ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA VẬT LÝ

NHÓM 1

ĐIỀU KHIỂN GÓC QUAY ĐỘNG CƠ SỬ DỤNG THUẬT TOÁN PID

Tiểu luận môn học

Học phần: Thực hành hệ thống nhúng

(Lớp K66 - Kỹ thuật Điện tử và Tin học)

Hà Nội - 2024

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA VẬT LÝ

NHÓM 1

ĐIỀU KHIỂN GÓC QUAY ĐỘNG CƠ SỬ DỤNG THUẬT TOÁN PID

Tiểu luận môn học

Học phần: Thực hành hệ thống nhúng

(Lớp K66 - Kỹ thuật Điện tử và Tin học)

Giảng viên hướng dẫn: Đỗ Anh Tuấn

Hà Nội - 2024

THÀNH VIÊN NHÓM 1

1. Lê Sơn Tùng - 21002243
2. Lê Trung Kiên - 21002214
3. Phạm Thu Thủy - 21002240
4. Từ Văn Hoài Nam - 21002220
5. Ngô Hoài Nam - 21002219
6. Trần Tuấn Phong - 21002223
7. Nguyễn Phan Ngọc Anh - 21002188

Lời cảm ơn

Lời đầu tiên, nhóm chúng tôi xin được gửi lời cảm ơn chân thành tới thầy Đỗ Anh Tuấn, trong suốt quá trình giảng dạy của môn học, thầy đã truyền tải những kiến thức và giải đáp những thắc mắc của chúng tôi trong môn lập trình nhúng AVR.

Nhóm xin được gửi lời cảm ơn tới các thầy Nguyễn Cảnh Việt và các thầy trong Bộ môn Tin học Vật lý đã luôn tạo điều kiện để chúng tôi được học tập và hoàn thành bài báo cáo này.

Lời cuối cùng, xin được cảm ơn tất cả các thành viên trong nhóm đã không ngại khó khăn, luôn đoàn kết cùng hỗ trợ lẫn nhau để hoàn thành đề tài này

Mục lục

Danh sách hình vẽ

Danh sách tên viết tắt

SP	**S**et **P**oint
PV	**P**rocess **V**ariable
MV	**M**anipulated **V**ariable
e	**e**rorr
P	**P**roportional
I	**I**ntegral
D	**D**erivative

MỞ ĐẦU

Điều khiển tự động là một lĩnh vực khoa học hiện đại. Hiện nay, ta có thể thấy điều khiển tự động xuất hiện trong nhiều lĩnh vực như: đời sống, công nghiệp, nông nghiệp, năng lượng, giao thông, quân sự... Rõ ràng, điều khiển tự động đóng vai trò quan trọng trong sự phát triển của đời sống cũng như trong cuộc sống thường ngày.

Thuật toán điều khiển hệ thống bắt đầu xuất hiện từ giữa thế kỷ XIX và phát triển mạnh mẽ vào những năm 60 của thế kỷ XX với lý thuyết điều khiển nâng cao. Sự ra đời của chúng giúp con người đạt được nhiều thành tựu to lớn trong nhiều lĩnh vực. Một trong những thuật toán điều khiển vẫn được sử dụng cho đến ngày này đó chính là bộ điều khiển PID. Đây là hệ thống điều khiển phản hồi kiểu vòng kín phổ biến nhất hiện nay với cấu trúc đơn giản, dễ thực hiện và tính hiểu quả cao.

Để làm rõ hơn về bộ điều khiển PID, tiểu luận này sẽ tìm hiểu về thuật toán PID và ứng dụng của nó trong các mô hình thực tế. Nội dung của tiểu luận sẽ được chia làm ba chương như sau:

Chương 1: Tổng quan về PID.

Chương 2: Thiết kế hệ thống và thực nghiệm.

Chương 3: Kết luận.

