùng vận tốc. Qua đó cần một cảm biến tốc độ trên mỗi động cơ để đưa vào luật điều khiển vòng kín để các động cơ đạt tốc độ giống nhau.

Hình 1.3. RV3100 Incremental Encoder của IFM

2. Cơ sở lý thuyết

Hai bài toán điều khiển vị trí và tốc độ động cơ được giải quyết bằng cách tích hợp cảm biến encoder vào mỗi động cơ. Chúng ta sẽ đi vào tìm hiểu các loại Encoder xoay (Rotary Encoder) và cách thức hoạt động của chúng, cụ thể là encoder tương đối và encoder tuyệt đối với công nghệ quang học (Optical). Ngoài ra còn có những loại Encoder tuyến tính (Linear) và công nghệ sử dụng hiệu ứng Hall… không được đề cập ở đây.

2.1. Encoder tương đối

Encoder tương đối (Incremental) thuộc lại Encoder quang (Optical Encoder), trên đĩa của Encoder có những vạch đen và trong suốt xen kẽ bằng nhau như hình dưới. Vạch trong suốt sẽ cho tia sáng của bộ phát xuyên qua và đến được bộ thu trong khi vạch đen thì chắn hoàn toàn tín hiệu quang học này. Do đó, bộ thu sẽ nhận xen kẽ tín hiệu, tạo thành một chuỗi xung 2 trạng thái 0 và 1.

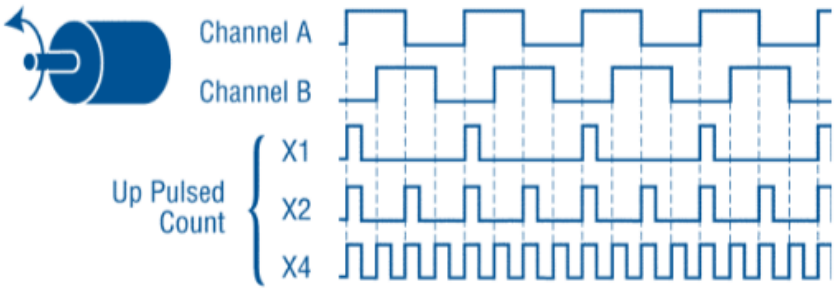
Hình 2.1. Incremental encoder’s disk

Đĩa Encoder tương đối thường có nhiều kênh khác nhau, điển hình sẽ có hai kênh A và B lệch nhau 90 độ điện. Dựa vào tín hiệu trả về từ 2 kênh này ta có thể xác định:

Vị trí tương đối góc quay: thông qua việc đếm delta số xung với điểm đặt.

Tốc độ động cơ: thông qua tần số (hoặc chu kỳ) của xung trả về.

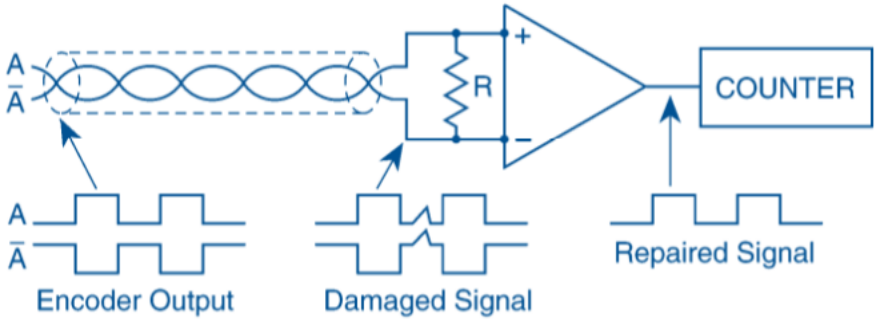
Chiều quay: thông qua sự có mặt trước hay sau của tín hiệu 2 kênh.



Hình 2.2. Dạng sóng ngõ ra ở kênh A và kênh B của Encoder

Ngoài ra, ở các encoder cần độ chính xác cao của công nghiệp, output của encoder sẽ có cả kênh A bù, B bù, Z và Z bù. Kênh Z là kênh phát ra một xung mỗi vòng quay của encoder. Các kênh bù chính là để so sánh với kênh đảo của nó để

trong trường hợp có nhiễu tác động như trong môi trường nhiều nhiễu từ ta có thể khử được bằng mạch so sánh Opamp.



