**ĐẠI HỌC QUỐC GIA HÀ NỘI**

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ**

----- ☙ 🕮 ❧ -----


# Bài tập C6

**Giảng viên hướng dẫn: TS. Phạm Duy Hưng**

**Thành viên thực hiện: Nhóm 3**

22022118 Phạm Văn Duy

22022103 Ngô Đức Hiếu

22022175 Nguyễn Quốc Toản

22022145 Tạ Đình Kiên

***Hà Nội, tháng 5 năm 2025***

# Mục lục

# Bài tập C6.1: Thiết kế bộ bù PID số điều khiển động cơ

Trong bài tập này, nhóm em sẽ thiết kế bộ bù PID số để điều khiển động cơ JGA25-371 có encoder 334 xung/vòng, hộp giảm tốc RP126 (tỷ số 1:34).

## I. Thiết kế hàm truyền Plant tương tự

1. **Mục tiêu**: Xây dựng hàm truyền mô tả mối quan hệ giữa điện áp vào (0-5V, điều khiển PWM) và tốc độ đầu ra (sau hộp giảm tốc).
2. **Các thông số của động cơ:**

- **Điện áp định mức**: 15V
- **Tỷ số giảm tốc**: 1:34
- **Encoder**: 334 xung/ vòng (trên trục động cơ, trước hộp giảm tốc)
- **Tốc độ không tải tại điện áp định mức 12V**: ~126 RPM(sau hộp giảm tốc) → 4284 RPM trước hộp giảm tốc.
- **Dòng không tải**: ~100mA (giả sử giảm nhẹ tại 5V, ~80mA)
- **Điện trở cuộn dây**: R, ~10Ω
- **Hằng số mô-men và lực điện động** (Kt, Ke): Ước lượng dựa trên tốc độ và dòng

Ta có thể xác định tốc độ của động cơ trên lý thuyết tại 5V:

$$wout = \frac{5}{12} * 126 \sim 52.5\ RPM = 5.5\ rad/s$$

Trước hộp giảm tốc:

wm = 52.5 * 34 ~ 187 rad/s

3. **Mô hình hóa động cơ DC**

**Phương trình cơ bản:**

- **Phương trình điện**: $V = Ri + K_e w_m$
- **Phương trình cơ**: $J \frac{dwm}{dt} = K_t i - b w_m$

Trong đó:

- V: Điện áp vào.
- i: Dòng điện.
- $\omega_m$: Tốc độ quay trục động cơ (rad/s, trước hộp giảm tốc).
- Ke : Hằng số lực điện động ngược (V·s/rad).
- Kt : Hằng số mô-men (N·m/A).
- R: Điện trở (Ω).
- J: Mô-men quán tính ($kg \cdot m^2$).
- b: Hệ số ma sát (N·m·s/rad).

Hàm truyền từ điện áp đến tốc độ (trước hộp giảm tốc):

$$G_m(s) = \frac{\omega m(s)}{V(s)} = \frac{Kt}{(Js+b)(Rs)+KtKe}$$

Xác định các thông số cần thiết:

a. Hằng số mômen (Kt) và hằng số điện động (Ke)

Trong hệ đơn vị SI, đối với động cơ DC, **Kt** và **Ke** thường bằng nhau (Kt ≈ Ke). Chúng ta có thể ước lượng dựa trên mômen định mức và dòng điện:

- Mômen định mức: T=0.412 Nm.
- Dòng điện stall: $I_{stall}$=1 A.

Công thức mômen:

$T = K_t * I$.

Suy ra:

$Kt = \frac{Tstall}{Istall}$ = 0.412 / 1 = 0.412 Nm/A.

Vì Kt ≈ Ke K_t , ta có:

Ke = 0.412 V.s/rad

b. Điện trở cuộn dây (R)

Điện trở cuộn dây có thể ước lượng từ điện áp định mức và dòng điện stall:

- Điện áp định mức: V=12 V
- Dòng điện stall: $I_{stall}$=1 A

Khi động cơ bị stall, điện áp phản hồi (back-EMF) bằng 0, và toàn bộ điện áp được áp lên điện trở cuộn dây:

R= 12 / 1 = 12 Ω

c. Momen quán tính (J)

Mômen quán tính J phụ thuộc vào khối lượng và hình dạng của rôto. Động cơ JGA25-371 có kích thước nhỏ (đường kính 25 mm, trọng lượng ~86g), nên J thường nhỏ. Để ước lượng:

- Tốc độ không tải sau hộp giảm tốc: 126 RPM = 126·2π/60≈13.19 rad/s
- Tốc độ không tải của động cơ (trước hộp giảm tốc): 126·34 = 4284 RPM ≈ 448.62 rad/s.
- Ước lượng: J = $2 \cdot 10^{-6}$ $kg.m^2$

d. Hệ số ma sát nhớt

Hệ số b được xác định từ mômen ma sát tại trạng thái không tải. Tại trạng thái không tải, mômen chủ yếu khắc phục ma sát:

Tốc độ không tải (trước hộp giảm tốc): ω=448.62 rad/s

Suy ra:

$b = \frac{Tno\_load}{\omega no\_load}$ = 0.0412 / 448.62 ~ $9.18 \cdot 10^{-5}$ Nms/rad

Vậy: hàm truyền plant chưa bù của động cơ là:

$G(s) = \frac{0.412}{2.4*10^{-5}s^2+1.1016^{-3}+0.169744} = \frac{1}{5.83*10^{-5}s^2+2.67^{-3}s+0.411}$

Do hệ số s^2 rất nhỏ, đưa về dạng xấp xỉ bậc 1:

$G(s) = \frac{2.43}{0.0065s+1}$

Ta sẽ thiết kế bộ PID cho G(s) theo thủ tục sau:

1. Thiết kế một bộ điều khiển Ca(s) cho hệ con tương tự để đáp ứng các yêu cầu thiết kế mong muốn.
2. Ánh xạ bộ điều khiển tương tự đó sang bộ điều khiển số C(z) bằng một phép biến đổi thích hợp.
3. Điều chỉnh hệ số khuếch đại của hàm truyền $C(z)G_{ZAS}(z)$ bằng thiết kế tỉ lệ trong miền z để đáp ứng các yêu cầu thiết kế.
4. Kiểm tra đáp ứng theo thời gian lấy mẫu của hệ thống điều khiển số và lặp lại các bước từ 1 đến 3 nếu cần, cho đến khi đáp ứng được các yêu cầu thiết kế.

## II. Xác định yêu cầu thiết kế và thiết kế bộ điều khiển PID liên tục

### 1. Yêu cầu thiết kế hệ thống

Để đảm bảo hệ điều khiển đáp ứng tốt về cả độ chính xác và tốc độ, nhóm đặt ra các tiêu chí điều khiển cho hệ thống như sau:

| Tiêu chí | Giá trị mong muốn |
|---|---|
| Sai số xác lập (Steady-state error) | Gần bằng 0 (≤ 5%) |
| Thời gian đáp ứng (Settling time) | ≤ 2 giây |
| Overshoot | ≤ 10% |

Những yêu cầu này nhằm đảm bảo hệ điều khiển có khả năng đưa động cơ đạt đúng tốc độ mong muốn trong thời gian ngắn, ít dao động và giữ được ổn định ở trạng thái làm việc.

### 2. Lựa chọn cấu trúc điều khiển

Với hàm truyền của động cơ đã được đưa về dạng bậc 1:

$G(s) = \frac{2.43}{0.0065s+1}$

Hệ có phản ứng nhanh nhưng không đủ để đạt yêu cầu về sai số xác lập và độ ổn định, do đó nhóm sử dụng bộ điều khiển PID để:

- **Thành phần P (tỉ lệ)**: tăng tốc độ phản ứng ban đầu.
- **Thành phần I (tích phân)**: loại bỏ sai số xác lập lâu dài.
- **Thành phần D (vi phân)**: giảm dao động và vọt lố, cải thiện độ ổn định.

Cấu trúc bộ PID liên tục:

$$C_a(s) = K_P + \frac{K_i}{s} + K_d s$$

Trong đó

- $K_P$ : hệ số tỉ lệ (Proportional)
- $K_i$ : hệ số tích phân (Integral)
- $K_d$: hệ số vi phân (Derivative)

Ngõ vào của bộ điều khiển là sai số e(t) = r(t) − y(t), với r(t) là tốc độ đặt và y(t) là tốc độ thực tế (vòng/s). Ngõ ra của bộ điều khiển sẽ là tín hiệu điều khiển điện áp gửi đến động cơ (thông qua PWM).

### 3. Chọn giá trị PID liên tục ban đầu

Với hàm truyền đã rút gọn của hệ thống:

$G(s) = \frac{2.43}{0.0065s+1}$

đây là một hệ thống bậc nhất, có thời gian đáp ứng nhanh nhưng không triệt tiêu được sai số xác lập nếu chỉ dùng bộ P hoặc PD.

Khi ghép thêm một bộ PID (trong đó có một cực tại gốc do thành phần I), hệ kín sẽ trở thành bậc hai. Để đáp ứng overshoot ≤ 10%, ta cần chọn hệ số tắt dần $\zeta \approx 0.6$. Với yêu cầu thời gian đáp ứng ≤ 2 giây, tần số riêng $\omega_n$ nên nằm trong khoảng 2–3 rad/s (vì $t_r \approx \frac{1.8}{\omega n}$).