Chương 1 TỔNG QUAN VỀ PID

1.1 Giới thiệu chung

PID (Proportional Integral Derivative) là một cơ chế phản hồi vòng điều khiển được sử dụng rộng rãi trong các hệ thống điều khiển tự động. Bộ điều khiển PID sẽ tính toán sai số là hiệu giữa giá trị trả về của hệ thống và giá trị điểm đặt mong muốn, từ đó thực hiện giảm sai số về mức nhỏ nhất bằng cách điều chỉnh giá trị điều khiển đầu vào. Để hệ thống đạt được kết quả tốt nhất, các thông số PID sẽ được điều chỉnh theo tính chất, đặc trưng của hệ thống.

1.2 Cơ bản về vòng điều khiển

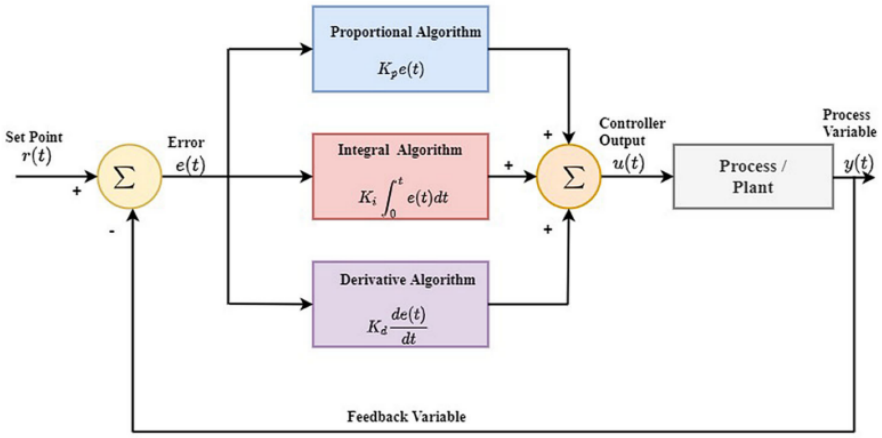
Xét ví dụ về một cánh tay robot có thể điều khiển chuyển động và xác định vị trí của nó. Động cơ điện được sử dụng sao cho khi cấp điện cho động cơ, cánh tay sẽ di chuyển nâng lên hoặc hạ xuống. Khi vận hành, cần điều khiển cánh tay đến một vị trí cụ thể, tuy nhiên việc cấp năng lượng đơn thuần không thể điều khiển cánh tay robot một cách chính xác bởi quán tính, trọng lực, ngoại lực tác động lên nó trong quá trình vận hành (dịch chuyển, nâng vật, hạ vật...). Chính vì thế, ta cần sử dụng một vòng điều khiển để có thể vận hành cánh tay robot với sai số nhỏ nhất có thể.

- PV (Process Variable): Biến quá trình - vị trí của cánh tay robot.
- SP (Setpoint): Vị trí điều khiển mong muốn.
- e (error): Sai số hệ thống, là hiệu giữa SP và PV.
- MV (Manipulated Variable): Biến điều khiển, là đầu ra của bộ điều khiển PID.

Từ việc xác định vị trí hiện tại của động cơ (PV) so với vị trí điểm đặt mong muốn, hệ thống sẽ tính toán sai số, kết hợp với các thuật toán, bộ điều khiển sẽ cung cấp cho động cơ một dòng điện phù hợp để cánh tay robot đạt được vị trí mong muốn.

1.3 Cấu trúc của bộ điều khiển PID

Tiểu luận này trình bày dạng song song của bộ điều khiển PID. Bộ điều khiển PID gồm ba thành phần: tỉ lệ, tích phân, vi phân.



