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 HỌC PHẦN MỞ RỘNG

LÝ THUYẾT ĐIỀU KHIỂN NÂNG CAO

ĐỀ TÀI: XÂY DỰNG BỘ ĐIỀU KHIỂN TỰ CHỈNH STR CHO ĐỘNG CƠ DC

GVHD: PhD. Nguyễn Vĩnh Hảo

Danh sách thành viên:

1. Phạm Bình Nguyên – 1811113
2. Thái Quang Nguyên – 1813294

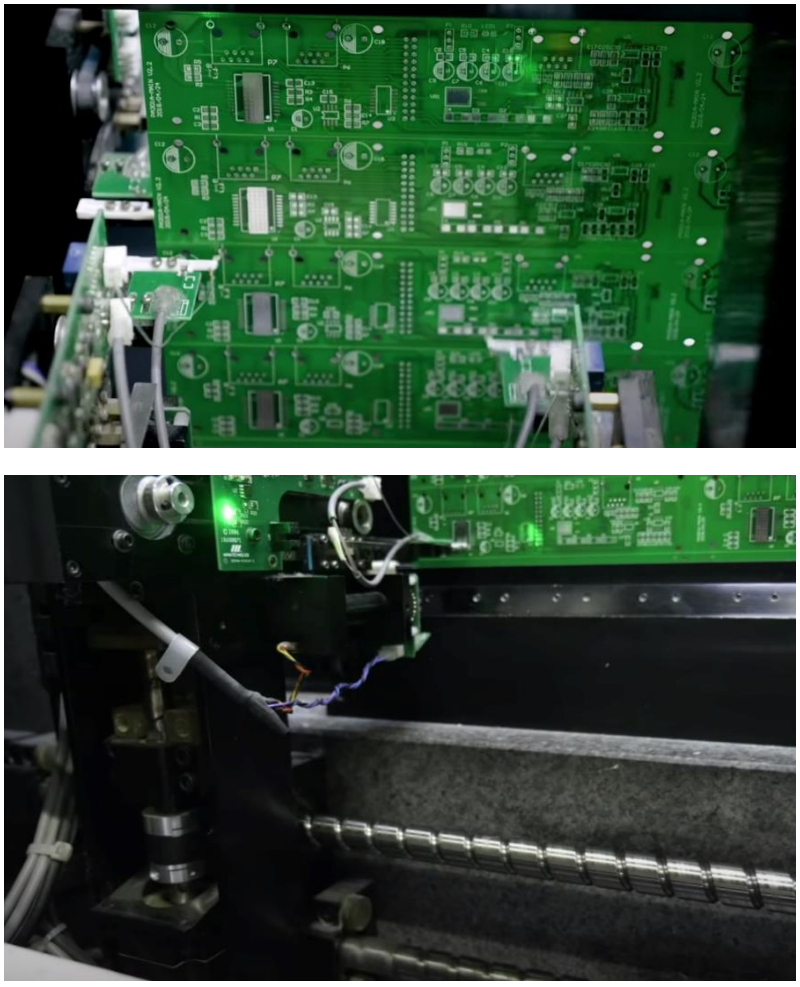
TPHCM, Tháng 1 Năm 2021

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. Hình ảnh đo thông mạch tự động trong quy trình sản xuất PCB

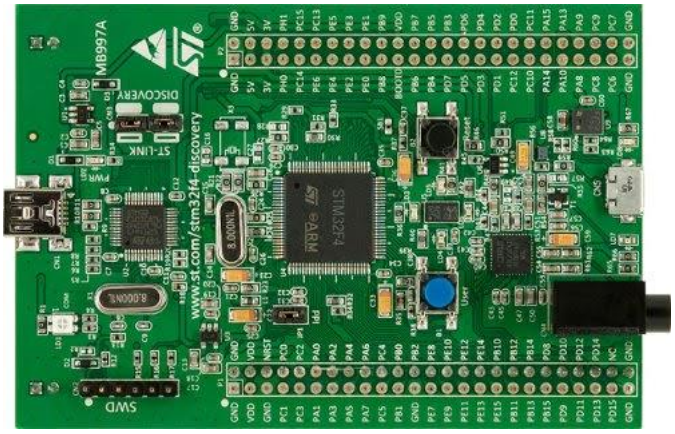
Tương tự, trong một dây chuyền sản xuất nhất định sẽ cần đến cơ cấu luân chuyển sản phẩm, thường thấy nhất là thông qua băng chuyền. Ở các nhà máy, thông thường các băng chuyền được nối dài liên tiếp nhau và chạy cùng tốc độ, đồng nghĩa với việc sử dụng một lúc nhiều đồng cơ để quay băng chuyền và di 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 1.2. RV3100 Incremental Encoder của IFM

2. TỔNG QUAN VỀ VI ĐIỀU KHIỂN 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, …

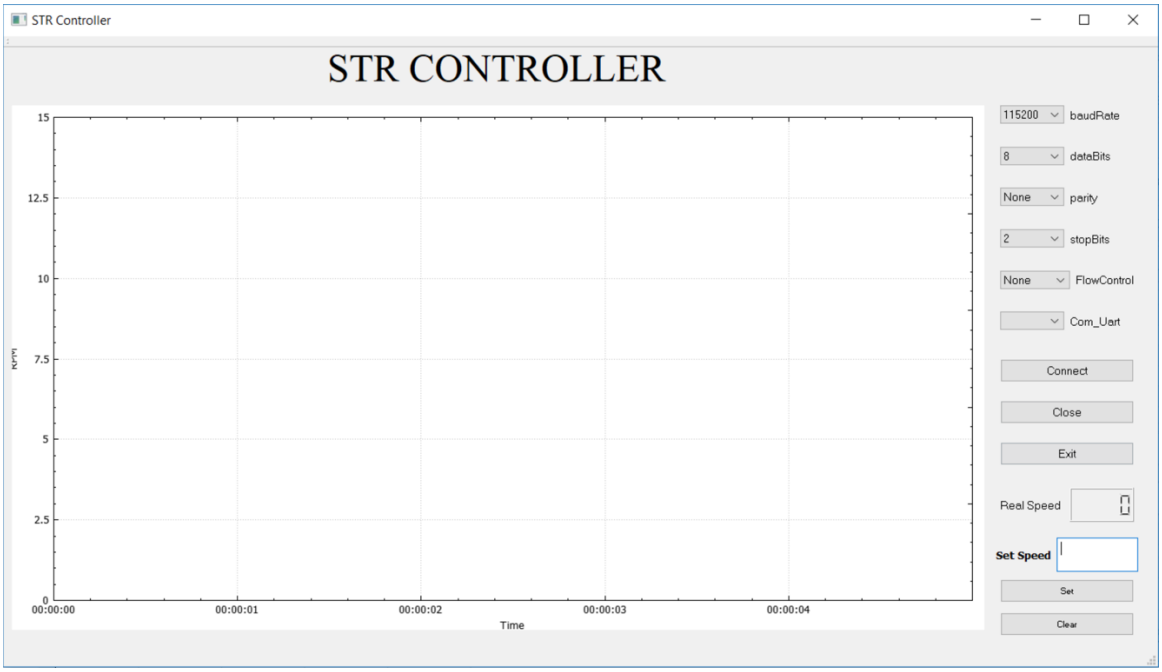


Hình 1.3. Hình ảnh vi điều khiển STM32F407

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.

3. TỔNG QUAN VỀ PHẦN MỀM VIẾT GUI QT CREATOR

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 1.4. Giao diện GUI điều khiển động cơ

4. 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.

4.1. 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.

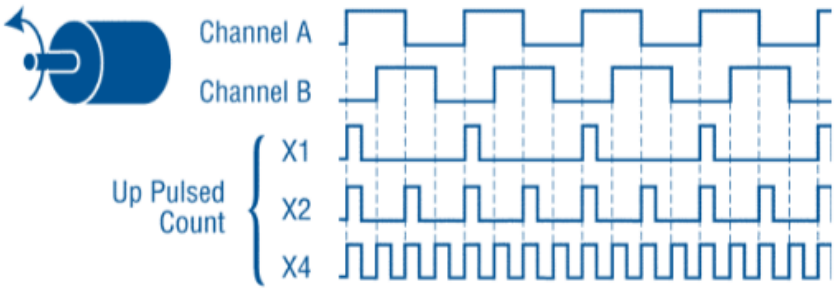
Hình 1.5. Incremental encoder's disk

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.

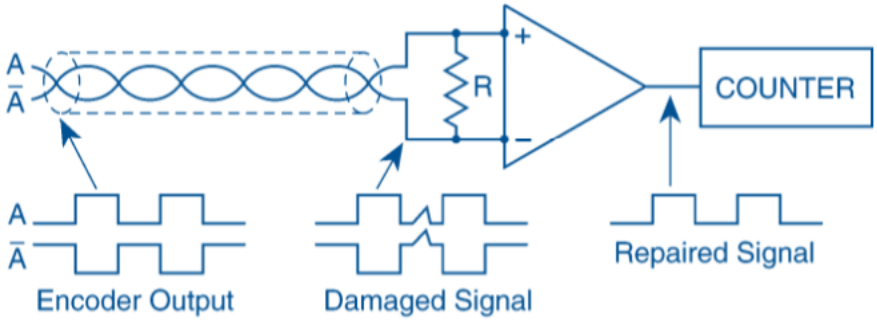
Tốc độ động cơ: thông qua tần số (hoặc chu kỳ) của xung trả về.

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



Hình 1.6. 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 1.7. Khử nhiễu dùng kênh đảo của Encoder

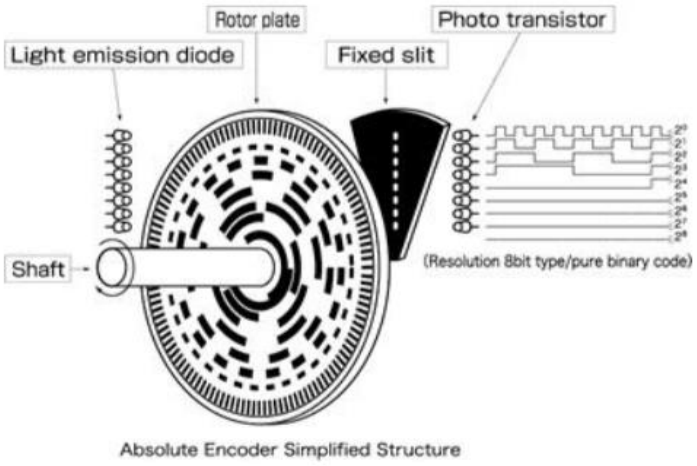
Qua đó, tín hiệu hồi tiếp từ Encoder tương đối có thể giúp ta thực hiện việc điều khiển chính xác tốc độ, vị trí tương đối và vòng quay. Bằng cách đọc các cạnh xung lên và/hoặc xung xuống của hai kênh A và B, ta có thể xác định được vị trí. Nếu kết hợp thêm với một bộ đếm Timer, ta hoàn toàn có thể tính 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.

4.2. 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 1.8. 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 1.9. Dạng sóng 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 1.1. 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

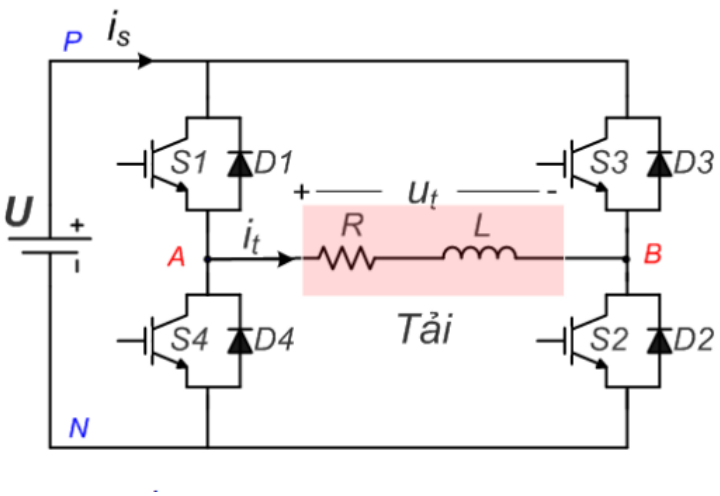
Ở trong bài này ta sẽ thực hiện mạch và giao tiếp máy tính để tương tác với Encoder loại tương đối.