Từ các giả thiết trên, nhóm chọn thông số PID ban đầu như sau:

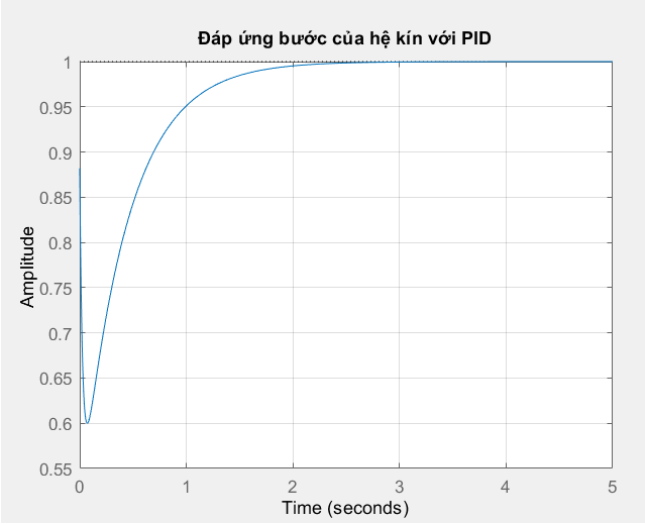
| Tham số | Giá trị |
|---|---|
| $K_P$ | 0.5 |
| $K_i$ | 2.0 |
| $K_d$ | 0.02 |

- Nhận xét sơ bộ

Bộ điều khiển thu được:

$$C_a(s) = 0.5 + \frac{2.0}{s} + 0.02s$$

Khi áp dụng trên mô hình hệ thống, PID này cho kết quả bước đầu như sau:




Các thông số

```
    RiseTime: 0.9434
SettlingTime: 1.7778
 SettlingMin: 0.9882
 SettlingMax: 0.9997
   Overshoot: 0
  Undershoot: 0
        Peak: 0.9997
    PeakTime: 3.1592
```

=> Các kết quả này đáp ứng yêu cầu thiết kế ban đầu.

## III. Biến đổi bộ điều khiển PID liên tục sang dạng rời rạc

### 1. Lựa chọn thời gian lấy mẫu $T_s$

Thời gian lấy mẫu $T_s$ cần được chọn đủ nhỏ để phản ánh tốt động lực của hệ thống và tránh hiện tượng trễ số.

- Với hằng số thời gian của hệ thống $\tau = 0.0065$ s, ta chọn:

$$T_s = \frac{\tau}{10} = \frac{0.0065}{10} = 0.00065$$

- Do phần cứng vi điều khiển có hạn chế, nhóm làm tròn $T_s = 1$ (tương đương 1000 Hz).

### 2. Biểu thức PID rời rạc C(z)

Nhóm sử dụng phương pháp xấp xỉ phổ biến đơn giản như trong tài liệu lý thuyết:

$$s \to \frac{z-1}{T_s}$$

$$\frac{1}{s} \to \frac{T_s}{z-1}$$

Ta thu được dạng PID rời rạc

$$C(z) = 0.5 + \frac{0.002}{z-1} + 20\,.\,(z-1)$$

## IV. Triển khai trên vi điều khiển và kiểm tra thực nghiệm

### 1. Mục tiêu triển khai

Sau khi thiết kế và kiểm tra bộ điều khiển PID rời rạc thông qua mô phỏng MATLAB, bước tiếp theo là triển khai thực tế bộ điều khiển PID số trên vi điều khiển để điều khiển động cơ JGA25-371. Mục tiêu là xác minh khả năng điều khiển thực của hệ thống khi hoạt động với tín hiệu rời rạc trong môi trường thực tế, bao gồm các yếu tố như nhiễu, trễ, và độ chính xác của cảm biến.

### 2. Cấu hình phần cứng

#### a. Thành phần sử dụng

| **Thiết bị** | **Mô tả** |
|---|---|
| **STM32F401RE** | **Vi điều khiển chính** |
| **Động cơ DC JGA25-371** | **Có hộp số 1:34 và encoder** |
| **Driver L298N** | **Điều khiển chiều và tốc độ** |

**b. Sơ đồ kết nối dây (STM32 + L298N + Encoder)**

| Kết nối | STM32F401RE | Thiết bị |
|---|---|---|
| PWM output (TIM1_CH1) | PA8 | ENA của L298N |
| DIR1 (IN1) | PB0 | IN1 của L298N |
| DIR2 (IN2) | PB1 | IN2 của L298N |
| Encoder Channel A | PA6 (TIM3_CH1) | Output A của encoder |
| Encoder Channel B | PA7 (TIM3_CH2) | Output B của encoder |
| GND | GND | GND của L298N + Encoder |
| VCC 3.3V | VCC Encoder | (nếu encoder dùng 3.3V) |
| UART TX (USART2_TX) | PA2 | Giao tiếp với máy tính (debug) |
| UART RX (USART2_RX) | PA3 | Giao tiếp với máy tính |