Hình 1.1: Cấu trúc của bộ điều khiển PID

Tổng của ba thành phần này chính là đầu ra của hệ thống(MV).

$$MV(t) = u(t) = P_{out} + I_{out} + D_{out} \tag{1.1}$$

$$= K_p.e(t) + K_i.\int_0^t e(t)dt + K_d.\frac{d}{dt}e(t) \tag{1.2}$$

1.3.1 Thành phần tỉ lệ

Tỷ lệ (Proportional): Làm thay đổi giá trị đầu ra tỷ lệ với giá trị sai số hiện tại. Tuy nhiên trong thực tế, do sự sai lệch nên sẽ không bao giờ đặt được đến giá trị mong muốn. Để đáp ứng tỉ lệ cần nhân sai số với độ lợi K_p. Độ lợi tỷ lệ cao dẫn đến sự thay đổi lớn cho đầu ra. Nếu độ lợi tỷ lệ quá cao, hệ thống có thể trở nên không ổn định. Ngược lại, độ lợi nhỏ dẫn đến phản hồi đầu ra nhỏ và bộ điều khiển kém nhạy hơn. Tỷ lệ đóng góp phần lớn sự thay đổi đầu ra.

Thành phần tỉ lệ được cho bởi:

$$P_{out} = K_p.e(t) \tag{1.3}$$

Trong đó:

- P_{out}: Đầu ra của khâu tỉ lệ.
- K_p: Độ lợi tỉ lệ(là một tham số điều chỉnh).
- e: sai số = SP - PV.
- t: thời gian tức thời(hiện tại).

1.3.2 Thành phần tích phân

Tích phân (Integral): Khâu này tỉ lệ với biên độ sai số và thời gian xảy ra sai số. Cộng dồn các sai số tức thời theo thời gian (tích phân sai số) cho ra tích lũy sai số. Tích lũy sai số sau đó được nhân với độ lợi tích phân K_p. Thuật ngữ tích phân tăng tốc độ di chuyển của hệ thống về phía điểm đặt và loại bỏ lỗi trạng thái ổn định còn lại xảy ra với bộ điều khiển chỉ sử dụng thành phần tỉ lệ. Tuy nhiên, việc cộng dồn các sai số tích lũy từ quá khứ của khâu tích phân có thể khiến giá trị hiện tại vượt quá giá trị điểm đặt.

Thành phần tích phân được cho bởi:

$$I_{out} = K_i . \int_0^t e(t)dt \tag{1.4}$$

Trong đó:

- P_{out}: Đầu ra của khâu tích phân.
- K_i: Độ lợi tích phân(là một tham số điều chỉnh).
- e: sai số = SP - PV.
- t: thời gian tức thời(hiện tại).

1.3.3 Thành phần đạo hàm

Đạo hàm (Derivative): Một thành phần khác được sử dụng trong bộ điều khiển PID là đạo hàm. Tốc độ thay đổi của sai số quá trình được tính toán bằng cách xác định độ dốc của sai số theo thời gian (tức là đạo hàm bậc một theo thời gian) và nhân với độ lợi K_p. Đạo hàm dự đoán hành vi của hệ thống từ đó cải thiện ổn định của hệ thống.

Thành phần đạo hàm được cho bởi:

$$D_{out} = K_d . \frac{d}{dt} e(t) \tag{1.5}$$

Trong đó:

- D_{out}: Đầu ra của khâu tích phân.
- K_d: Độ lợi(là một tham số điều chỉnh).
- e: sai số = SP - PV.
- t: thời gian tức thời(hiện tại).

1.4 Một số bộ điều khiển được xây dựng từ P, I và D

Tùy vào đặc trưng của một số hệ thống và mục đích sử dụng, có thể kết hợp các thành phần: tỷ lệ (P), tích phân (I) và đạo hàm (D) để tạo nên một số bộ điều khiển.

1.4.1 Bộ điều khiển tỷ lệ tích phân

Bộ điều khiển tỷ lệ tích phân (PI) được xây dựng bằng việc kết hợp hai thành phần tỷ lệ (P), tích phân (I) và không sử dụng thành phần đạo hàm (D) của thuật toán PID. Hệ thống PI là một hình thức kiểm soát phản hồi. Nó cung cấp thời gian phản hồi nhanh hơn so với điều khiển chỉ sử dụng thành phần tích phân (I) do bổ sung thành phần tỷ lệ (P). Điều khiển PI ngăn hệ thống dao động và điều khiển hệ thống về điểm đặt (SP).