4.3. Bộ điều khiển tự chỉnh STR (Self-tuning Regulator)

CHƯƠNG 2: XÂY DỰNG PROJECT

1. GIA CÔNG MẠCH CẦU H ĐIỀU KHIỂN ĐỘNG CƠ

1.1. Sơ đồ nguyên lý (Schematics)

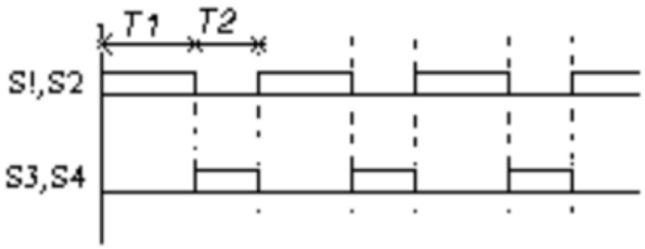


Hình 1.10. Sơ đồ nguyên lý mạch cầu H

Mạch cầu H, hay còn gọi là bộ biến đổi kép dạng tổng quát có 2 giản đồ kích như sau:

- Với giản đồ kích 1, ta sẽ kích từng cặp (S1, S2) và (S3, S4) ngược pha nhau, tức là cùng một lúc, S1&S2 sẽ được kích hoặc S3&S4 được kích. Theo đó, dòng ngõ ra có thể chạy theo cả 2 chiều trên tải, dẫn đến điện áp Ut sẽ biến thiên giữa -U và U. Giá trị trung bình của điện áp ngõ ra (ở chế độ dòng liên tục) là:

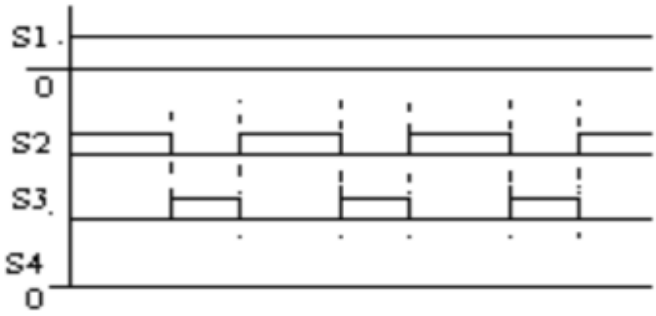
$$U_t = U(\frac{2.T_1}{T} - 1) = U(2\gamma - 1)$$



Hình 1.11. Giản đồ xung kích 1 cho S1, S2, S3 và S4

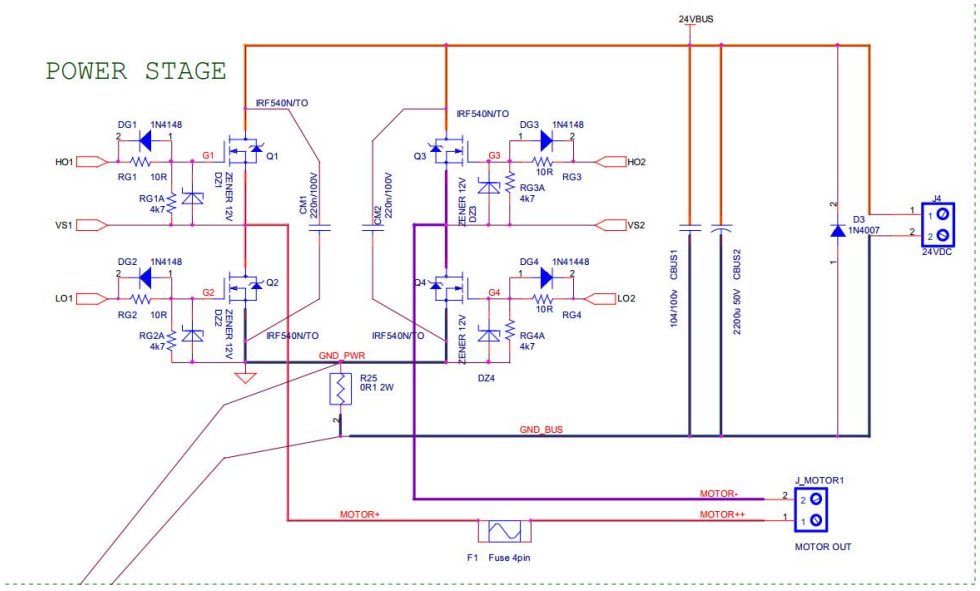
- Với giản đồ kích 2, ta sẽ giữ S1 luôn ở trạng thái ON và S4 luôn ở trạng thái OFF. S2 và S3 sẽ được kích ngược pha với nhau. Với giản đồ kích này thì Ut sẽ luôn dương và có công thức như sau, với T_1 là thời gian S2 dẫn:

$$U_t = U\frac{T_1}{T} = U\gamma$$

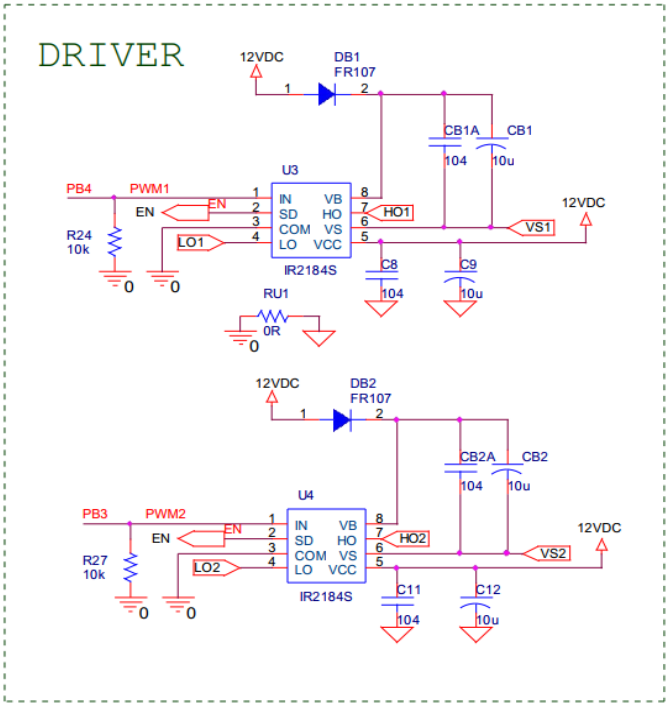


Hình 1.12. Giản đồ xung kích 2 cho S1, S2, S3 và S4

Với giản đồ kích 2, để Ut < 0, thì S3 luôn ON và S2 luôn OFF, kích ngược pha S1 và S4.



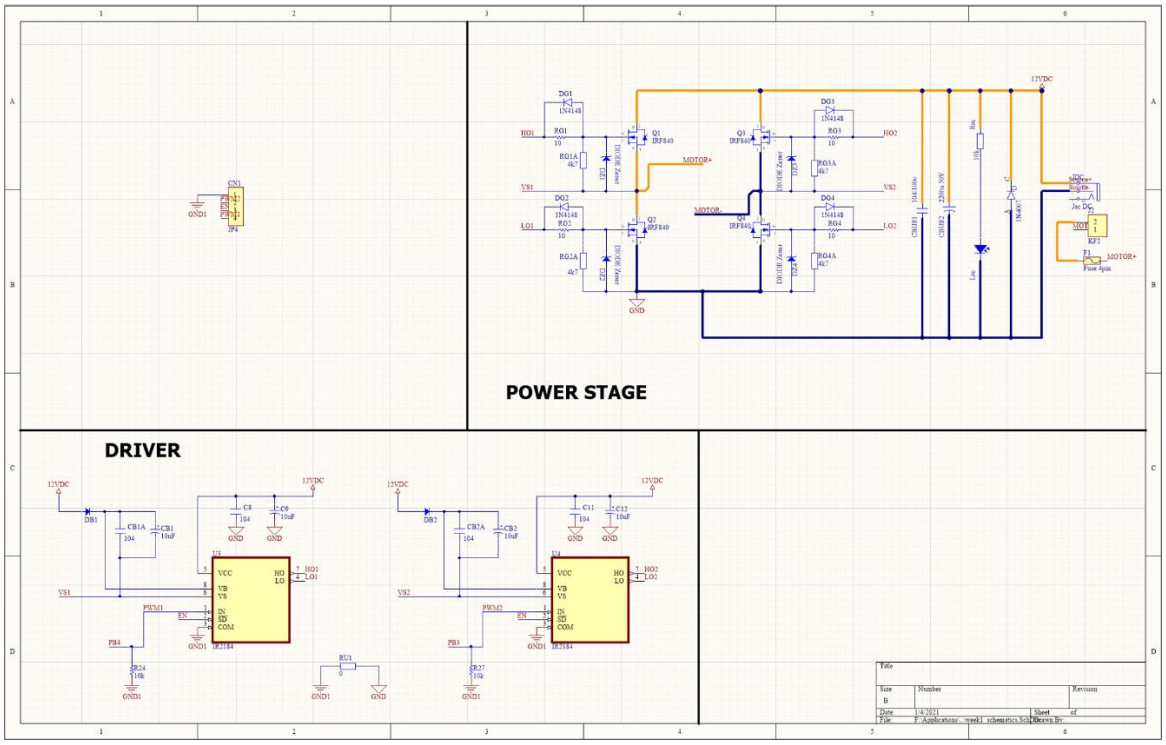
Hình 1.13. Mạch cầu H của PIF



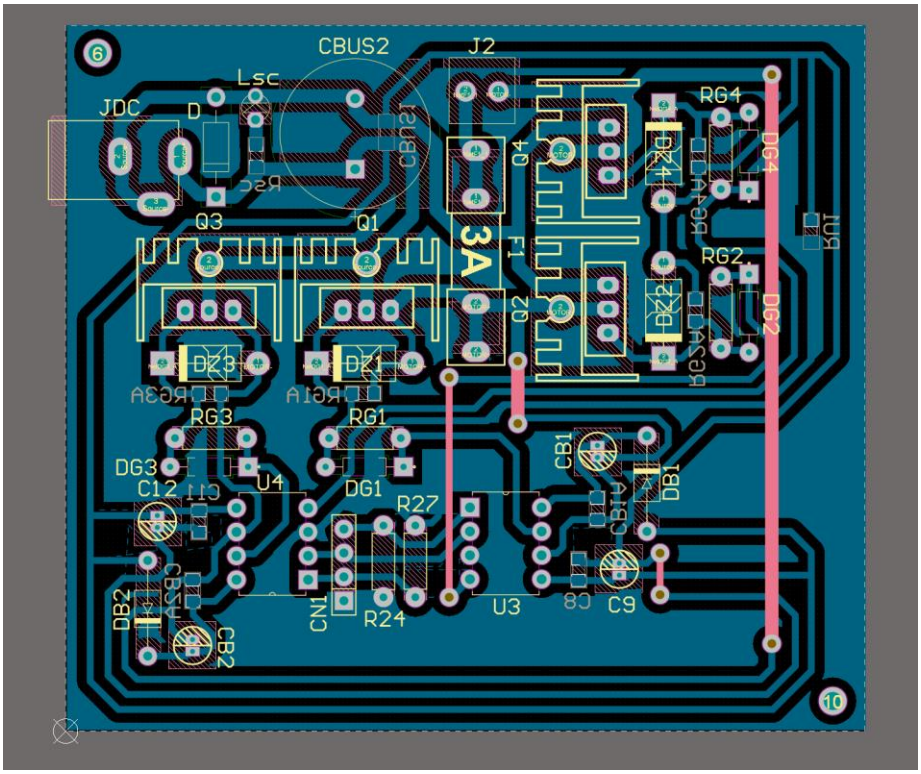
Hình 1.14. Mạch driver

1.2. Sơ đồ PCB

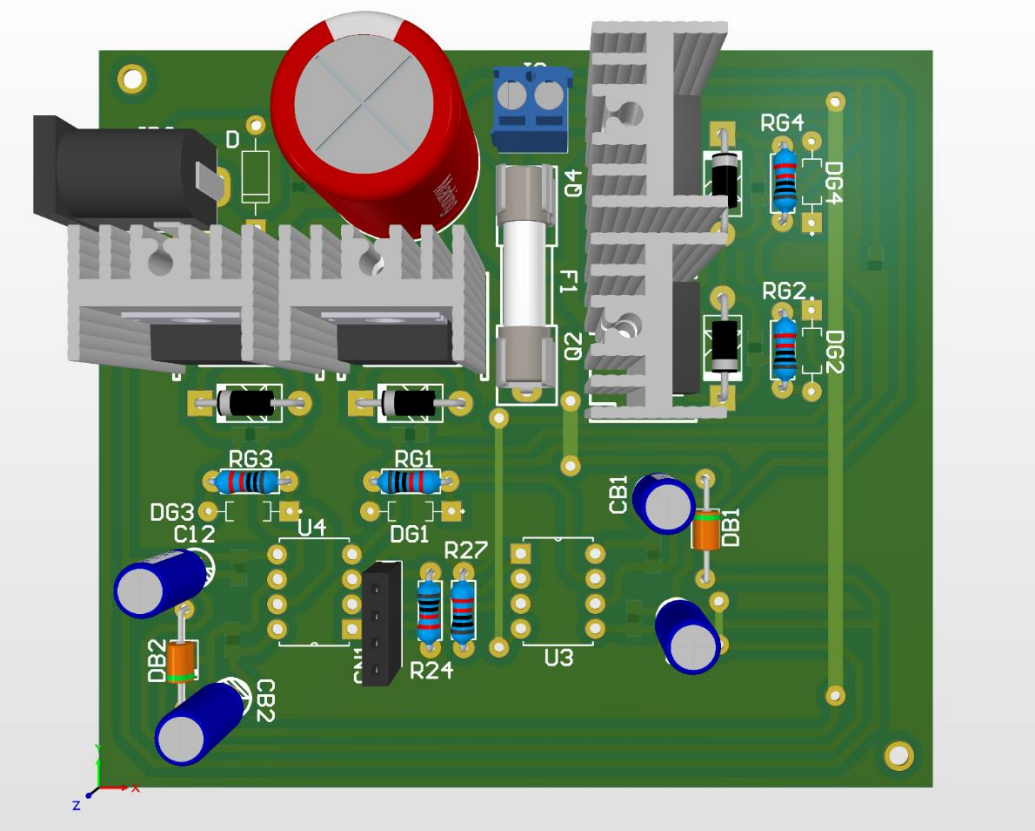
Nhóm sử dụng Altium để vẽ sơ đồ nguyên lý mạch Schematic và mạch gia công PCB