Hình 2.3. Khử nhiễu dùng kênh đảo của Encoder

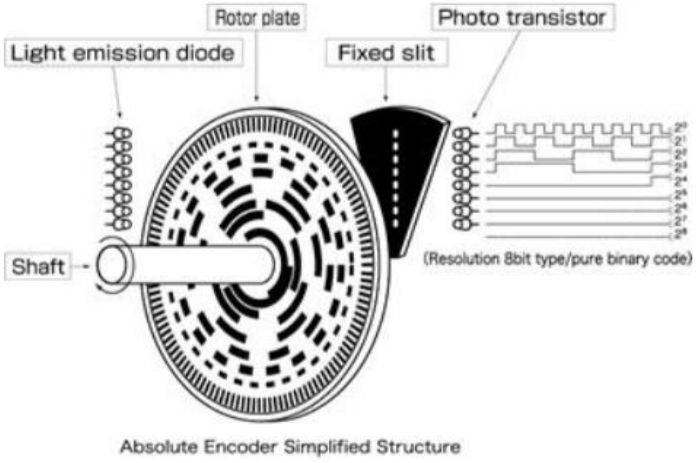
Qua đó, tín hiệu hồi tiếp từ Encoder tương đối có thể giúp ta thực hiện việc điều khiển chính xác tốc độ, vị trí tương đối và vòng quay. Bằng cách đọc các cạnh xung lên và/hoặc xung xuống của hai kênh A và B, ta có thể xác định được vị trí. Nếu kết hợp thêm với một bộ đếm Timer, ta hoàn toàn có thể tính ra được vận tốc RPM của động cơ. Ngoài ra, việc phát hiện cạnh lên và xuống của cả A và B giúp tăng độ phân giải của Encoder lên gấp 4 lần, kết hợp với tỉ số của hộp số, số xung trên vòng của Encoder có thể được gia tăng rất nhiều, theo đó độ chính xác sẽ càng cao.

2.2. Encoder tuyệt đối

Encoder tương đối tuy nhiên sẽ không thể lưu được vị trí chính xác của động cơ mà chỉ tính toán vị trí dựa trên sai khác số xung giữa điểm đặt và điểm hiện tại. Encoder tuyệt đối (Absolute Encoder) ra đời để giải quyết vấn đề này. Tương tự như Encoder tương đối, Encoder tuyệt đối sử dụng công nghệ quang, tuy nhiên tín hiệu trả về không phải là một dãy xung mà là một dãy các bit định danh vị trí hiện tại của trục rotor.

Hình 2.4. Absolute encoder’s disk

Dựa vào dãy bit này, ta map các giá trị với các góc xác định chứ không cần phải tính delta như Encoder tương đối.



Hình 2.5. Dạng song ngõ ra của Absolute encoder

Để tránh sai sót trong việc truyền tín hiệu, chuỗi bit trả về sử dụng mã gray, tức là tăng một nấc thì sẽ khác một bit.

Bảng 2.1. Bảng mã Gray 4 bit

Decimal	Gray Code	Binary
0	0000	0000
1	0001	0001
2	0011	0010
3	0010	0011
4	0110	0100
5	0111	0101
6	0101	0110
7	0100	0111
8	1100	1000
9	1101	1001
10	1111	1010
11	1110	1011
12	1010	1100
13	1011	1101
14	1001	1110
15	1000	1111

Tuy nhiên ở trong bài này ta sẽ thực hiện mạch và giao tiếp máy tính để calib encoder, do đó ta sẽ dùng Encoder loại tương đối.

CHƯƠNG 2: XÂY DỰNG PROJECT

1. Lập trình đọc cảm biến Encoder sử dụng STM32F407

1.1. Tổng quan về STM32F407

Dòng vi điều khiển ARM được phát triển bởi Advanced RISC Machines Ltd dựa kiến trúc RISC giúp tối gián hóa tập lệnh cũng như tối giản thời gian xử lý câu lệnh, gồm ba phân nhánh chính:

- Dòng A cho các ứng dụng cao cấp: Cortex – A78C, Cortex – A78AE, …
- Dòng R cho các ứng dụng thời gian thực: Cortex – R82, Cortex – R52, …
- Dòng M cho các ứng dụng vi điều khiển chi phí thấp: Cortex – M55, Cortex - M4, …

Kit STM32F407 sử dụng vi xử lý trung tâm Cortex – M4 được nâng cấp từ dòng Cortex – M3 với hiệu suất hệ thống được cải thiện đáng kể với mức năng lượng tiêu thụ khá thấp, được sử dụng trong nhiều ứng dụng điều khiển chi phí thấp với tần số xung Clock 84Mhz, 1MB bộ nhớ Flash và 192kB bộ nhớ SRAM được xây dựng trên kiến trúc Harvard (bộ nhớ chương trình và bộ nhớ dữ liệu được tách biệt với nhau) giúp tăng hiệu suất của chip trong xử lý đa nhiệm. Tuy nhiên trong một số ứng dụng với chương trình có dung lượng cao, ta phải kiểm soát bộ nhớ của chương trình để không đè lên bộ nhớ của dữ liệu của vi xử lý. Kit STm32F407 Discovery hỗ trợ 82 chân I/O với tần số Clock tối đa là 84Mhz, hộ trợ xử lý các phép toán dấu chấm động 32 bit, hộ trợ nhiều giao thức truyền thông với 3 cổng giao tiếp I2C hỗ trợ DMA, 8 cổng giao tiếp USART với 6 cổng USART hỗ trợ DMA, 6 cổng giao tiếp SPI với 3 cổng SPI hỗ trợ DMA, 2 cổng giao tiếp CAN 2.0B cùng nhiều chức năng giao tiếp