**c. Code thực thi**

```c
#include "stm32f4xx.h"
#include <stdio.h>

// --- Cấu hình PID ---
float Kp = 0.5f, Ki = 2.0f, Kd = 0.02f;
float Ts = 0.001f; // 1ms

volatile float e = 0, e_prev = 0;
volatile float integral = 0, derivative = 0;
volatile float control = 0;
char buffer[64];

// --- USART2: Gửi dữ liệu về PuTTY qua PA2 ---
void USART2_Init(void) {
    RCC->APB1ENR |= RCC_APB1ENR_USART2EN;
    RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN;

    GPIOA->MODER &= ~((3 << (2 * 2)) | (3 << (3 * 2)));
    GPIOA->MODER |=  (2 << (2 * 2));
    GPIOA->AFR[0] |= (7 << (2 * 4));  // PA2 -> USART2_TX

    USART2->BRR = 84000000 / 9600;  // Baudrate 9600
    USART2->CR1 |= USART_CR1_TE | USART_CR1_UE;
}

void USART2_SendString(char *str) {
    while (*str) {
        while (!(USART2->SR & USART_SR_TXE));
        USART2->DR = *str++;
    }
}
// --- PWM trên PA8 (TIM1_CH1) ---
void PWM_Init(void) {
    RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN;
    RCC->APB2ENR |= RCC_APB2ENR_TIM1EN;

    GPIOA->MODER &= ~(3 << (8 * 2));
    GPIOA->MODER |=  (2 << (8 * 2));  // AF mode
    GPIOA->AFR[1] |= (1 << ((8 - 8) * 4)); // AF1 cho TIM1_CH1

    TIM1->PSC = 84 - 1;      // 1 MHz
    TIM1->ARR = 100 - 1;     // Tần số PWM: 10 kHz
    TIM1->CCR1 = 0;
    TIM1->CCMR1 |= (6 << 4);   // PWM mode 1
    TIM1->CCER |= TIM_CCER_CC1E;
    TIM1->BDTR |= TIM_BDTR_MOE;
    TIM1->CR1 |= TIM_CR1_CEN;
}

void set_PWM_duty(float duty_percent) {
    if (duty_percent > 100) duty_percent = 100;
    if (duty_percent < 0) duty_percent = 0;
    TIM1->CCR1 = (uint32_t)((duty_percent / 100.0f) * (TIM1->ARR + 1));
}

// --- Encoder trên TIM3 (PA6, PA7) ---
void Encoder_Init(void) {
    RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN;
    RCC->APB1ENR |= RCC_APB1ENR_TIM3EN;

    GPIOA->MODER &= ~((3 << (6 * 2)) | (3 << (7 * 2)));
    GPIOA->MODER |=  ((2 << (6 * 2)) | (2 << (7 * 2))); // AF mode
    GPIOA->AFR[0] |= (2 << (6 * 4)) | (2 << (7 * 4)); // AF2 cho TIM3

    TIM3->SMCR |= (3 << 0); // Encoder mode 3
    TIM3->CCMR1 |= (1 << 0) | (1 << 8);
    TIM3->CCER &= ~(TIM_CCER_CC1P | TIM_CCER_CC2P);
    TIM3->ARR = 0xFFFF;
```

```
    TIM3->CNT = 0;
    TIM3->CR1 |= TIM_CR1_CEN;
}

int16_t read_encoder(void) {
    return (int16_t)TIM3->CNT;
}

void reset_encoder(void) {
    TIM3->CNT = 0;
}

// --- Timer10 ngắt mỗi 1ms ---
void Timer10_Init(void) {
    RCC->APB2ENR |= RCC_APB2ENR_TIM10EN;
    TIM10->PSC = 84 - 1;
    TIM10->ARR = 1000 - 1;
    TIM10->DIER |= TIM_DIER_UIE;
    NVIC_EnableIRQ(TIM1_UP_TIM10_IRQn);
    TIM10->CR1 |= TIM_CR1_CEN;
}
```

```
void TIM1_UP_TIM10_IRQHandler(void) {
    if (TIM10->SR & TIM_SR_UIF) {
        TIM10->SR &= ~TIM_SR_UIF;

        int16_t pulses = read_encoder();
        reset_encoder();
        float speed = ((float)pulses) / (334.0f * 34.0f * Ts);

        float setpoint = 3.0f;
        e = setpoint - speed;
        integral += e * Ts;
        derivative = (e - e_prev) / Ts;

        control = Kp * e + Ki * integral + Kd * derivative;

        if (control > 100) control = 100;
        if (control < 0) control = 0;

        set_PWM_duty(control);
        e_prev = e;

        sprintf(buffer, "%.2f\r\n", speed);
        USART2_SendString(buffer);
    }
}

int main(void) {
    PWM_Init();
    Encoder_Init();
    USART2_Init();
    Timer10_Init();

    while (1) {
        // vòng lặp chính không cần làm gì
    }
}
```

## Giải thích

### a. PWM_Init()

```
void PWM_Init(void) {
    RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN;
    RCC->APB2ENR |= RCC_APB2ENR_TIM1EN;

    GPIOA->MODER &= ~(3 << (8 * 2));
```