Hình 1.15. Sơ đồ nguyên lý mạch cầu H sử dụng Altium



Hình 1.16. Sơ đồ mạch in PCB sử dụng Altium

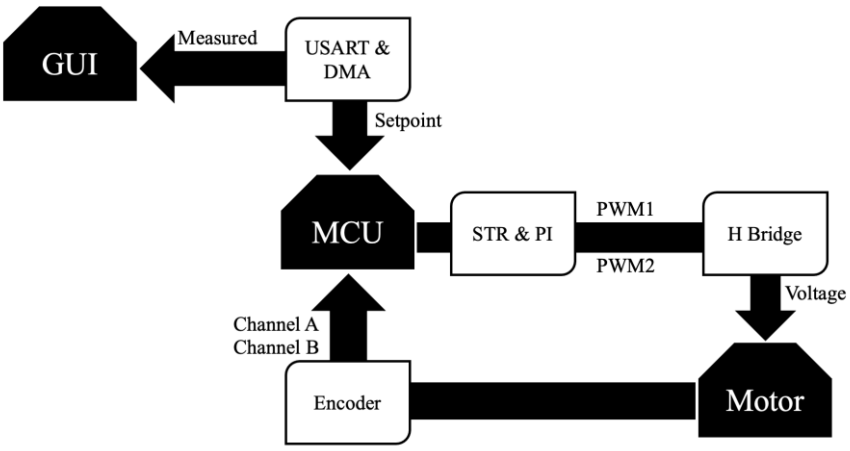


Hình 1.17. Sơ đồ mạch in PCB sử dụng Altium

Linh kiện chính nhóm sử dụng thi công mạch PCB:

- Driver **IR2184** (Half-Bridge, 1 con **IR2184** cấp xung kích PWM cho 1 nhánh FET; mỗi nhánh FET (1 half-bridge, hay 1 LEG, hay 1 ARM) gồm 2 FET, con nối với nguồn được gọi là con High-side, con nối với GND được gọi là Low-side.
- Để kích được con High-side phải dùng mạch kích kiểu bootstrap. Mà khi dùng mạch kích kiểu bootstrap thì duty không thể đạt tới 100%.
- Cần 4 con FET để mạch hoạt động. Ở đây dùng **IRF540**.

2. LẬP TRÌNH VI ĐIỀU KHIỂN STM32F407



Hình 2.1. Sơ đồ khối liên kết GUI, MCU và động cơ

2.1. Khởi tạo các struct và variables

Các biến toàn cục gồm có baudrate, stopbit, Rx & Tx Buffer Size của USART. Số xung trên 1 vòng quay của động cơ được cho bởi NSX là 374 xung. Ngoài ra nhóm tác giả còn định nghĩa 2 struct với 2 object là mainMotor và m_data. mainMotor là biến để lưu giá trị về vận tốc động cơ, m_data là để phục vụ việc giao tiếp kiểu floating point IEEE 32-bit với GUI thông qua USART. Protocol giao tiếp MCU và GUI sẽ được đề cập ở phần sau.

```
#define MAIN_BAUDRATE 115200
#define MAIN_STOPBITS USART_StopBits_2
#define MAIN_PARITY USART_Parity_No
#define BUFF_SIZE_RX 14
#define BUFF_SIZE_TX 14
#define RPMmax 250
#define RPMmin -250
#define IC_StartPoint 1000
#define counterPerRound 374

extern char RXBuffer[BUFF_SIZE_RX];
extern char TXBuffer[BUFF_SIZE_TX];

struct motor_Values{
    float measure_RPM;
    int16_t counter;
} extern mainMotor;

union ByteToFloat{
    float myfloat;
    char mybyte[4];
} extern m_data;
```

2.2. Lập trình xuất PWM điều khiển mạch cầu H

Dựa trên mạch cầu H đã gia công, động cơ có thể được điều khiển theo hai xung PWM, sau đây tạm gọi là PWM1 và PWM2. Có hai cách để điều khiển cầu H:

- Điều khiển chu kỳ nhiệm vụ một xung và cho xung còn lại nối với đất, đảo vai trò của hai xung để đổi chiều động cơ.

- Điều khiển một trong hai xung và cho chân còn lại là xung đảo của nó. Theo đó, ở chu kỳ nhiệm vụ là 50% thì động cơ sẽ đứng yên, lớn hơn 50% thì quay một chiều và bé hơn 50% thì quay chiều còn lại.

Ở đây nhóm tác giả chọn phương pháp cấp xung đầu tiên để thông qua mạch cầu H điều khiển động cơ, cụ thể là Channel 1 và Channel 3 của Timer 1 với tần số là 1KHz. Code để config Timer 1 và hai Channel như sau:

```
TIM_TimeBaseInitTypeDef     TIM_BaseStruct;
TIM_OCInitTypeDef           TIM_OCStruct;
GPIO_InitTypeDef            GPIO_InitStruct;

RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM1, ENABLE);

GPIO_InitStruct.GPIO_Pin = GPIO_Pin_8 | GPIO_Pin_10;
GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;
GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF;
GPIO_InitStruct.GPIO_Speed = GPIO_Speed_100MHz;
GPIO_Init(GPIOA, &GPIO_InitStruct);

GPIO_PinAFConfig(GPIOA, GPIO_PinSource8, GPIO_AF_TIM1);
GPIO_PinAFConfig(GPIOA, GPIO_PinSource10, GPIO_AF_TIM1);

TIM_BaseStruct.TIM_Prescaler         = 83;
TIM_BaseStruct.TIM_Period            = 999;
TIM_BaseStruct.TIM_CounterMode       = TIM_CounterMode_Up;
TIM_BaseStruct.TIM_ClockDivision     = 0;
```

```
TIM_BaseStruct.TIM_RepetitionCounter        = 0;
TIM_TimeBaseInit(TIM1, &TIM_BaseStruct);
TIM_Cmd(TIM1, ENABLE);
TIM_OCStruct.TIM_OCMode             = TIM_OCMode_PWM1;
TIM_OCStruct.TIM_OutputState        = TIM_OutputState_Enable;
TIM_OCStruct.TIM_OCPolarity         = TIM_OCPolarity_High;

TIM_OCStruct.TIM_Pulse              = 33;
TIM_OC1Init(TIM1, &TIM_OCStruct);
TIM_OC1PreloadConfig(TIM1, TIM_OCPreload_Enable);

TIM_OC3Init(TIM1, &TIM_OCStruct);
TIM_OC3PreloadConfig(TIM1, TIM_OCPreload_Enable);

TIM_ARRPreloadConfig(TIM1,ENABLE);
TIM_Cmd(TIM1,ENABLE);
TIM_CtrlPWMOutputs(TIM1, ENABLE);
```

2.3. Lập trình đọc tốc độ từ Encoder sử dụng Input Capture

Vi điều khiển STM32F4 cung cấp một chế độ để đọc Encoder tự động mà không tốn tài nguyên xử lý gọi là Input Capture. Thông qua chế độ này, với Timer 5 ở chế độ Counter ta có thể đọc các xung từ Channel A và Channel B của Encoder trả về thông qua biến Counter của Channel 5 (ở đây nhóm tác giả đọc Input Capture cho cả 4 xung của 2 kênh). Sau mỗi chu kỳ cố định, ta sử dụng một Timer khác (ở đây là Timer 3) để tạo ngắt và đọc tốc độ động cơ. Chu kỳ tràn của Timer 3 được chọn bằng 5ms, trong khoảng thời gian này Counter của Timer 5 sẽ chỉ có thể đếm lên hoặc xuống đến một giá trị nhất định trong khoảng dưới 200. Ta đặt giá trị khởi động của Counter Timer 5 là 1000, sau 5ms ta lấy

giá trị Counter Timer 5 trừ đi 1000 và nhân chia tỉ lệ với số xung trên vòng (được cung cấp bởi nhà sản xuất) thì ta sẽ ra được vận tốc và chiều quay của động cơ.

Sau đây là code config mode Input Capture cho Timer 5 và ngắt cho Timer 3:

```
GPIO_InitTypeDef            GPIO_InitStruct;
NVIC_InitTypeDef            NVIC_InitStruct;
TIM_TimeBaseInitTypeDef     TIM_BaseStruct;
TIM_ICInitTypeDef           TIM_ICInitStruct;

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;
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, IC_StartPoint);
	TIM_Cmd(TIM5, ENABLE);
	TIM_ClearFlag(TIM5, TIM_FLAG_Update);

	RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);
	TIM_BaseStruct.TIM_Prescaler		= 8399;
	TIM_BaseStruct.TIM_Period		= 249; //delay 5ms
	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;
NVIC_Init(&NVIC_InitStruct);
```

Hàm ngắt đọc giá trị vận tốc động cơ được định nghĩa như sau:

```
void TIM3_IRQHandler(void){
    if (TIM_GetITStatus(TIM3, TIM_IT_Update) != RESET){
        TIM_Cmd(TIM3, DISABLE);
        TIM_SetCounter(TIM3, 0);

        // Calculate the speed
        int lalala = TIM5->CNT;
        mainMotor.measure_RPM = (lalala - IC_StartPoint)*60*40/
(4*counterPerRound);
        if ((mainMotor.measure_RPM > 300) || (mainMotor.measure_RPM
< -300) || (mainMotor.measure_RPM == 0)){
            mainMotor.measure_RPM = 1;
        }
        checkSpeed();
        TIM_SetCounter(TIM5, IC_StartPoint);
        TIM_Cmd(TIM3, ENABLE);
        TIM_ClearITPendingBit(TIM3, TIM_IT_Update);
    }
}
```

2.4. Lập trình giao tiếp với GUI thông qua USART và 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 tới tác vụ từ CPU, tiết kiệm tài nguyên CPU cho các hoạt động khác. Ở đây nhóm sử dụng USART3 kết hợp với DMA1.

Nhiệm vụ của GUI là để quan sát đáp ứng ngõ ra, đồng thời máy tính có nhiệm vụ gửi setpoint cho MCU để đưa vào STR. Để thực hiện được điều này, MCU phải có cơ chế gửi giá trị vận tốc động cơ lên GUI theo chu kỳ, song song với việc này, GUI cũng phải ngắt theo chu kỳ để đọc giá trị được MCU gửi lên. Ngoài tín hiệu theo chu kỳ, tín hiệu đặt setpoint được người dùng gửi do đó MCU phải có cơ chế nhận giá trị này.