mở rộng như SDIO, USB 2.0, Ethernet, giao tiếp camera với 8-14 bit. Ngoài ra Kit STM32F407 còn hỗ trợ nhiều chức năng khác như 3 bộ ADC 12 bits với 16 kênh 2.4 MSPS, 2 bộ DAC 12 bit 7.2 MSPS, 12 Timer 16 bits, 2 Timer 32 bits hỗ trợ đọc Encoder, 2 Watchdog timers, hỗ trợ 82 channels interrupt. Trong project này, nhóm sử dụng kit STM32F407 Discovery với vi xử lý trung tâm là Cortex – M4 và lập trình bằng STM32 Standard Periphral Libraries, sử dụng USART 3 với hỗ trợ DMA 1, Stream 1, Channel 4 để giao tiếp với máy tính và Timer 5 với chức năng Encoder Interface để đọc cảm biến Encoder.

1.2. Đọc cảm biến Encoder sử dụng STM32F407

1.2.1. Khai báo các Timer cần thiết

Ở project này, ta sẽ sử dụng 3 Timer cho 3 công việc hoàn toàn khác nhau, đầu tiên ta khởi tạo biến TIM_BaseStruct với kiểu TIM_TimeBaseInitTypeDef.

```
TIM_TimeBaseInitTypeDef        TIM_BaseStruct;
```

Sử dụng Timer 6 16-bit với chức năng cơ bản, ta có thể thiết lập được hàm Delay. Khởi tạo Timer 6 với tốc độ tràn đúng bằng 1ms:

```
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM6, ENABLE);
TIM_BaseStruct.TIM_Prescaler                = 83;
TIM_BaseStruct.TIM_Period                   = 999; //delay 1ms
TIM_BaseStruct.TIM_CounterMode              = TIM_CounterMode_Up;
TIM_BaseStruct.TIM_ClockDivision            = 0;
TIM_BaseStruct.TIM_RepetitionCounter        = 0;
TIM_TimeBaseInit(TIM6, &TIM_BaseStruct);
TIM_UpdateDisableConfig(TIM6, DISABLE);
TIM_ARRPreloadConfig(TIM6, ENABLE);
```

Dựa trên cờ tràn của Timer 6, ta có thể code hàm delay_ms() với đối số vào là số ms cần delay:

```
void delay_ms(uint32_t milliSeconds)
{
    while (milliSeconds--)
    {
        TIM_SetCounter(TIM6, 0);
        TIM_Cmd(TIM6, ENABLE);
        while (TIM_GetFlagStatus(TIM6, TIM_FLAG_Update) != SET);
        TIM_Cmd(TIM6, DISABLE);
        TIM_ClearFlag(TIM6, TIM_FLAG_Update);
}
}
```

Ở chức năng thứ hai, nhóm dùng Timer 3 để tạo thời gian lấy mẫu đọc InputCapture và xuất ra giá trị góc đo được. Đồng thời liên kết cờ tràn của Timer với một hàm ngắt để thực hiện chức năng đọc này. Trong đó, khai báo biến NVIC_InitStruct với kiểu NVIC_InitTypeDef rồi khai báo Timer và ngắt như sau:

```
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);
TIM_BaseStruct.TIM_Prescaler         = 8399;
TIM_BaseStruct.TIM_Period            = 999; //delay 100ms
TIM_BaseStruct.TIM_CounterMode       = TIM_CounterMode_Up;
TIM_BaseStruct.TIM_ClockDivision     = 0;
TIM_BaseStruct.TIM_RepetitionCounter = 0;
TIM_TimeBaseInit(TIM3, &TIM_BaseStruct);
TIM_UpdateDisableConfig(TIM3, DISABLE);
TIM_ARRPreloadConfig(TIM3, ENABLE);
TIM_ITConfig(TIM3,TIM_IT_Update,ENABLE);
TIM_Cmd(TIM3, ENABLE);
NVIC_InitStruct.NVIC_IRQChannel                    = TIM3_IRQn;
NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStruct.NVIC_IRQChannelSubPriority         = 1;
NVIC_InitStruct.NVIC_IRQChannelCmd                 = ENABLE;
```

Hàm ngắt của Timer 3 sẽ được đề cập sau ở phần InputCapture. Ngoài ra còn có Timer 5 được dùng với mode InputCapture để đọc các xung Encoder trả về cũng sẽ được để cập ở phần InputCapture.