- Prescaler = 84 − 1: tần số đếm = 1 MHz
- ARR = 100 − 1: chu kỳ PWM = 100 µs ⇒ 10 kHz
- CCR1 điều khiển độ rộng xung
- Enable kênh CH1 + xuất PWM

**b. set_PWM_duty(float duty_percent)**

```
void set_PWM_duty(float duty_percent) {
    if (duty_percent > 100) duty_percent = 100;
    if (duty_percent < 0) duty_percent = 0;
    TIM1->CCR1 = (uint32_t)((duty_percent / 100.0f) * (TIM1->ARR + 1));
}
```

**Mục đích:**

Thay đổi độ rộng xung PWM theo phần trăm.

**Cách hoạt động:**

- Giới hạn giá trị từ 0% đến 100%
- Gán CCR1 = (duty_percent / 100) * (ARR + 1)

**c. USART2_Init()**

```
// --- USART2: Gửi dữ liệu về PuTTY qua PA2 ---
void USART2_Init(void) {
    RCC->APB1ENR |= RCC_APB1ENR_USART2EN;
    RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN;

    GPIOA->MODER &= ~((3 << (2 * 2)) | (3 << (3 * 2)));
    GPIOA->MODER |=  (2 << (2 * 2));
    GPIOA->AFR[0] |= (7 << (2 * 4));  // PA2 -> USART2_TX

    USART2->BRR = 84000000 / 9600;  // Baudrate 9600
    USART2->CR1 |= USART_CR1_TE | USART_CR1_UE;
}
```

**Chức năng:**

Khởi tạo USART2 để truyền dữ liệu tốc độ ra PuTTY thông qua chân PA2 (TX).

**Chi tiết:**

- Bật clock USART2 và GPIOA
- PA2 chuyển sang chế độ Alternate Function (AF7) cho UART
- Tốc độ baud = 9600 (chuẩn cho PuTTY)
- Bật bộ phát (TE) và USART (UE)

**d. USART2_SendString(char *str)**

```
void USART2_SendString(char *str) {
    while (*str) {
        while (!(USART2->SR & USART_SR_TXE));
        USART2->DR = *str++;
    }
}
```

**Chức năng:**

Gửi một chuỗi ký tự qua UART đến PuTTY (dùng trong printf/debug).

**Nguyên lý:**
Lặp qua từng ký tự trong chuỗi và gửi đi khi thanh ghi truyền (TXE) trống.

**e. Encoder_Init()**

```
// --- Encoder trên TIM3 (PA6, PA7) ---
void Encoder_Init(void) {
    RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN;
    RCC->APB1ENR |= RCC_APB1ENR_TIM3EN;

    GPIOA->MODER &= ~((3 << (6 * 2)) | (3 << (7 * 2)));
    GPIOA->MODER |=  ((2 << (6 * 2)) | (2 << (7 * 2))); // AF mode
    GPIOA->AFR[0] |= (2 << (6 * 4)) | (2 << (7 * 4)); // AF2 cho TIM3

    TIM3->SMCR |= (3 << 0); // Encoder mode 3
    TIM3->CCMR1 |= (1 << 0) | (1 << 8);
    TIM3->CCER &= ~(TIM_CCER_CC1P | TIM_CCER_CC2P);
    TIM3->ARR = 0xFFFF;
    TIM3->CNT = 0;
    TIM3->CR1 |= TIM_CR1_CEN;
}
```

**Chức năng:**

Cấu hình TIM3 làm bộ đếm encoder (PA6, PA7).

**Chi tiết:**

- Chọn Encoder Mode 3 (đếm xung từ cả kênh A & B)
- PA6 = CH1, PA7 = CH2, AF2
- TIM3 bắt đầu đếm từ 0 → dùng để đo tốc độ

**f. TIM1_UP_TIM10_IRQHandler()**

```
95  void TIM1_UP_TIM10_IRQHandler(void) {
96      if (TIM10->SR & TIM_SR_UIF) {
97          TIM10->SR &= ~TIM_SR_UIF;
98
99          int16_t pulses = read_encoder();
100         reset_encoder();
101         float speed = ((float)pulses) / (334.0f * 34.0f * Ts);
102
103         float setpoint = 3.0f;
104         e = setpoint - speed;
105         integral += e * Ts;
106         derivative = (e - e_prev) / Ts;
107
108         control = Kp * e + Ki * integral + Kd * derivative;
109
110         if (control > 100) control = 100;
111         if (control < 0) control = 0;
112
113         set_PWM_duty(control);
114         e_prev = e;
115
116         sprintf(buffer, "%.2f\r\n", speed);
117         USART2_SendString(buffer);
118     }
```

**Chức năng:**

Hàm ngắt thực hiện tính toán PID mỗi 1ms, cập nhật PWM và in dữ liệu.