Sau đây là code config USART3 và DMA1 tương ứng:

```
GPIO_InitTypeDef            GPIO_InitStruct;
USART_InitTypeDef           USART_InitStruct;
DMA_InitTypeDef             DMA_InitStructure;
NVIC_InitTypeDef            NVIC_InitStruct;

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);

/* Khoi tao chan TX & RX Uart*/
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);
```

Sau đây là hàm ngắt DMA đọc giá trị setpoint gửi về từ GUI:

```
void DMA1_Stream1_IRQHandler(void){
    char checksum_Rx = 0;
    DMA_ClearITPendingBit(DMA1_Stream1, DMA_IT_TCIF1);
    GPIO_ToggleBits(GPIOD, GPIO_Pin_12);
    if(RXBuffer[0]=='$' && RXBuffer[1]=='S' && RXBuffer[2]=='P' && RXBuffer[3]=='E' && RXBuffer[4]=='E' && RXBuffer[5]=='D' && RXBuffer[6]==','){
        for(int k = 0; k < 4; k++){
            m_data.mybyte[3-k] = RXBuffer[7+k];
            checksum_Rx += RXBuffer[7+k];
        }
        if(checksum_Rx == RXBuffer[11])
            if((m_data.myfloat <= 250) && (m_data.myfloat >= -250)){
                PI.setpoint = m_data.myfloat;
                if(PI.setpoint >= 0){
                    PI.direction = 1;
                }else PI.direction = -1;
            }
```

```
                }
        DMA_Cmd(DMA1_Stream1, ENABLE);
    }
```

Để MCU gửi dữ liệu lên cho máy tính xử lý thực hiện việc hiển thị giá trị cũng như vẽ đồ thị vận tốc của động cơ, nhóm quy định Frame truyền từ vi điều khiển gửi lên cho máy tính và ngược lại gồm 14 bytes được quy định như sau:

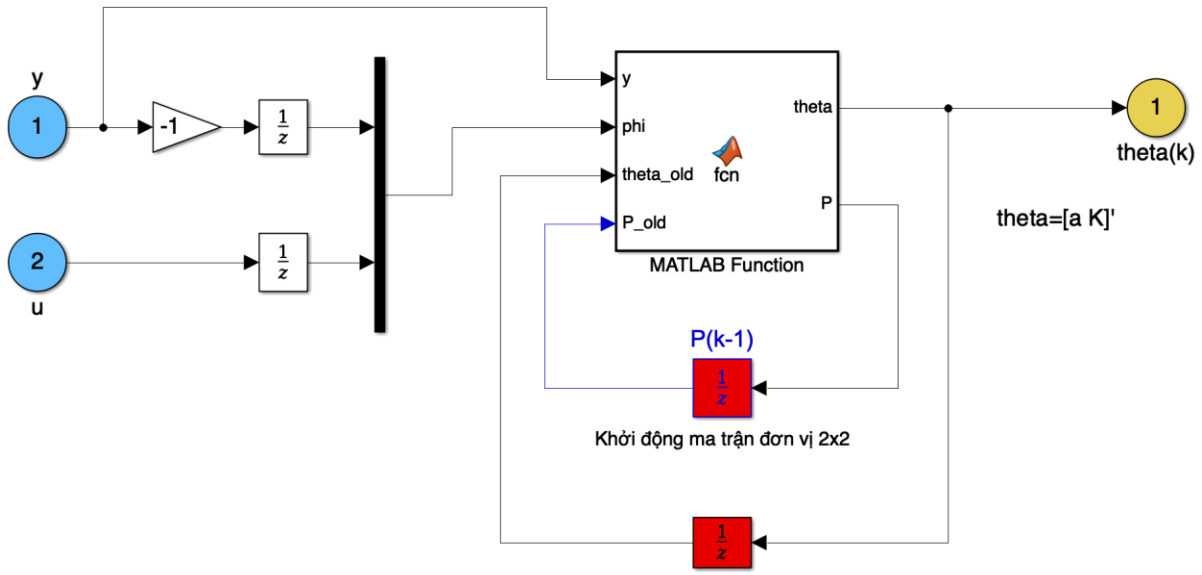
```
<'$SPEED,'><4-byte Float data><1-byte Checksum><'\r'><'\n'>
```

Trong đó 7 bytes đầu quy định thông số được gửi lên cho máy tính là thông số gì, 4 bytes data chính là 4 bytes lưu giá trị của dữ liệu kiểu float, 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.

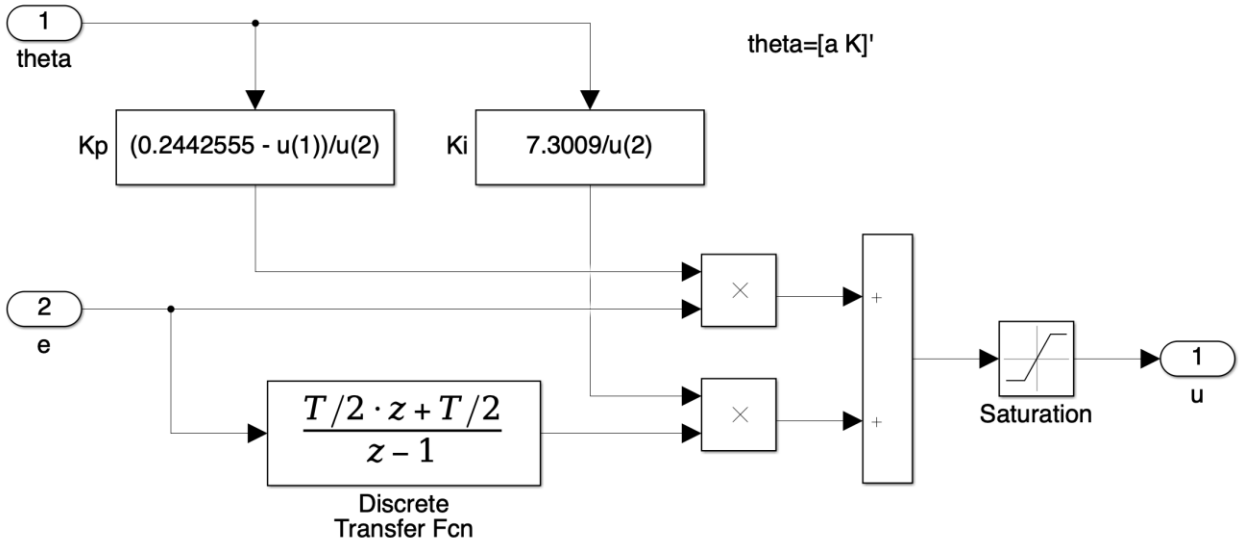
2.5. Lập trình bộ điều khiển tự chỉnh STR (Self-tuning Regulator)

2.5.1. Khai báo các struct và variables

Dựa trên lý thuyết, sơ đồ mô phỏng simulink của khối LMS Estimation và hàm MATLAB Function, ta có thể liệt kê ra các biến cần thiết, nhóm tác giả đặt các biến này cùng một struct là STRPara và PIPara.



Hình 2.2. Sơ đồ mô phỏng Simulink của khối LMS Estimation



Hình 2.3. Sơ đồ mô phỏng simulink của khối PI Controller

```
#define ARRAY_MAX_SIZE 2


struct STRPara{
    float lambda, error, y, pre_y, u, pre_u;
    float phi[ARRAY_MAX_SIZE][ARRAY_MAX_SIZE];
    float phi_T[ARRAY_MAX_SIZE][ARRAY_MAX_SIZE];
```

```
        float theta[ARRAY_MAX_SIZE][ARRAY_MAX_SIZE];
        float pre_theta[ARRAY_MAX_SIZE][ARRAY_MAX_SIZE];
        float L[ARRAY_MAX_SIZE][ARRAY_MAX_SIZE];
        float P[ARRAY_MAX_SIZE][ARRAY_MAX_SIZE];
        float pre_P[ARRAY_MAX_SIZE][ARRAY_MAX_SIZE];
    } extern STR;


    struct PIPara{
        float T, error, pre_error, setpoint;
    } extern PI;
```

2.5.2. Lập trình thư viện các hàm cần thiết

Để tính toán ra tín hiệu điều khiển và mô phỏng lại bộ điều khiển tự chỉnh STR, ta cần có các hàm tính toán với ma trận. Vì ngôn ngữ C không hỗ trợ các hàm trả về ma trận nên ta sẽ đưa ma trận trả về vào một trong các đối số của hàm, việc này khiến các hàm có lượng đối số lớn. Sau đây là thư viện các hàm thao tác với ma trận cần dùng cho bộ điều khiển tự chỉnh STR được lưu trong file user_STR_Controller.c:

```
    void matrixDelete(float inputMatrix[][ARRAY_MAX_SIZE], int rowInput, int
columnInput){
        for(int i = 0; i < rowInput; i++){
            for(int j = 0; j < columnInput; j++){
                inputMatrix[i][j] = 0;
            }
        }
    }


    void matrix_numberMultiply(float inputMatrix[][ARRAY_MAX_SIZE], float
outputMatrix[][ARRAY_MAX_SIZE], int rowInput, int columnInput, float gain){
```

```
            for (int row = 0; row < rowInput; row++){
                for (int col = 0; col < columnInput; col++){
                    outputMatrix[row][col] = inputMatrix[row][col]*gain;
                }
            }
        }

        void matrixCreate(float returnMatrix[][ARRAY_MAX_SIZE], int outputRow,
int outputColumn, float data[ARRAY_MAX_SIZE]){
            for (int row = 0; row < outputRow; row++){
                for (int col = 0; col < outputColumn; col++){
                    returnMatrix[row][col] = data[row*outputColumn+col];
                }
            }
        }

        void matrixMultiply(float firstMatrix[][ARRAY_MAX_SIZE], float
secondMatrix[][ARRAY_MAX_SIZE], float returnMatrix[][ARRAY_MAX_SIZE],
int rowFirst, int columnFirst, int rowSecond, int columnSecond){
            if (columnFirst == rowSecond){
                matrixDelete(returnMatrix, rowFirst, columnSecond);
                for (int row = 0; row < rowFirst; row++){
                    for (int col = 0; col < columnSecond; col++){
                        for (int cnt = 0; cnt < columnFirst; cnt++){
                            returnMatrix[row][col] += firstMatrix[row][cnt]*
secondMatrix[cnt][col];
                        }
```

```
                }
            }
        }
    }

    void matrixSum(float firstMatrix[][ARRAY_MAX_SIZE], float
secondMatrix[][ARRAY_MAX_SIZE], float returnMatrix[][ARRAY_MAX_SIZE],
int rowInput, int columnInput){
        for (int row = 0; row < rowInput; row++){
            for (int col = 0; col < columnInput; col++){
                returnMatrix[row][col] = firstMatrix[row][col] +
secondMatrix[row][col];
            }
        }
    }

    void matrixTranspose(float inputMatrix[][ARRAY_MAX_SIZE], float
returnMatrix[][ARRAY_MAX_SIZE], int rowInput, int columnInput){
        int rowOutput = columnInput;
        int columnOutput = rowInput;
        for (int row = 0; row < rowOutput; row++){
            for (int col = 0; col < columnOutput; col++){
                returnMatrix[row][col] = inputMatrix[col][row];
            }
        }
    }
```

2.5.3. Lập trình bộ STR và PI

Đầu tiên ta định nghĩa 2 hàm để khai báo các giá trị đầu cho các biến PIPara và STRPara, với thời gian lấy mẫu được chọn là 25ms và lambda bằng 0.97. Ma trận θ là ma trận rand(2,1), ở đây nhóm tác giả chọn 2 giá trị khởi đầu của ma trận đều bằng 1. Ma trận P là ma trận eye(2) theo như lý thuyết.

```
void initPI_Para(){
    PI.T            = 0.025;
    PI.pre_error    = 0;
    PI.error        = 0;
    PI.setpoint     = 0;
}
void initSTR_Para(){
    STR.lambda      = 0.97;
    STR.error       = 0;
    STR.pre_y       = 0;
    STR.y           = 0;
    STR.pre_u       = 0;
    STR.u           = 0;
    float tmp_data[ARRAY_MAX_SIZE]          = {1, 1};
    float tmp_data1[ARRAY_MAX_SIZE*2]       = {1, 0, 0, 1};

    matrixDelete(STR.phi, ARRAY_MAX_SIZE, ARRAY_MAX_SIZE);
    matrixDelete(STR.phi_T, ARRAY_MAX_SIZE, ARRAY_MAX_SIZE);
    matrixCreate(STR.theta, 2, 1, tmp_data);
    matrixCreate(STR.pre_theta, 2, 1, tmp_data);
    matrixDelete(STR.L, ARRAY_MAX_SIZE, ARRAY_MAX_SIZE);
```

```
        matrixCreate(STR.P, 2, 2, tmp_data1);
        matrixCreate(STR.pre_P, 2, 2, tmp_data1);
    }
```