1.2.2. USART hỗ trợ DMA

Ta sử dụng chuẩn truyền nối tiếp USART được hỗ trợ trên STM32F4 để làm chuẩn truyền thông giữa vi điều khiển và GUI của Qt Creator. Kết hợp với chức

năng DMA (Direct Memory Access), ta có thể truyền dữ liệu tốc độ cao từ ngoại vi tới bộ nhớ, cũng như từ bộ nhớ tới bộ nhớ. Dữ liệu có thể được di chuyển từ GUI tới vi điều khiển một cách nhanh chóng mà không cần tác vụ từ CPU, tiết kiệm tài nguyên CPU. Nhóm sử dụng USART3 kết hợp với DMA1.

```
void USART_DMA_Configuration(unsigned int BaudRate){
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART3, ENABLE);
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_DMA1, ENABLE);
    GPIO_PinAFConfig(GPIOD, GPIO_PinSource8, GPIO_AF_USART3);
    GPIO_PinAFConfig(GPIOD, GPIO_PinSource9, GPIO_AF_USART3);
    GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_UP;
    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF;
    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_8 | GPIO_Pin_9;
    GPIO_InitStruct.GPIO_Speed = GPIO_Speed_100MHz;
    GPIO_Init(GPIOD, &GPIO_InitStruct);
    USART_InitStruct.USART_BaudRate = BaudRate;
    USART_InitStruct.USART_WordLength = USART_WordLength_8b;
    USART_InitStruct.USART_StopBits = MAIN_STOPBITS;
    USART_InitStruct.USART_Parity = MAIN_PARITY;
    USART_InitStruct.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
    USART_InitStruct.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
    USART_Init(USART3, &USART_InitStruct);
    USART_Cmd(USART3, ENABLE);
```

```
/* Enable USART3 DMA */
    USART_DMACmd(USART3, USART_DMAReq_Tx, ENABLE);
    USART_DMACmd(USART3, USART_DMAReq_Rx, ENABLE);
    /* Configure DMA Initialization Structure */

    DMA_InitStructure.DMA_FIFOMode = DMA_FIFOMode_Disable ;
    DMA_InitStructure.DMA_FIFOThreshold = DMA_FIFOThreshold_HalfFull ;
    DMA_InitStructure.DMA_MemoryBurst = DMA_MemoryBurst_Single ;
    DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_Byte;
    DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
    DMA_InitStructure.DMA_Mode = DMA_Mode_Normal;

    DMA_InitStructure.DMA_PeripheralBaseAddr =(uint32_t) (&(USART3->DR)) ;
    DMA_InitStructure.DMA_PeripheralBurst = DMA_PeripheralBurst_Single;
    DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Byte;
    DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
    DMA_InitStructure.DMA_Priority = DMA_Priority_High;

    /* Configure TX DMA */
    DMA_InitStructure.DMA_BufferSize = BUFF_SIZE_TX;
    DMA_InitStructure.DMA_Channel = DMA_Channel_4 ;
    DMA_InitStructure.DMA_DIR = DMA_DIR_MemoryToPeripheral ;
    DMA_InitStructure.DMA_Memory0BaseAddr =(uint32_t)TXBuffer ;
    DMA_Init(DMA1_Stream3,&DMA_InitStructure);
    DMA_Cmd(DMA1_Stream3, ENABLE);
```

```
/* Configure RX DMA */
    DMA_InitStructure.DMA_BufferSize = BUFF_SIZE_RX;
    DMA_InitStructure.DMA_Channel = DMA_Channel_4 ;
    DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralToMemory ;
    DMA_InitStructure.DMA_Memory0BaseAddr =(uint32_t)&RXBuffer;
    DMA_Init(DMA1_Stream1,&DMA_InitStructure);
    DMA_Cmd(DMA1_Stream1, ENABLE);

    /* Enable DMA Interrupt to the highest priority */
    NVIC_InitStruct.NVIC_IRQChannel = DMA1_Stream1_IRQn;
    NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0;
    NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStruct);
    /* Transfer complete interrupt mask */
    DMA_ITConfig(DMA1_Stream1, DMA_IT_TC, ENABLE);
}
```

Ngoài hàm ngắt để nhận dữ liệu từ GUI, ta cần một hàm để gửi dữ liệu lên GUI theo chu kì

- SendUSART

```
void SendUSART(char* data){ //dia chi bat dau cua chuoi
    for (int i = 0; *(data+i)!= NULL; i++){
            USART_SendData(USART3, *(data+i));
            *(data+i) = NULL;
            while (USART_GetFlagStatus(USART3, USART_SR_TXE) == RESET);
    }
}
```

- Hàm gửi giá trị góc liên tục:

```
void sendAngle(){
    char checksum_Tx = 0;
    m_data.myfloat = mainMotor.angle;
    for (int i = 0; i < 4; i++){
            TXBuffer[6+i] = m_data.mybyte[3-i];
            checksum_Tx += m_data.mybyte[3-i];
    }
    TXBuffer[10] = checksum_Tx;
    DMA_ClearFlag(DMA1_Stream3, DMA_FLAG_TCIF3);
  DMA1_Stream3->NDTR = BUFF_SIZE_TX;
  DMA_Cmd(DMA1_Stream3, ENABLE);
}
```