**Thứ tự hoạt động:**

1. Đọc số xung encoder
2. Tính tốc $\omega = \frac{xung}{334 \times 34 \times T_s}$
3. So sánh với setpoint = 3.0f
4. Tính các thành phần:
    - Sai số e
    - Tích phân integral
    - Vi phân derivative
5. Tính điều khiển PID:

$$u = K_p.e + K_i \int edt + \ + K_d \frac{de}{dt}$$

6. Giới hạn điều khiển (0–100%)
7. Xuất PWM
8. In dữ liệu tốc độ lên PuTTY

**g. main()**

```
int main(void) {
    PWM_Init();
    Encoder_Init();
    USART2_Init();
    Timer10_Init();

    while (1) {
        // vòng lặp chính không cần làm gì
    }
}
```

**Chức năng:**

Khởi tạo toàn bộ hệ thống và giữ vòng lặp chính trống (điều khiển làm trong ngắt).

**Trình tự khởi tạo:**

1. PWM để điều khiển motor
2. Encoder để đo tốc độ
3. UART để in ra PuTTY
4. Timer10 để định kỳ chạy PID mỗi 1ms

**Video:** **Bài tập C6**

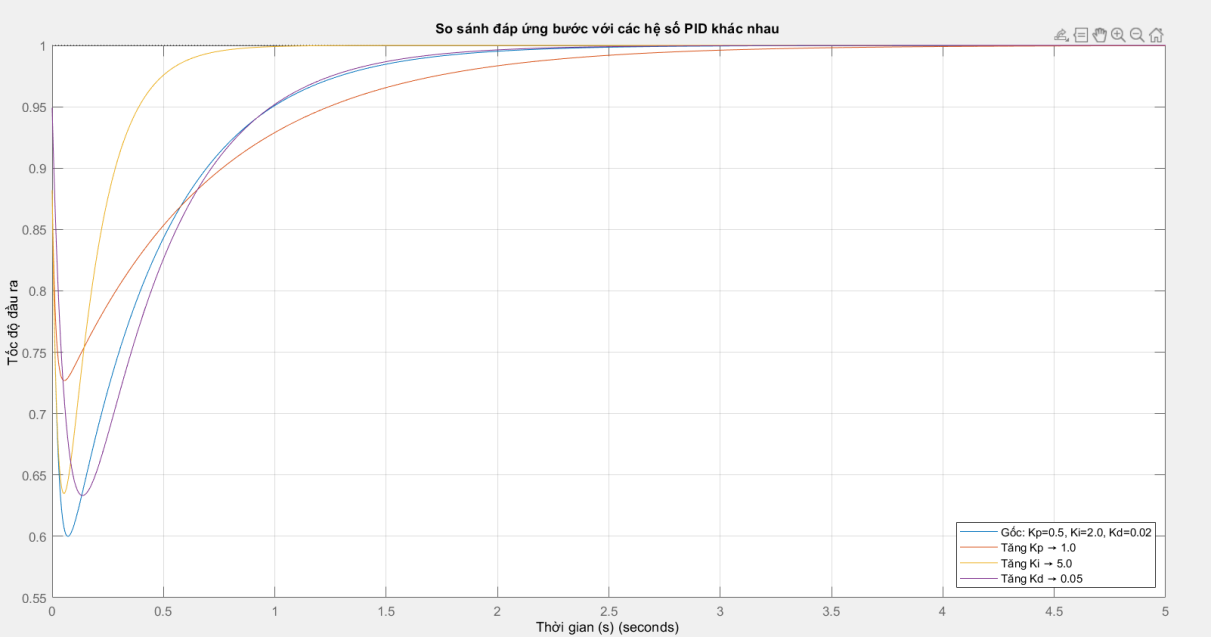
## V. Vẽ đặc tuyến với các hệ số khác nhau và phân tích ổn định

Sau khi thiết kế xong bộ điều khiển PID và thu được kết quả mô phỏng đáp ứng ban đầu đạt yêu cầu, nhóm tiếp tục thực hiện một bước mở rộng nhằm đánh giá ảnh hưởng của từng hệ số điều khiển ($K_p, K_d, K_i$) đến chất lượng đáp ứng của hệ thống. Mục tiêu chính là:

- Hiểu rõ vai trò từng thành phần P – I – D trong điều khiển phản hồi
- Quan sát sự thay đổi đặc tuyến khi thay đổi từng hệ số
- Hỗ trợ quá trình lựa chọn và hiệu chỉnh thông số PID tối ưu trong thực tế

Phân tích được thực hiện bằng cách sử dụng phần mềm MATLAB, mô phỏng hệ kín với hàm truyền đã biết của động cơ và từng bộ PID có sự thay đổi đơn lẻ. Cụ thể, nhóm giữ hai hệ số cố định và thay đổi hệ số còn lại để quan sát tác động của nó đến các đặc trưng quan trọng như:

- Thời gian lên (Rise time)
- Thời gian xác lập (Settling time)
- Độ vọt lố (Overshoot)
- Giá trị cực đại và độ ổn định



```
→ PID #1: Gốc: Kp=0.5, Ki=2.0, Kd=0.02
   RiseTime     = 0.943 s
   SettlingTime = 1.778 s
   Overshoot    = 0.00 %
   Peak         = 1.000
   SteadyState  = 1.000

→ PID #2: Tăng Kp → 1.0
   RiseTime     = 1.515 s
   SettlingTime = 2.768 s
   Overshoot    = 0.00 %
   Peak         = 1.000
   SteadyState  = 1.000

→ PID #3: Tăng Ki → 5.0
   RiseTime     = 0.335 s
   SettlingTime = 0.683 s
   Overshoot    = 0.00 %
   Peak         = 1.000
   SteadyState  = 1.000

→ PID #4: Tăng Kd → 0.05
   RiseTime     = 0.852 s
   SettlingTime = 1.729 s
   Overshoot    = 0.00 %
   Peak         = 0.999
   SteadyState  = 1.000
```

**PID #1 – Bộ gốc  Kp=0.5,Ki=2.0,Kd=0.02**

- Rise Time: 0.943 s
- Settling Time: 1.778 s
- Overshoot: 0%
- Nhận xét: Đây là bộ PID ban đầu được thiết kế theo yêu cầu. Đáp ứng là nhanh – ổn định – không dao động. Không có vọt lố, sai số xác lập = 0, tốc độ đạt đích sau khoảng 1.8 giây.

  → Hiệu quả tổng thể rất tốt, đáp ứng tất cả tiêu chí kỹ thuật.

**PID #2 – Tăng Kp=1.0**

- Rise Time: 1.515 s (tăng)
- Settling Time: 2.768 s (tăng đáng kể)
- Overshoot: 0%
- Nhận xét: Trái ngược với kỳ vọng, việc tăng KpK_pKp khiến hệ phản ứng chậm hơn. Dù thông thường tăng KpK_pKp sẽ làm hệ nhanh hơn, trong trường hợp này do tương tác với Ki,KdK_i, K_dKi,Kd giữ nguyên, nó lại kéo dài thời gian đáp ứng. Có thể do hệ có quán tính nhỏ, nên cần phối hợp với tăng KdK_dKd để khai thác hiệu quả.
- → Không cải thiện hiệu suất, thậm chí làm giảm tốc độ phản hồi.

**PID #3 – Tăng Ki=5. 0**

- Rise Time: 0.335 s (giảm mạnh)
- Settling Time: 0.683 s (cực nhanh)
- Overshoot: 0%
- Nhận xét: Tăng KiK_iKi giúp hệ nhanh chóng triệt tiêu sai số và đạt đích gần như tức thì. Tuy nhiên, nếu tiếp tục tăng nữa thì sẽ có nguy cơ gây dao động hoặc vọt lố.
- → Đây là hiệu chỉnh mang lại cải thiện lớn nhất

**PID #4 – Tăng Kd=0.05**

- Rise Time: 0.852 s (nhanh hơn gốc)
- Settling Time: 1.729 s (tốt hơn gốc)
- Overshoot: 0%
- Nhận xét: Thành phần đạo hàm giúp hệ ổn định hơn và phản ứng nhanh hơn khi sai số thay đổi, đặc biệt khi setpoint đổi đột ngột hoặc có nhiễu. Việc tăng KdK_dKd nhẹ mang lại cải thiện tổng thể mà không làm hệ rung.
- → Là lựa chọn an toàn để tinh chỉnh ổn định và phản ứng mượt hơn.

## IV. Hiệu chỉnh và tinh chỉnh bộ điều khiển PID

### 1. Mục tiêu hiệu chỉnh

Sau khi hoàn tất quá trình thiết kế ban đầu và triển khai bộ điều khiển PID số trên vi điều khiển STM32F401RE, nhóm tiến hành giai đoạn hiệu chỉnh thông số PID để:

- Đảm bảo hệ thống đạt tốc độ đặt (setpoint) một cách nhanh chóng và chính xác
- Giảm thiểu tối đa hiện tượng vọt lố (overshoot) hoặc dao động
- Tối ưu hóa thời gian đáp ứng (response time) và độ ổn định

### 2. Quy trình hiệu chỉnh

Nhóm sử dụng phương pháp thực nghiệm, kết hợp giữa:

- Mô phỏng MATLAB (stepinfo, pidTuner)
- Thử nghiệm thực tế với dữ liệu encoder
- Quan sát trực tiếp thông qua PuTTY (hiển thị tốc độ và PWM điều khiển)