Để mô phỏng lại 2 khối là LMS Estimation và PI Controller trên vi điều khiển, nhóm tác giả định nghĩa 2 hàm tương ứng, với các lệnh sử dụng thư viện tính toán với ma trận có sẵn để từng bước tính ra θ, tín hiệu điều khiển u…. Sau mỗi hàm này, các biến STR.u, STR.theta, STR.P, sẽ tự động cập nhật cho biến pre tương ứng. Riêng đặc thù của 2 biến STR.y và PI.error thì sẽ được cập nhật biến pre sau mỗi lần đọc tốc độ động cơ từ hàm ngắt Timer 3 đã có đề cập ở trên.

```
void PI_Controller(){
    STR.pre_u = STR.u;
    STR.u = STR.pre_u + (-0.6447 - STR.theta[0][0])*(PI.error - PI.pre_error)
/STR.theta[1][0] + 2.0498*PI.T*(PI.error + PI.pre_error)/(STR.theta[1][0]*2);
}

void LMS_Estimation(){
    float tmp_matrix1[ARRAY_MAX_SIZE][ARRAY_MAX_SIZE];
    matrixDelete(tmp_matrix1, ARRAY_MAX_SIZE, ARRAY_MAX_SIZE);
    float tmp_matrix2[ARRAY_MAX_SIZE][ARRAY_MAX_SIZE];
    matrixDelete(tmp_matrix2, ARRAY_MAX_SIZE, ARRAY_MAX_SIZE);
    float tmp_matrix3[ARRAY_MAX_SIZE][ARRAY_MAX_SIZE];
    matrixDelete(tmp_matrix3, ARRAY_MAX_SIZE, ARRAY_MAX_SIZE);

    float tmp_data[ARRAY_MAX_SIZE] = {-STR.pre_y, STR.pre_u};

     matrixCreate(STR.phi, 2, 1, tmp_data);
```

```
        matrixTranspose(STR.phi, STR.phi_T, 2, 1);


        matrixMultiply(STR.phi_T, STR.pre_theta, tmp_matrix1, 1, 2, 2, 1);
         STR.error = STR.y - tmp_matrix1[0][0];                    //Calculate error


        matrixMultiply(STR.pre_P, STR.phi, tmp_matrix1, 2, 2, 2, 1);
        matrixMultiply(STR.phi_T, STR.pre_P,  tmp_matrix2, 1, 2, 2, 2);
         matrixMultiply(tmp_matrix2, STR.phi, tmp_matrix3, 1, 2, 2, 1);
        matrix_numberMultiply(tmp_matrix1, STR.L, 2, 1, 1/tmp_matrix3[0][0]);
//Calculate L


        matrixMultiply(STR.pre_P, STR.phi, tmp_matrix1, 2, 2, 2, 1);
        matrixMultiply(tmp_matrix1, STR.phi_T, tmp_matrix2, 2, 1, 1, 2);
        matrixMultiply(tmp_matrix2, STR.pre_P, tmp_matrix3, 2, 2, 2, 2);
        matrixMultiply(STR.phi_T, STR.pre_P, tmp_matrix1, 1, 2, 2, 2);
         matrixMultiply(tmp_matrix1, STR.phi,  tmp_matrix2, 1, 2, 2, 1);
         matrix_numberMultiply(tmp_matrix3, tmp_matrix1, 2, 2, -1/(tmp_matrix2[0][0]
+ STR.lambda));
        matrixSum(STR.pre_P, tmp_matrix1, tmp_matrix2, 2, 2);
        matrix_numberMultiply(tmp_matrix2, STR.P, 2, 2, 1/STR.lambda); //Calculate P


        matrix_numberMultiply(STR.L, tmp_matrix1, 2, 1, STR.error);
        matrixSum(tmp_matrix1, STR.pre_theta, STR.theta, 2, 1);        //Calculate theta


        matrixDelete(tmp_matrix1, ARRAY_MAX_SIZE, ARRAY_MAX_SIZE);
   matrixSum(tmp_matrix1, STR.P, STR.pre_P, 2, 2);
```

```
        matrixDelete(tmp_matrix1, ARRAY_MAX_SIZE, ARRAY_MAX_SIZE);
    matrixSum(tmp_matrix1, STR.theta, STR.pre_theta, 2, 1);
}
```

2.6. Kết hợp vào hàm main()

```
int main(void){
        SystemInit();
        USART_DMA_Configuration(MAIN_BAUDRATE);
        My_GPIO_Init();
        My_TIMER_Init();
        My_PWM_Init();
        Encoder_Init();
        initPI_Para();
        initSTR_Para();
        while(1){
                if(LMS_Pend_Flag)
                PI_Controller();
                if (STR.u > 990)
                        STR.u = 990;
                else if (STR.u < -990)
                        STR.u = -990;
                convert();
                STR.pre_y = STR.y;
                STR.y = mainMotor.measure_RPM;
                PI.pre_error = PI.error;
                PI.error = PI.setpoint - STR.y;
```

```
                if (LMS_Pend_Flag)
                        LMS_Estimation();
                delay_ms(PI.T*1000);
            }
        }
```

3. LẬP TRÌNH GUI BẰNG QT CREATOR

3.1. Lập trình khởi tạo thông số và kết nối với vi điều khiển

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 và ngược lại với chân RX của USB. Cụ thể ở đây nhóm sử dụng USART 3 với chức năng DMA.

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);
        }
```

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);
}
```

3.2. Lập trình giao tiếp với vi điều khiển STM32F407

Để vi xử lý gửi dữ liệu lên cho máy tính xử lý thực hiện việc hiển thị giá trị cũng như vẽ đồ thị vận tốc của động cơ, nhóm quy định Frame truyền từ vi xử lý gửi lên cho máy tính gồm 14 bytes được quy định như sau:

<’$SPEED,’><4-byte Float data><1-byte Checksum><’\r’><’\n’>

Trong đó 7 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 tốc độ và vẽ đồ thị, 1 byte checksum để kiểm tra lỗi và 2 bytes quy định kết thúc frame truyền

Để máy tính gửi giá trị đặt xuống cho vi xử lý, nhóm quy định frame truyền tín hiệu đặt như sau:

<'$SPEED,'><4-byte Float data><1-byte Checksum><'\r'><'\n'>

Cũng tương tự như khi vi xử lý gửi dữ liệu lên cho máy tính, 7 bytes đầu quy định thông số gửi xuống vi xử lý là giá trị đặt, 4 bytes data kiểu Union, sau khi gửi xuống máy tính sẽ được chuyển từ 4 bytes thành 1 biến float để điều chỉnh tốc độ động cơ.

3.3. Lập trình vẽ đồ thị đáp ứng tốc độ đặt và tốc độ ngõ ra

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 Real Speed đồng thời hàm ReadData() cũng được chạy để nhận giá trị vận tốc gửi lên từ vi xử lý, sau đó hàm RealTimeData() sẽ được chạy để vẽ đồ thị vận tốc với giá trị vừa gửi lên.

Hàm ReadData()

```
void MainWindow::ReadData()
{
 QByteArray byte_data = SerialPort->readAll();
 if(!byte_data.isEmpty()){
if(byte_data.startsWith("$SPEED,")&&byte_data[13]=='\n'&&byte_data[12]=
='\r')
 {
 for (int i=0;i<4;i++)
    CheckSumRX+=byte_data[7+i];
 qDebug()<<CheckSumRX<<byte_data[11];
    if(CheckSumRX==byte_data[11]){
        for(int k= 0;k<4;k++){
        m_data_RX.mybyte[3-k] = byte_data[7+k];
        }
```

```
            ui->RealSpeed->display(m_data_RX.myfloat);
            }
            CheckSumRX=0;
            RealTimeData();
            }
        } }
```

Hàm RealTimeData()

```
void MainWindow::RealTimeData()
{
 static int yMax=0;
 static int yMin=0;
 static double temp_time=0;
 static QTime time(QTime::currentTime());
 if(flag_PlotTimer)
 {
 flag_PlotTimer = false;
 temp_time=time.elapsed()/1000.0;
 yMax=0;
 }
 double key = time.elapsed()/1000.0;
 ui->customPlot->graph(0)->addData(key, m_data_RX.myfloat);
 ui->customPlot->graph(1)->addData(key,Out);
 ui->customPlot->graph(0)->rescaleValueAxis();
 ui->customPlot->xAxis->setRange(temp_time, key, Qt::AlignLeft);
 if(setpoint >= yMax)
    yMax = setpoint;
```

```
        if(m_data_RX.myfloat >= yMax)
            yMax = (int)m_data_RX.myfloat;
        if(setpoint <= yMin)
            yMin = setpoint;
        if(m_data_RX.myfloat <= yMin)
            yMin = (int)m_data_RX.myfloat;
        ui->customPlot->yAxis->setRange(-300, 300);//(yMin-200, yMax+200);
        ui->customPlot->replot();
        // calculate frames per second:
        static double lastFpsKey;
        static int frameCount;
        ++frameCount;
        lastFpsKey = key;
        frameCount = 0;
       }
```

3.4. Lập trình bộ lọc tín hiệu đặt

Để gửi giá trị đặt từ GUI xuống cho vi xử lý, ta kết nối nút nhất Set Speed với sự kiện clicked(). Mỗi khi nhấn nút Set Speed, chương trình sẽ lấy giá trị trong ô Set Point để gửi xuống vi xử lý. Tuy nhiên nếu ta gửi tín hiệu đột ngột dạng xung vuông, vi xử lý cũng như động cơ sẽ không đáp ứng kịp dẫn đến vọt lố, vì vậy thay vì gửi tín hiệu đột ngột, nhóm sử dụng bộ lọc thông thấp có hàm truyền liên tục

$$H(s) = \frac{64}{8s^2 + 48s + 64}$$

Sau đó tiến hành rời rạc hóa hàm truyền trên với tần số lấy mẫu 0.025s. Mỗi khi nút Set Speed được nhấn Speed_Timer của QT sẽ được chạy, mỗi khi timer tràn chương trình sẽ chạy đến hàm Send_Speed để gửi tín hiệu đặt đã được đi qua bộ lọc

$$H(z) = \frac{az + b}{z^2 + cz + d} = \frac{0.002379z + 0.002263}{z^2 - 1.856z + 0.8607}$$

```
void MainWindow::Send_Speed(){
  char checksum_Tx = 0;
  QByteArray txbuff;
  txbuff="$SPEED,";
  Out=a*setpoint+b*pre_setpoint-c*Pre_Out1-d*Pre_Out2;
  Pre_Out2=Pre_Out1;
  Pre_Out1=Out;
  pre_setpoint=setpoint;
  m_data.myfloat = Out;
  for(int k=0;k<4;k++)
  {
  txbuff[7+k]=m_data.mybyte[-k+3];
  checksum_Tx += m_data.mybyte[3-k];
  }
  txbuff[11]=checksum_Tx;
  txbuff[12]= '\r';
  txbuff[13]= '\n';
  SerialPort->write(txbuff,14);
  counter++;
 if((Out>=0.99*setpoint)&&(Out<=1.01*setpoint))
  {
     counter=0;
     Speed_Timer->stop();
  }
 if(counter >= 200)
 {
```

```
            counter=0;
            Speed_Timer->stop();
            m_data.myfloat = setpoint;
            for(int k=0;k<4;k++)
            {
            txbuff[7+k]=m_data.mybyte[-k+3];
            checksum_Tx += m_data.mybyte[3-k];
            }
            txbuff[11]=checksum_Tx;
            txbuff[12]= '\r';
            txbuff[13]= '\n';
            SerialPort->write(txbuff,14);
        }
```

Khi ngưng điều khiển động cơ, ta nhấn nút Exit. Nút Exit được liên kết với sự kiện clicked(), khi nút Exit được nhấn, chương trình sẽ chạy hàm on_btnExit_clicked() gửi giá trị đặt bằng 0 để vi xử lý ngừng động cơ.

```
    void MainWindow::on_btnExit_clicked()
    {
       setpoint = 0;
       char checksum_Tx = 0;
       QByteArray txbuff;
       txbuff="$SPEED,";
       m_data.myfloat = setpoint;
       for(int k=0;k<4;k++)
       {
       txbuff[7+k]=m_data.mybyte[-k+3];
```

```
            checksum_Tx += m_data.mybyte[3-k];
            }
            txbuff[11]=checksum_Tx;
            txbuff[12]= '\r';
            txbuff[13]= '\n';
            SerialPort->write(txbuff,14);
            this->close();
        }
```