- Hàm ngắt nhận dữ liệu từ GUI:

```
void DMA1_Stream1_IRQHandler(void){
	DMA_ClearITPendingBit(DMA1_Stream1, DMA_IT_TCIF1);
	char checksum_Rx = 0;
	if(RXBuffer[0]=='$' && RXBuffer[1]=='Z' && RXBuffer[2]=='E' && RXBuffer[3]=='R' && RXBuffer[4]=='O' && RXBuffer[5]==','){
		mainMotor.angle = 0;
		TIM_SetCounter(TIM5, 0);
	}
	else if(RXBuffer[0]=='$' && RXBuffer[1]=='S' && RXBuffer[2]=='P' && RXBuffer[3]=='A' && RXBuffer[4]=='N' && RXBuffer[5]==','){
		for(int k = 0; k < 4; k++){
		m_data.mybyte[3-k] = RXBuffer[6+k];
		checksum_Rx += RXBuffer[6+k];
		}
		if(checksum_Rx == RXBuffer[10])
			if(m_data.myfloat>=0 && m_data.myfloat<360){
				mainMotor.calib_angle_span = m_data.myfloat;
				calibPara.span = mainMotor.calib_angle_span/mainMotor.counter;
				mainMotor.counter_per_round = (int)(mainMotor.counter/(4*mainMotor.calib_angle_span)*360);
			}
	}
	DMA_Cmd(DMA1_Stream1, ENABLE);
}
```

- Chức năng InputCapture:

+ Khai báo và thiết lập các chế độ:

```
void Encoder_Init(void){
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM5, ENABLE);
     GPIO_InitStruct.GPIO_Pin        = GPIO_Pin_0| GPIO_Pin_1;
    GPIO_InitStruct.GPIO_OType      = GPIO_OType_PP;
    GPIO_InitStruct.GPIO_Mode       = GPIO_Mode_AF;
    GPIO_InitStruct.GPIO_PuPd       = GPIO_PuPd_NOPULL;
    GPIO_InitStruct.GPIO_Speed      = GPIO_Speed_100MHz;
    GPIO_Init(GPIOA,&GPIO_InitStruct);
    GPIO_PinAFConfig(GPIOA,GPIO_PinSource0,GPIO_AF_TIM5);
    GPIO_PinAFConfig(GPIOA,GPIO_PinSource1,GPIO_AF_TIM5);
    TIM_BaseStruct.TIM_Prescaler                = 0;
    TIM_BaseStruct.TIM_Period                   = 0xFFFF;//8 chu F thi tot hon
    TIM_BaseStruct.TIM_ClockDivision            = TIM_CKD_DIV1;
    TIM_TimeBaseInit(TIM5,&TIM_BaseStruct);
    TIM_ICInitStruct.TIM_Channel                = TIM_Channel_1 | TIM_Channel_2;
    TIM_ICInitStruct.TIM_ICFilter               = 10;
    TIM_ICInitStruct.TIM_ICPolarity             = TIM_ICPolarity_Rising;
    TIM_ICInitStruct.TIM_ICPrescaler            = TIM_ICPSC_DIV4;
    TIM_ICInitStruct.TIM_ICSelection            = TIM_ICSelection_DirectTI;
    TIM_ICInit(TIM5,&TIM_ICInitStruct);
```

```
TIM_EncoderInterfaceConfig(TIM5,TIM_EncoderMode_TI12,TIM_ICPolarity_Rising,TIM_ICPolarity_Rising);
 TIM_SetCounter(TIM5, 0);
 TIM_Cmd(TIM5,ENABLE);
 TIM_ClearFlag(TIM5,TIM_FLAG_Update);
}
```

+ Hàm đọc giá trị góc quay mỗi chu kì:

```
void TIM3_IRQHandler(void){
    if (TIM_GetITStatus(TIM3, TIM_IT_Update) != RESET){
            TIM_Cmd(TIM3, DISABLE);
            TIM_SetCounter(TIM3, 0);
            GPIO_ToggleBits(GPIOD, GPIO_Pin_12);
            while (TIM5->CNT > COUNTER_MAX){
                    mainMotor.counter = (TIM5->CNT)%(COUNTER_ROUND);
                    TIM_SetCounter(TIM5,mainMotor.counter);
            }
            if(TIM5->CNT <= COUNTER_MAX)
                    mainMotor.counter = TIM5->CNT;
            mainMotor.angle = (int)(mainMotor.counter*calibPara.span)%360;
            mainMotor.angle = (mainMotor.angle >= 0)? mainMotor.angle:(360+mainMotor.angle);
            TIM_Cmd(TIM3, ENABLE);
            TIM_ClearITPendingBit(TIM3, TIM_IT_Update);
    }
}
```