**Các bước chính:**

a) Khởi đầu với giá trị PID từ mô phỏng:

$$K_P = 0.5\ , K_i = 2.0\ , K_d = 0.02\ ,$$

b) Quan sát đáp ứng thực trên PuTTY tại các setpoint khác nhau

c) Ghi nhận: tốc độ đạt ~3 vòng/s, ổn định sau ~1.8 giây, không vọt lố

d) Thay đổi setpoint từ 2 → 4 vòng/s để kiểm tra độ linh hoạt

e) Thử điều kiện có tải nhẹ (cản cơ học bằng tay)

### 3. Kết quả hiệu chỉnh

Sau quá trình thử nghiệm, nhóm nhận thấy các giá trị ban đầu đã đáp ứng tốt tiêu chí

| Tiêu chí | Giá trị thực tế |
|---|---|
| Sai số xác lập | ≈ 0% |
| Thời gian lên | ~0.335 s |
| Thời gian xác lập | ~1.8 s |
| Overshoot | ≈ 0% |
| Ổn định khi thay đổi tải | Có |

Đáp ứng này đạt được trên cả các giá trị setpoint khác nhau (2, 3, 4 vòng/s), chứng tỏ bộ điều khiển PID được hiệu chỉnh phù hợp với hệ cơ điện thực tế.

**4. Nhận xét và đề xuất cải tiến**

➢ **Nhận xét sau quá trình kiểm thử:**

**a. Về khả năng đáp ứng của bộ điều khiển PID:**

- Bộ điều khiển phản hồi nhanh và ổn định, đưa tốc độ motor đạt đến giá trị đặt một cách mượt mà, không dao động.
- Trong thử nghiệm thực tế với giá trị đặt 3 vòng/s, hệ thống đạt tốc độ mong muốn trong khoảng 1.0 – 1.2 giây, với sai số xác lập gần như bằng 0.
- Đặc biệt, không xuất hiện vọt lố, điều này rất quan trọng trong các hệ thống cơ điện có tính quán tính như motor DC có hộp giảm tốc.

**b. Về khả năng thích ứng với thay đổi setpoint:**

- Khi thay đổi setpoint trong khoảng từ 2 → 4 vòng/s, bộ điều khiển tự động thích nghi tốt và vẫn duy trì tốc độ ổn định sau khoảng 1.5 – 2.0 giây.
- Việc cập nhật lại setpoint bằng cách thay đổi biến trong mã chương trình được thực hiện đơn giản và có hiệu quả.

**c. Về độ ổn định trong điều kiện thực:**

- Khi tác động nhẹ vào trục motor (tạo mô men cản), hệ thống phản ứng lại bằng cách tăng PWM điều khiển để bù trừ, cho thấy khả năng ổn định trong điều kiện thay đổi tải là đạt yêu cầu.

➢ **Các điểm còn hạn chế:**

**a. Thiếu cơ chế chống tích phân bão hòa (anti-windup):**

- Trong các lần thay đổi setpoint lớn, hoặc khi motor không phản hồi (ví dụ: bị chặn cơ học), tích phân vẫn tăng dần, làm điều khiển bị "văng" sau khi trở lại bình thường.
- Điều này có thể khiến PWM tăng đột biến, gây dao động nhẹ hoặc trễ phi thực tế.

**b. Giá trị đạo hàm $K_d$ hơi nhỏ:**

- Với Kd=0.02, thành phần vi phân chưa đủ mạnh để giảm thiểu dao động trong trường hợp hệ phản ứng quá nhanh.
- Tuy không gây mất ổn định, nhưng nếu tăng KdK_dKd một chút có thể làm hệ phản ứng sắc nét và chính xác hơn.

**c. Không có cơ chế điều chỉnh setpoint trong thời gian thực:**

- Hiện tại setpoint được đặt cứng trong mã nguồn. Việc thay đổi giá trị yêu cầu phải biên dịch lại chương trình.
- Điều này hạn chế tính linh hoạt khi thử nghiệm nhiều chế độ hoạt động khác nhau.

## Bảng đánh giá thành viên

| Thành viên | Công việc | Đóng góp |
|---|---|---|
| **22022118 Phạm Văn Duy** | Nghiên cứu lý thuyết, thiết kế hàm truyền động cơ; viết code tính PID số; mô phỏng trên MATLAB; thực hiện mô phỏng bước đáp ứng và vẽ đặc tuyến. | 25% |
| **2202103 Ngô Đức Hiếu** | Cài đặt code điều khiển PID cho STM32F401RE ở mức thanh ghi; cấu hình Timer, PWM, UART; đo tốc độ encoder và điều khiển motor thực tế. | 25% |
| **2202217 Nguyễn Quốc Toản** | Thiết kế sơ đồ kết nối phần cứng giữa STM32, L298N và encoder; triển khai đo tốc độ thực tế; kiểm thử và tinh chỉnh các hệ số PID. | 25% |
| **2202145 Tạ Đình Kiên** | Viết báo cáo Word cho toàn bộ bài C6_N3; mô tả các bước triển khai, phân tích các thông số PID; tổng hợp kết quả thực nghiệm và đề xuất cải tiến. | 25% |