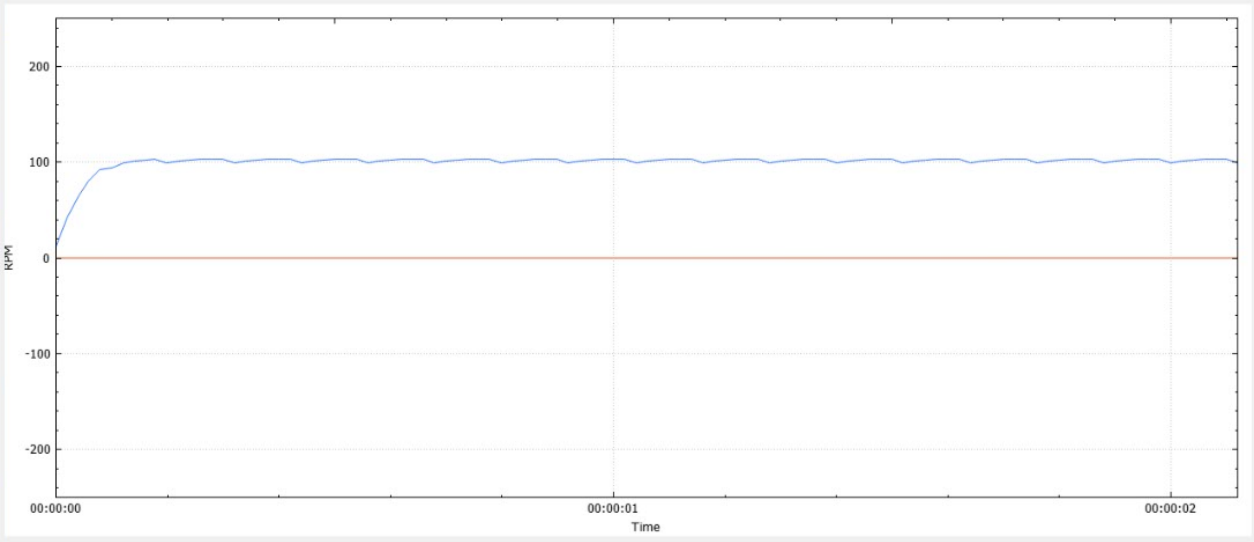
CHƯƠNG 3: KHẢO SÁT ĐIỀU KHIỂN VÀ KẾT QUẢ

1. LỰA CHỌN THÔNG SỐ BỘ ĐIỀU KHIỂN

Để nhận diện hàm truyền đối tượng động cơ, nhóm tiến hành cầm tín hiệu đặt là xung vuông sau đó quan sát đáp ứng tốc độ của động cơ. Với đáp ứng của động cơ khi tín hiệu đặt có dạng xung vuông như hình 3.1, ta có thể xấp xỉ hàm truyền đối tượng động cơ có dạng bậc nhất, rời rạc hóa hàm truyền động cơ với chu kì lấy mẫu $T = 0.025s$ ta được hàm truyền rời rạc

$$G(s) = \frac{1}{Ks + 1}$$

$$\rightarrow G(z) = \frac{\hat{a}}{z + \hat{b}}$$



Hình 3.1. Đáp ứng của động cơ khi tín hiệu đặt có dạng xung vuông

Sau khi nhận diện đối tượng, nhóm tiến hành thiết kế bộ điều khiển PI dựa vào các thông số hàm truyền nhận diện ở trên

Hàm tuyền bộ điều khiển PI

$$G_c(z) = K_p + \frac{K_I T}{2}\frac{z + 1}{z - 1}$$

Phương trình đặc trưng hệ kín

$$1 + G_c(z)G(z) = 0$$

$$\rightarrow z^2 + \left(\hat{b}K_p + 0.25\hat{b}K_I + \hat{a} - 1\right)z + \left(\hat{b}K_I - \hat{b}K_p - \hat{a}\right) = 0$$

Nhóm lựa chọn cặp cực quyết định mong muốn có $\xi = 1, \omega_n = 20$ từ đó ta có cặp cực phức mong muốn

$$z_{1,2} = 0.6065$$

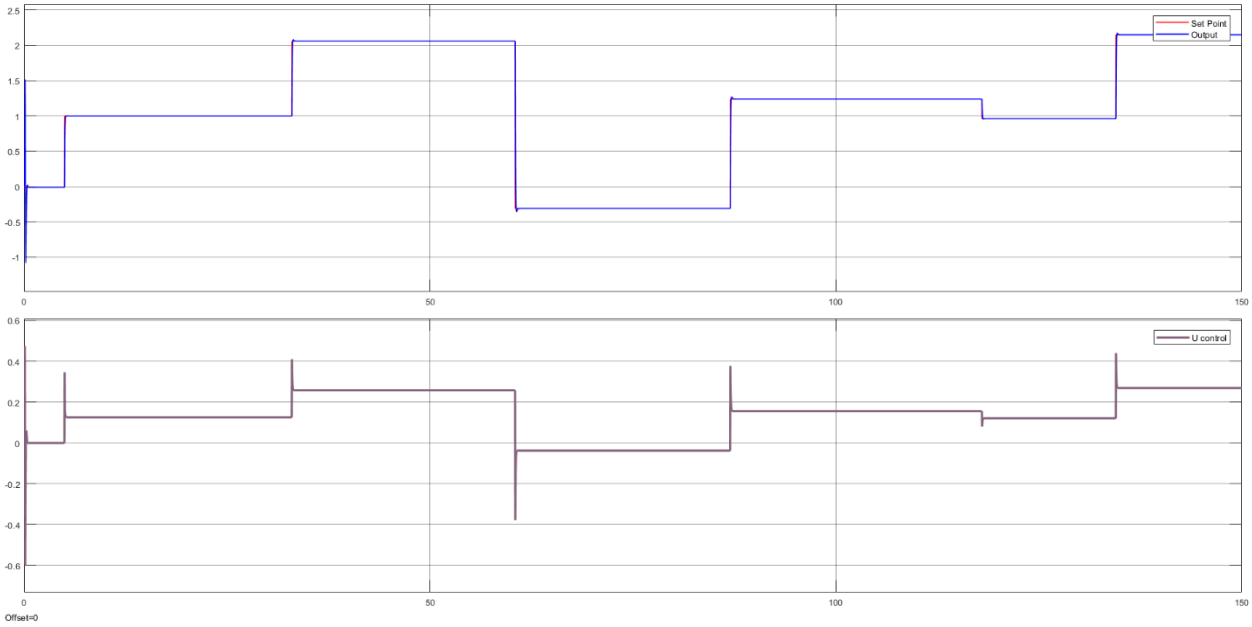
Từ cặp cực phức mong muốn ta có phương trình đặc trưng mong muốn

$$z^2 - 1.2131z + 0.3679 = 0$$

Đồng nhất thức với phương trình đặc trưng hệ kín ta có được hệ phương trình

$$\begin{cases} \hat{b}K_p + \dfrac{\hat{b}K_I T}{2} + \hat{a} - 1 = -1.2131 \\ \dfrac{\hat{b}K_I T}{2} - \hat{b}K_p - \hat{a} = 0.3679 \end{cases}$$

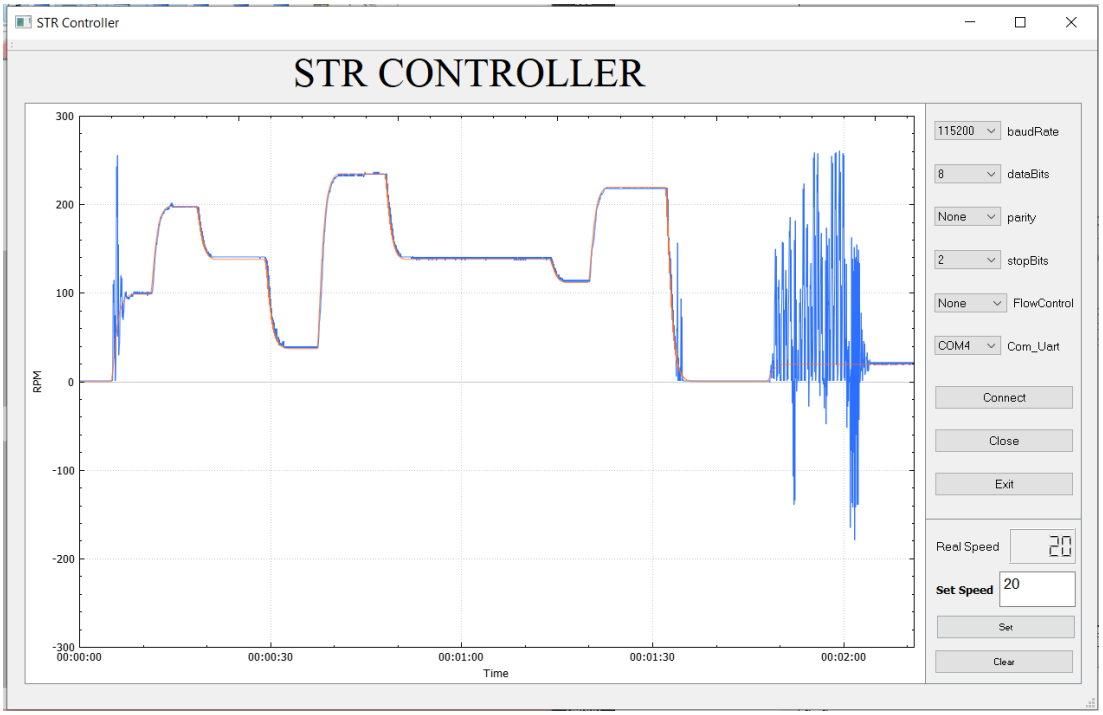
$$\rightarrow \begin{cases} K_I = \dfrac{6.1927}{\hat{b}} \\ K_p = \dfrac{-0.2905r - \hat{a}}{\hat{b}} \end{cases}$$



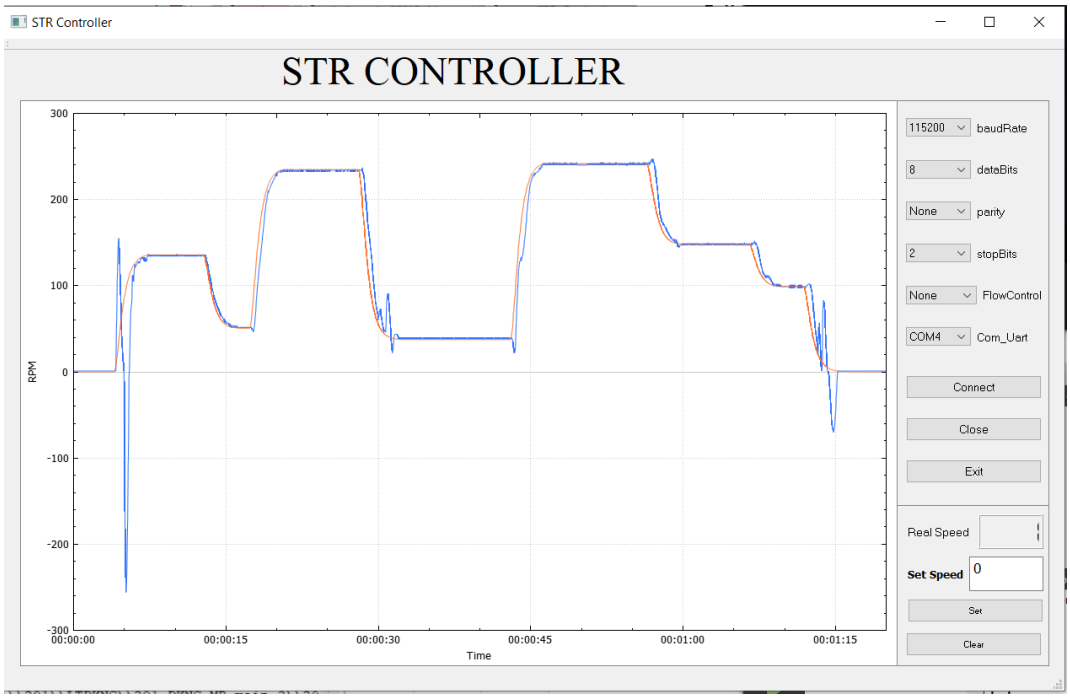
Hình 3.2. Kết quả mô phỏng hệ thống với thông số đã chọn