2. Giao thức truyền nhận

Để giao vi xử lý gửi dữ liệu lên cho máy tính xử lý thực hiện việc hiển thị giá trị cũng như vẽ đồ thị góc quay của Encoder, nhóm qui định Frame truyền từ vi xử lý gửi lên cho máy tính gồm 13 bytes được quy định như sau:

<$ANGL,> <4 bytes data> <1 byte Checksum> <'\r'> <'\n'>

Trong đó 6 bytes đầu quy định thông số được gửi lên cho máy tính là thông số gì, 4 bytes data là kiểu dữ liệu Union, sau khi được gửi lên máy tính sẽ được chuyển từ 4 bytes thành 1 biến float để hiển thị góc quay và vẽ đồ thị, 1 byte checksum để kiểm tra lỗi và 2 bytes quy định kết thúc frame truyền

Để máy tính gửi thông số calib cảm biến xuống cho vi xử lý, nhóm quy định frame truyền calib zero và calib span như sau:

<$ZERO,> <4 bytes data> <1 byte Checksum> <'\r'> <'\n'>

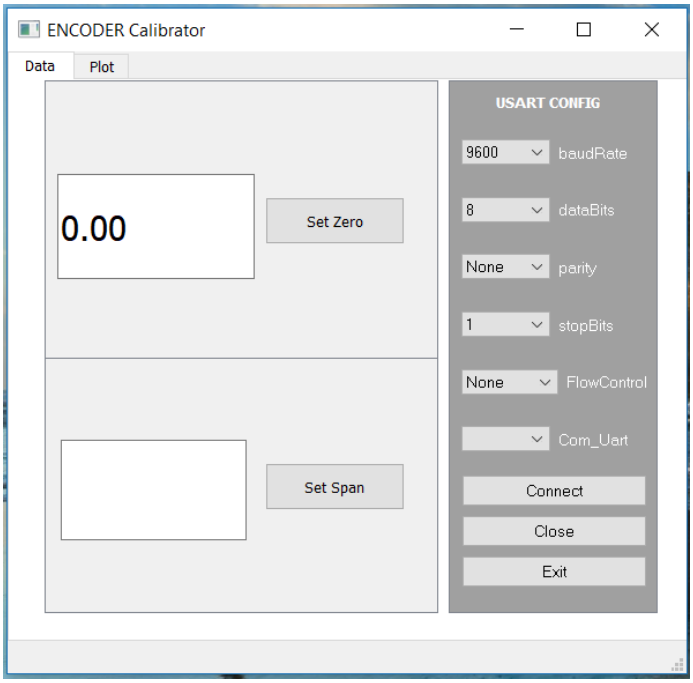
<$SPAN,> <4 bytes data> <1 byte Checksum> <'\r'> <'\n'>

Cũng tương tự như khi vi xử lý gửi dữ liệu lên cho máy tính, 6 bytes đầu quy định thông số gửi xuống vi xử lý là calib zero hay calib span, 4 bytes data kiểu Union, sau khi gửi xuống máy tính sẽ được chuyển từ 4 bytes thành 1 biến float để calib zero hoặc calib span của cảm biến.

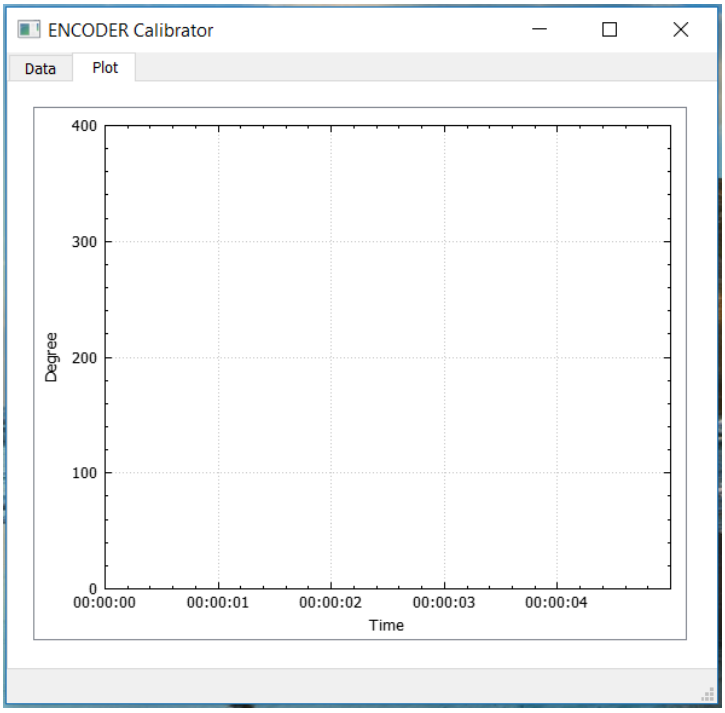
3. Lập trình GUI (Graphical User Interface) sử dụng QT

3.1. Tổng quan về QT

QT là phần mềm được phát triển bởi The QT Company là một ứng dụng đa nền tảng như Linus, Windows, MacOS, Android cùng bộ công cụ hỗ trợ lập trình giao diện đồ họa người dùng (GUI) được giới thiệu lần đầu vào ngày 20 tháng 5 năm 1995. Một số ứng dụng được viết trên nền tảng QT có thể kể đến như KDE, Opera, Google Earth, Skype,…Trong project này, nhóm sử dụng phần mềm QT 5.14.2 với trình biên dịch MinGW 64bit cùng các thư viện hỗ trợ như QtSerialPort để kết nối máy tính với vi xử lý STM32F407 và qcustomplot để vẽ đồ thị tín hiệu ngõ ra của Encoder.