Đầu ra của bộ điều khiển PI:

$$MV(t) = P_{out} + I_{out} \tag{1.6}$$

$$= K_p.e(t) + K_i.\int_0^t e(t)dt \tag{1.7}$$

1.4.2 Bộ điều khiển tỷ lệ đạo hàm

Một sự kết hợp khác của hệ thống PID là bộ điều khiển tỷ lệ đạo hàm (PD). Thành phần của bộ điều khiển gồm hai thành phần tỷ lệ (P) và đạo hàm (D), bỏ qua thành phần tích phân (I). Một hệ thống PD hoạt động trên quy trình hiện tại và dự đoán. Đầu ra điều khiển là sự kết hợp tuyến tính của tín hiệu lỗi và đạo hàm của nó.

Đầu ra của bộ điều khiển PD:

$$MV(t) = P_{out} + D_{out} \tag{1.8}$$

$$= K_p.e(t) + K_d.\frac{d}{dt}e(t) \tag{1.9}$$

1.4.3 Bộ điều khiển tỷ lệ - tích phân - đạo hàm

Như đã đề cập ở trên, bộ điều khiển PID là sự kết hợp của cả ba khâu: tỷ lệ, tích phân và đạo hàm. Hệ thống PID được sử dụng rộng rãi nhất bởi nó là sự kết hợp các ưu điểm của ba thành phần này. Mỗi thành phần có vai trò đặc trưng giúp cho hệ thống được tối ưu trong quá trình hoạt động và mang lại hiệu suất cao.

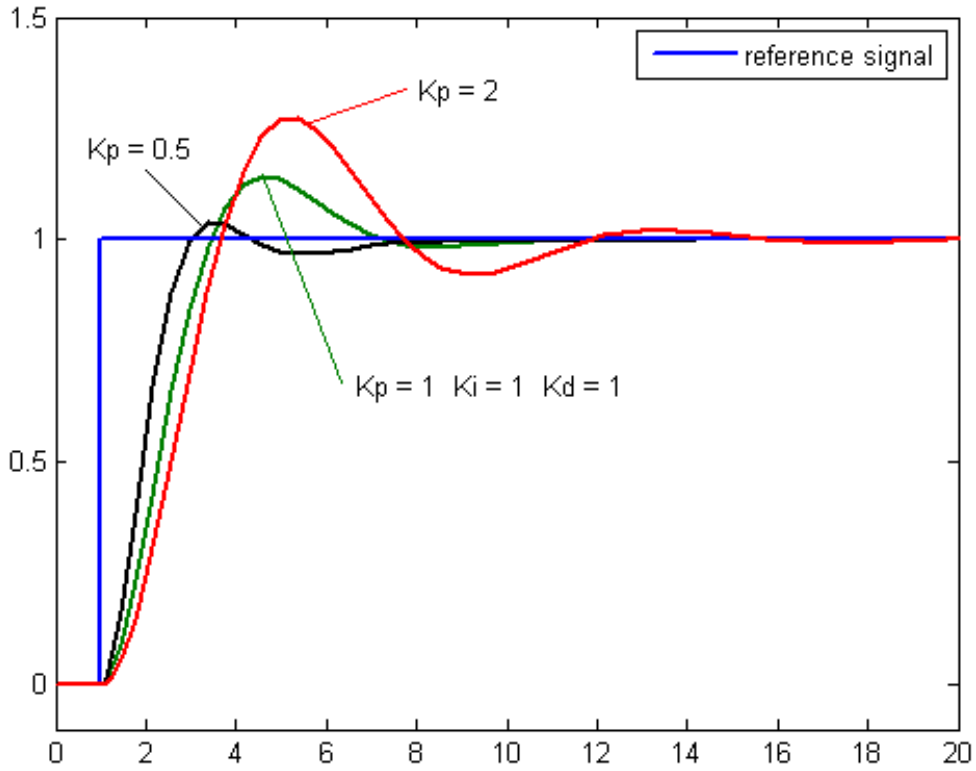
1.5 Các thành phần độ lợi của hệ thống

Với cấu trúc đơn giản và hiệu quả mang lại cao mà ngày nay điều khiển PID vẫn được sử dụng rộng rãi trong công nghiệp. Ba hằng số độ lợi: tỷ lệ (K_p), tích phân (K_i) và đạo hàm (K_d) có ý nghĩa quan trọng trong một hệ thống PID. Việc thay đổi các hằng số này sẽ quyết định đến đầu ra của hệ thống. Chính vì thế, một bộ điều khiển PID được "tinh chỉnh tốt" có thể mang lại hiệu suất tuyệt vời. Từ "tinh chỉnh tốt" nhấn mạnh rằng hiệu