2. KHẢO SÁT ĐÁP ỨNG HỆ THỐNG

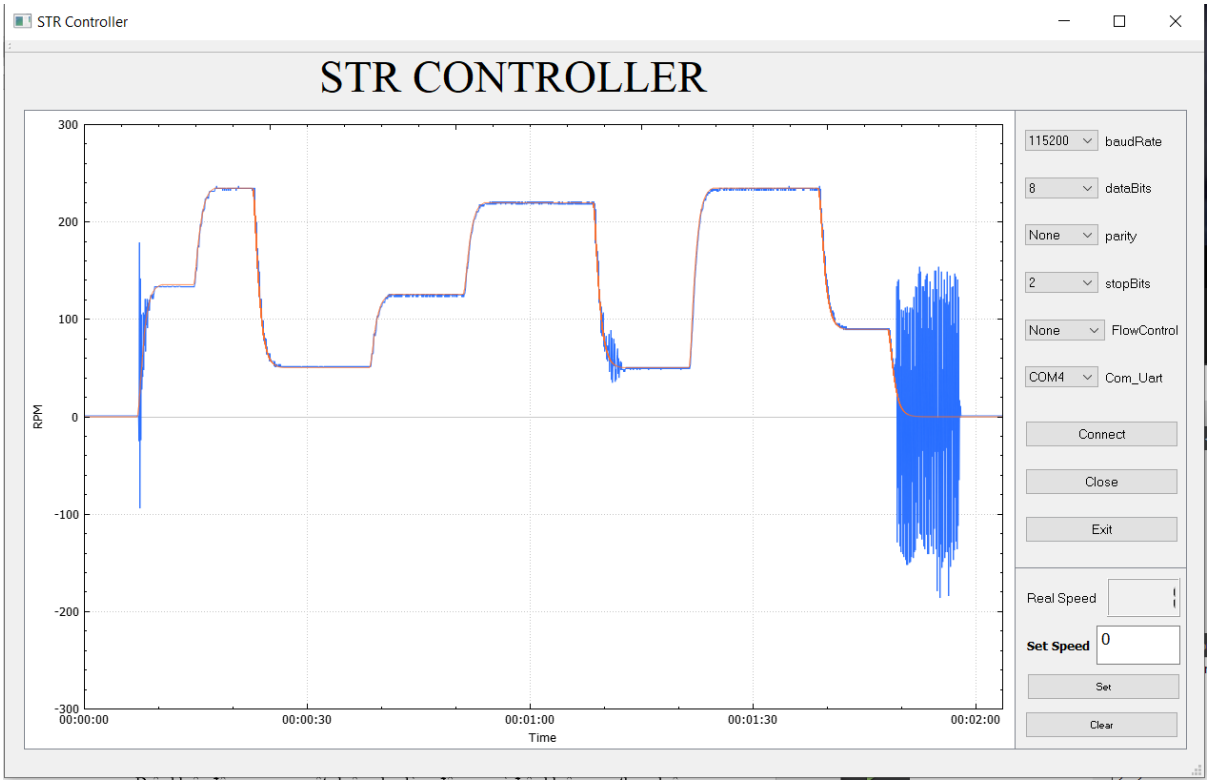
Nhóm tác giả tiến hành khảo sát đáp ứng của bộ điều khiển STR với các thông số ξ, ω_n khác nhau để khảo sát ảnh hưởng của ξ, ω_n lên đáp ứng hệ thống



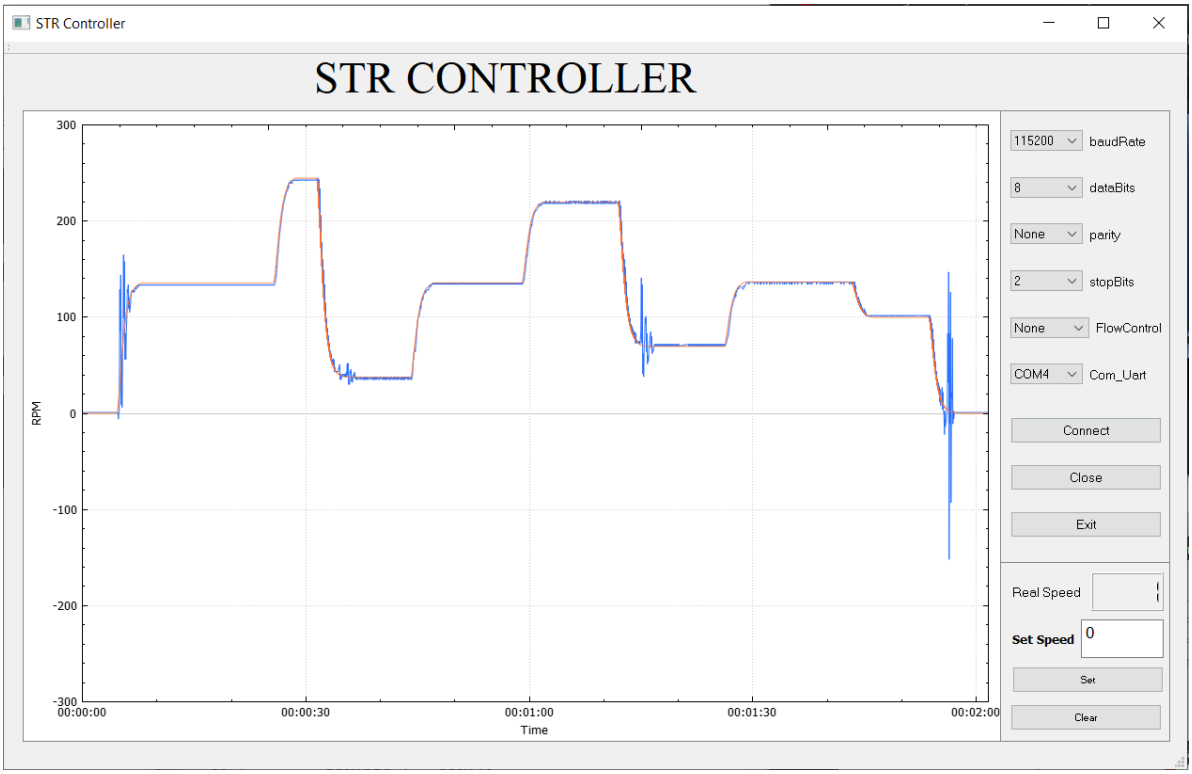
Hình 3.3. Đáp ứng của động cơ khi $\xi = 1,\ \omega_n = 20$