Hình 3.1. Giao diện Calib cảm biến



Hình 3.2. Giao diện vẽ đồ thị ngõ ra cảm biến

3.2. Kết nối giữa máy tính với vi xử lý.

Ta lựa chọn giao thức truyền USART để giao tiếp giữa máy tính với vi xử lý STM32F407. Để truyền dữ liệu xuống vi xử lý đồng thời nhận dữ liệu từ vi xử lý ta kết nối cổng COM máy tính với USB UART CP2120 với chân TX của USB kết nối với chân RX của vi xử lý mà cụ thể ở đây nhóm sử dụng USART 3 với chức năng DMA và ngược lại với chân RX của USB.

Ở tab Data, nhóm thiết kế giao diện kết nối UART thông qua cổng COM máy tính bằng thư viện QtSerialPort với các chức năng bao gồm chọn tốc độ baud, số bit truyền, bit parity, giao thức kết nối và số stop bits bằng hàm fillPortAction()

Hàm fillPortAction()

```
void MainWindow::fillPortAction() {
    const auto infos = QSerialPortInfo::availablePorts();
    for(const QSerialPortInfo &info : infos){
        ui->Com_Uart->addItem(info.portName());
    }
    ui->baudRate->addItem(QStringLiteral("9600"),QSerialPort::Baud9600);
    ui->baudRate->addItem(QStringLiteral("19200"), QSerialPort::Baud19200);
    ui->baudRate->addItem(QStringLiteral("38400"), QSerialPort::Baud38400);
    ui->baudRate->addItem(QStringLiteral("57600"), QSerialPort::Baud57600);
    ui->baudRate->addItem(QStringLiteral("115200"), QSerialPort::Baud115200);
    ui->baudRate->addItem(tr("Custom"));

    ui->dataBits->addItem(QStringLiteral("5"), QSerialPort::Data5);
    ui->dataBits->addItem(QStringLiteral("6"), QSerialPort::Data6);
    ui->dataBits->addItem(QStringLiteral("7"), QSerialPort::Data7);
    ui->dataBits->addItem(QStringLiteral("8"), QSerialPort::Data8);
    ui->dataBits->setCurrentIndex(3);

    ui->parity->addItem(tr("None"), QSerialPort::NoParity);
    ui->parity->addItem(tr("Even"), QSerialPort::EvenParity);
    ui->parity->addItem(tr("Odd"), QSerialPort::OddParity);
    ui->parity->addItem(tr("Mark"), QSerialPort::MarkParity);
    ui->parity->addItem(tr("Space"), QSerialPort::SpaceParity);

    ui->stopBits->addItem(QStringLiteral("1"), QSerialPort::OneStop);
    ui->stopBits->addItem(QStringLiteral("2"), QSerialPort::TwoStop);

    ui->flowControl->addItem(tr("None"), QSerialPort::NoFlowControl);
    ui->flowControl->addItem(tr("RTS/CTS"), QSerialPort::HardwareControl);
    ui->flowControl->addItem(tr("XON/XOFF"), QSerialPort::SoftwareControl);

}
```

Cũng ở tab Data, sau khi lựa chọn được cấu hình cho USB UART, ta nhấn nút Connect. Nút Connect được liên kết với sự kiện nhấn clicked(), khi nút Connect được nhấn chương trình sẽ tiến hành sử dụng các thông số chúng ta vừa chọn để cấu hình cho USB USART.

Hàm on_btn_SetUart_clicked()

```
void MainWindow::on_btn_SetUart_clicked()
{
    SerialPort->setPortName(ui->Com_Uart->currentText());
    SerialPort->setBaudRate(ui->baudRate->currentText().toInt());
    SerialPort->setDataBits(static_cast<QSerialPort::DataBits>(ui->dataBits
    ->itemData(ui->dataBits->currentIndex()).toInt()));
    SerialPort->setParity(static_cast<QSerialPort::Parity>(ui->parity
    ->itemData(ui->parity->currentIndex()).toInt()));
    SerialPort->setStopBits(static_cast<QSerialPort::StopBits>(ui->stopBits
    ->itemData(ui->stopBits->currentIndex()).toInt()));
    SerialPort->setFlowControl(static_cast<QSerialPort::FlowControl>(ui
    ->flowControl->itemData(ui->flowControl->currentIndex()).toInt()));
    SerialPort->open(QIODevice::ReadWrite);
    connect(SerialPort, &QSerialPort::readyRead, this,&MainWindow::readData);
    Uart_Timer->start(20);

}
```