Hình 3.4. Đáp ứng của động cơ khi $\xi = 1,\ \omega_n = 10$

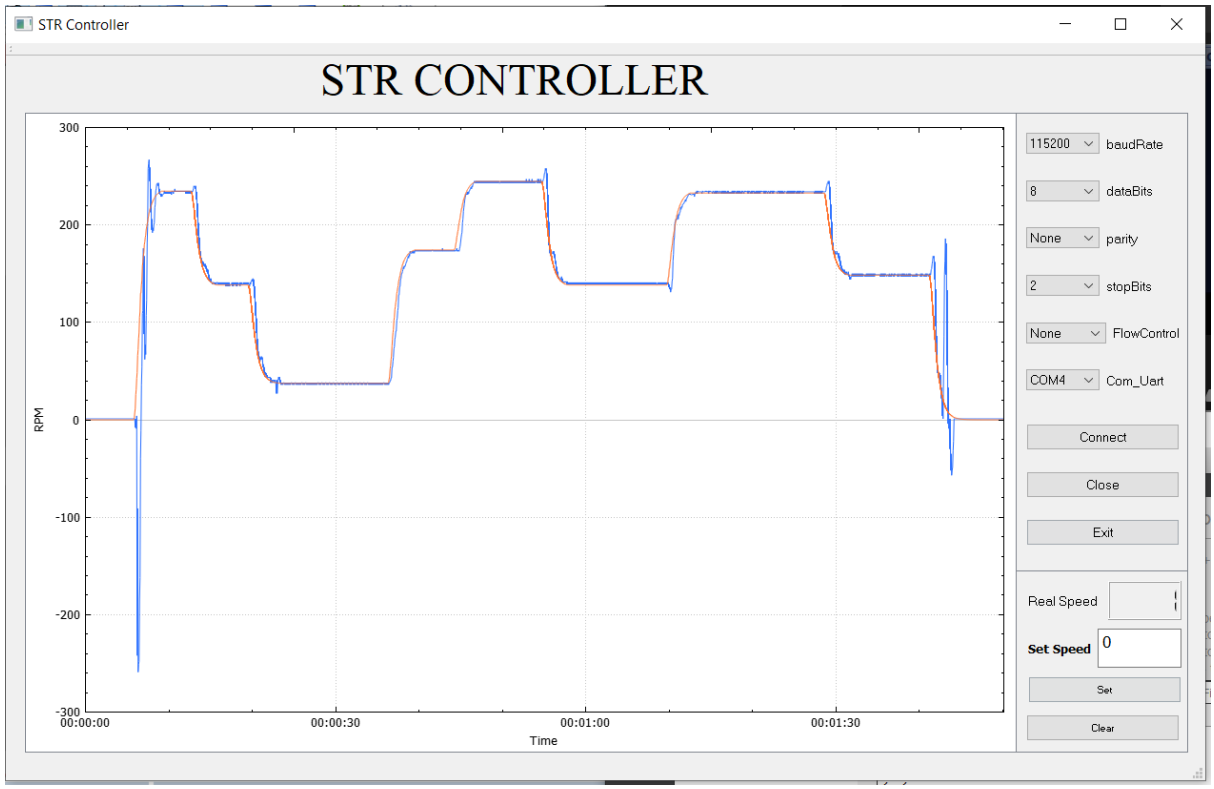


Hình 3.5. Đáp ứng của động cơ khi $\xi = 1,\ \omega_n = 30$



Hình 3.6. Đáp ứng của động cơ khi $\xi = 0.8,\ \omega_n = 20$

Hình 3.7. Đáp ứng của động cơ khi $\xi = 0.6,\ \omega_n = 20$

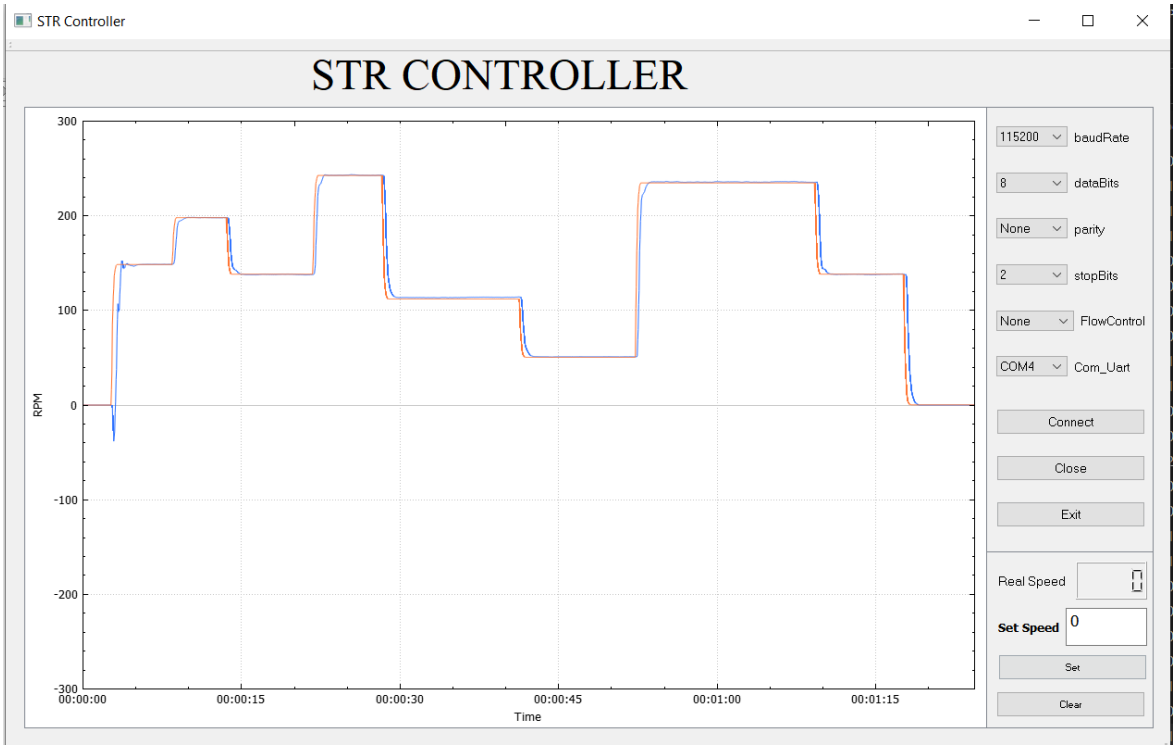


Hình 3.8. Đáp ứng của động cơ khi $\xi = 0.6,\ \omega_n = 10$

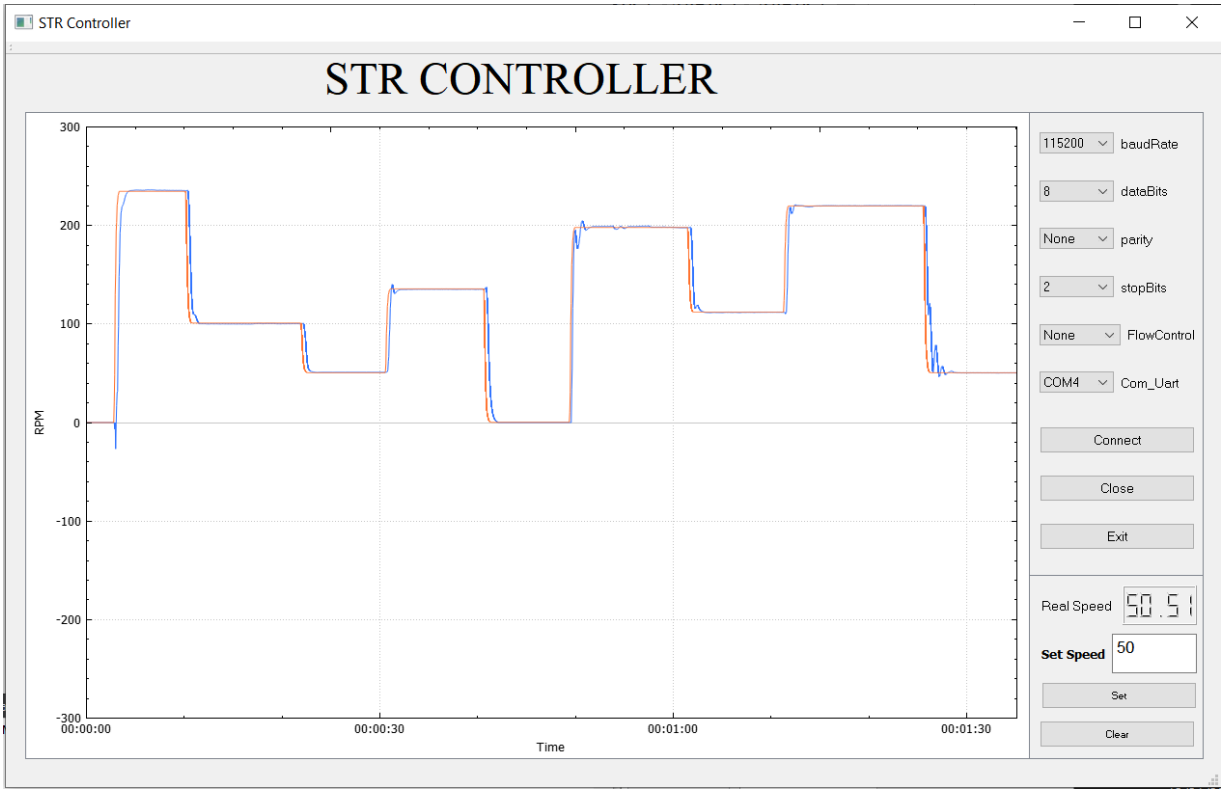
Nhận xét: khi ta giữ giá trị của $\xi = 1$ và thay đổi giá trị của ω_n, đáp ứng của động cơ gần như không có vọt lố, thời gian quá độ của đáp ứng cũng giảm dần khi tăng giá trị của ω_n, tuy nhiên khi $\omega_n = 30$, thì thời gian quá độ của động cơ quá nhỏ khiến bộ điều khiển không đáp ứng kịp với tốc độ thay đổi của giá trị đặt dẫn đến dễ xảy ra dao động. Khi nhóm giữ nguyên $\omega_n = 20$ và thay đổi ξ, đáp ứng của động cơ bắt đầu xảy ra vọt lố dù giá trị đặt nhóm gửi xuống vi xử lý có dạng hàm e mũ, ngoài ra đáp ứng của động cơ dễ xảy ra dao động. Ngoài ra với các thông số khảo sát, khi giảm tốc độ quay của động cơ xuống quá thấp thì động cơ khó có thể đáp ứng với tốc độ đặt này mà sẽ xảy ra dao động điều hòa và xác lập về giá trị đặt như hình 3.7.

Nhóm tác giả tiến hành khảo sát đáp ứng của bộ điều khiển STR khi:

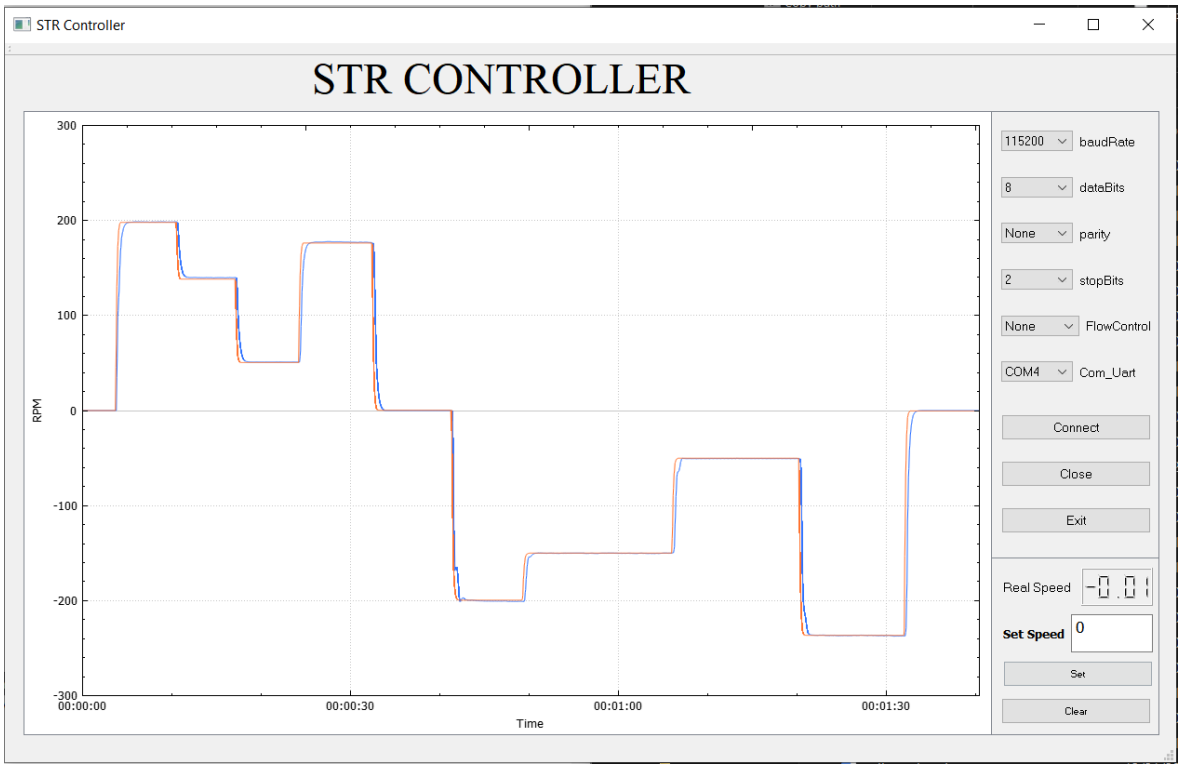
- Điều khiển động cơ quay 1 chiều sau đó dừng động cơ.
- Điều khiển động cơ quay một chiều, cho dừng động cơ và tiếp tục điều khiển theo chiều quay cũ.
- Điều khiển động cơ quay một chiều, cho dừng động cơ và điều khiển quay theo chiều ngược lại.
- Điều khiển động cơ quay hai chiều.



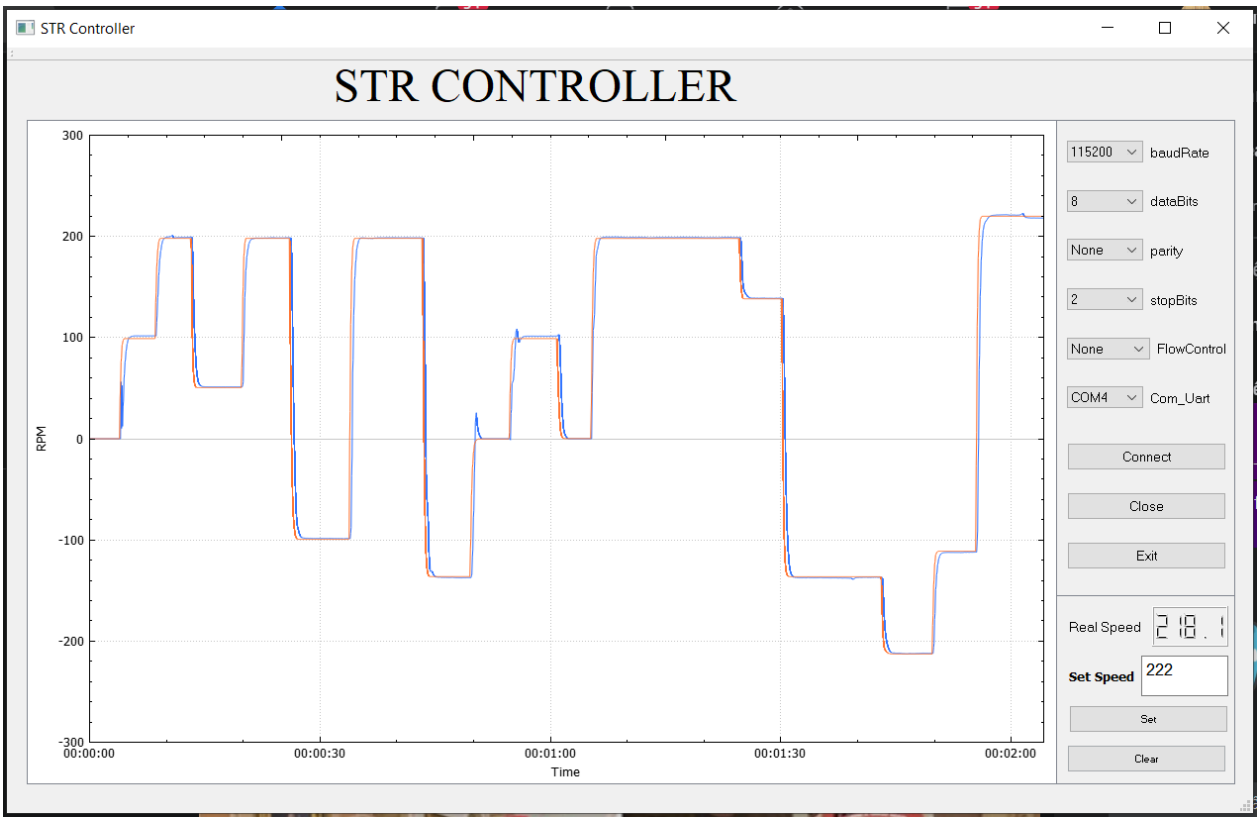
Hình 3.9. Điều khiển động cơ quay một chiều sau đó dừng động cơ



Hình 3.10. Điều khiển động cơ quay một chiều, dừng động cơ, điều khiển theo chiều cũ



Hình 3.11. Điều khiển động cơ quay một chiều, dừng động cơ, điều khiển theo chiều ngược lại



Hình 3.12. Điều khiển động cơ quay hai chiều

Nhận xét: Khi bắt đầu tiến hành chạy, đáp ứng của động cơ xuất hiện vọt lố và dao động tắt dần về giá trị đặt, sở dĩ có hiện tượng này là do bộ ước lượng trạng thái chưa ước lượng được chính xác thông số của hệ thống, sau khi thông số của hệ thống hội tụ, đáp ứng của hệ thống bám tương đối tốt lấy tín hiệu đặt và gần như không có vọt lố. Nếu ta dừng động cơ và tiếp tục điều khiển, bộ ước lượng cũng chưa ước lượng chính xác được thông số hệ thống nên cũng xảy ra hiện tượng như lúc ta đặt giá trị vận tốc lần đầu. Khi đổi chiều quay động cơ, hiện tượng này cũng xảy ra do bộ ước lượng chưa đáp ứng kịp với sự thay đổi đột ngột cũng như tín hiệu xung PWM dùng để điều khiển mạch lái mà nhóm sử dụng là hai channel khác nhau chứ không sử dụng channel đảo dẫn tới độ trễ khi thay đổi chiều quay.

TÀI LIỆU THAM KHẢO

Giáo trình môn học Lý Thuyết Điều Khiển Nâng Cao chương 4 - thầy Huỳnh Thái Hoàng.