Sau khi hàm on_btn_SetUart_clicked() chạy, timer Uart_Timer của QT cũng bắt đầu chạy với thời gian tràn 20ms, mỗi lần Uart_Timer tràn, chương trình sẽ chạy hàm readData() để nhận dữ liệu USART gửi lên từ vi xử lý và hiển thị lên ô Angle đồng thời hàm RealTimeData cũng được chạy để vẽ đồ thị ở tab Plot.

Hàm readData()

```
void MainWindow::readData()
{
 //ui->m_console->moveCursor(QTextCursor::End);
 QByteArray byte_data = SerialPort->readAll();
 if(!byte_data.isEmpty()){
 if(byte_data.startsWith("$ANGL,")&& byte_data[12]=='\n'&&
byte_data[11]=='\r')
 {
 for (int i=0;i<4;i++)
     CheckSumRX+=byte_data[6+i];
     if(CheckSumRX==byte_data[10]){
         for(int k= 0;k<4;k++){
         m_data_RX.mybyte[3-k] = byte_data[6+k];}}
         my_latest_angle=m_data_RX.myfloat;
         QString my_little_data=QString::number(my_latest_angle);
         ui->RealAngle->setText(my_little_data);
         CheckSumRX=0;
         RealTimeData();}}}
```

Hàm RealTimeData()

```
void MainWindow::RealTimeData()
{
 static double temp_time;
 double key;
 static QTime time(QTime::currentTime());
 if(flag_PlotTimer){
 flag_PlotTimer = false;
 temp_time=time.elapsed()/1000.0;
 }
 Else
    key = time.elapsed()/1000.0;
 if (key<20)
     temp_time=0;
 else
     temp_time=key-20;
 ui->Data_plot->graph(0)->addData(key, m_data_RX.myfloat);
 ui->Data_plot->graph(0)->rescaleValueAxis();
 ui->Data_plot->xAxis->setRange(temp_time, key);
 ui->Data_plot->yAxis->setRange(0,400);
 ui->Data_plot->replot();
 // calculate frames per second:
 static double lastFpsKey;
 static int frameCount;
 ++frameCount;
 lastFpsKey = key;
 frameCount = 0;
}
```

Nếu như ta muốn ngắt kết nối USART với cổng COM, ta nhấn nút Close để ngắt đường truyền, nhấn nút Exit ta sẽ đồng thời ngắt kết nối cổng COM và tắt giao diện GUI.

3.3. Calib cảm biến bằng GUI

Để Calib cảm biến sử dụng GUI, nhóm lập trình hai chức năng là Set Zero và Set Span. Khi nút nhất Set Zero được nhấn, chương trình sẽ gửi xuống vi xử lý frame dữ liệu gồm 13 bytes để bi xử lý lựa chọn vị trí hiện tại là điểm Zero. Frame truyền bao gồm

<$ZERO,> <4 bytes data> <1 byte Checksum> <'\r'> <'\n'>

Hàm on_SetZero_clicked()

```
void MainWindow::on_SetZero_clicked()
{
    QByteArray txbuff;
    txbuff="$ZERO,";
    m_data.myfloat = 0;
    for(int k=0;k<4;k++)
    {
    txbuff[6+k]=0;
    }
    txbuff[10]=0;
    txbuff[11]= '\r';
    txbuff[12]= '\n';
    SerialPort->write(txbuff,13);
    ui->Set_My_Span->setPlainText("0");

   }
```

Ta quay Encoder đến một góc xác định trên trục quay sau đó nhất nút Set Span để chương trình gửi xuống vi xử lý vị trí hiện tại, từ đó vi xử lý có thể lấy giá trị góc vừa tính được chia cho số xung đếm được hiện tại trừ cho số xung đếm được tại thời điểm Set Zero và có được độ lợi của cảm biến. Frame truyền SetSpan

<$SPAN,> <4 bytes data> <1 byte Checksum> <'\r'> <'\n'>

Hàm on_SetSpan_clicked()

```
void MainWindow::on_SetSpan_clicked()
{
    my_little_set=ui->Set_My_Span->toPlainText().toFloat();
    char Checksum_TX=0;
    QByteArray txbuff;
    txbuff="$SPAN,";
    m_data.myfloat = my_little_set;
    for(int k=0;k<4;k++)
    {
    txbuff[6+k]=m_data.mybyte[3-k];
    Checksum_TX+=txbuff[6+k];
    }
    txbuff[10]=Checksum_TX;
    txbuff[11]= '\r';
    txbuff[12]= '\n';
    SerialPort->write(txbuff,13);

   }
```