suất của bộ điều khiển PID chủ yếu phụ thuộc vào quá trình điều chỉnh [1]. Mỗi hệ thống cần có bộ số K_p, K_i, Kd khác nhau sao cho phù hợp để hệ thống hoạt động ổn định, đảm bảo hiệu suất với bài toán đặt ra.

1.5.1 Độ lợi tỷ lệ

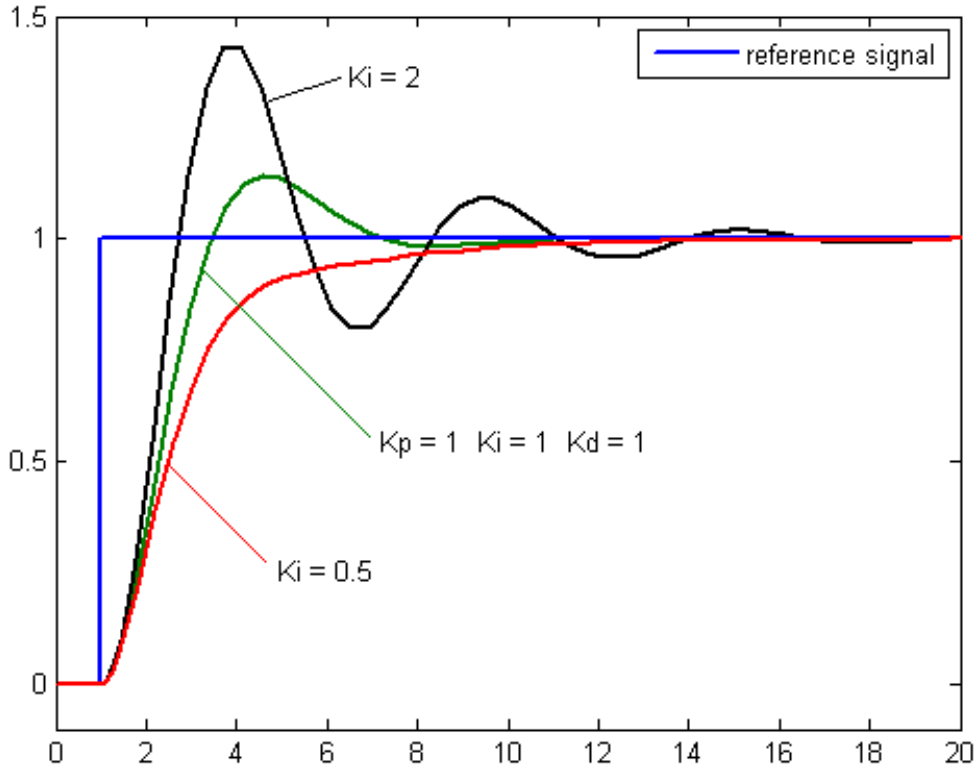
Giá trị của độ lợi tỷ lệ (K_p) càng lớn thì hệ thống đáp ứng càng nhanh, tuy nhiên sai số sẽ càng lớn. Một giá trị K_p quá lớn sẽ làm cho hệ thống hoạt động bị sai lệch, thiếu sự ổn định và tin cậy. Ngược lại, nếu K_p quá nhỏ cũng khiến cho hệ thống hoạt động kém nhạy, dẫn đến mất tính ổn định cho toàn hệ thống.



Hình 1.2: Đáp ứng khâu tỷ lệ của hệ thống với Ki, K_d không đổi

1.5.2 Độ lợi tích phân

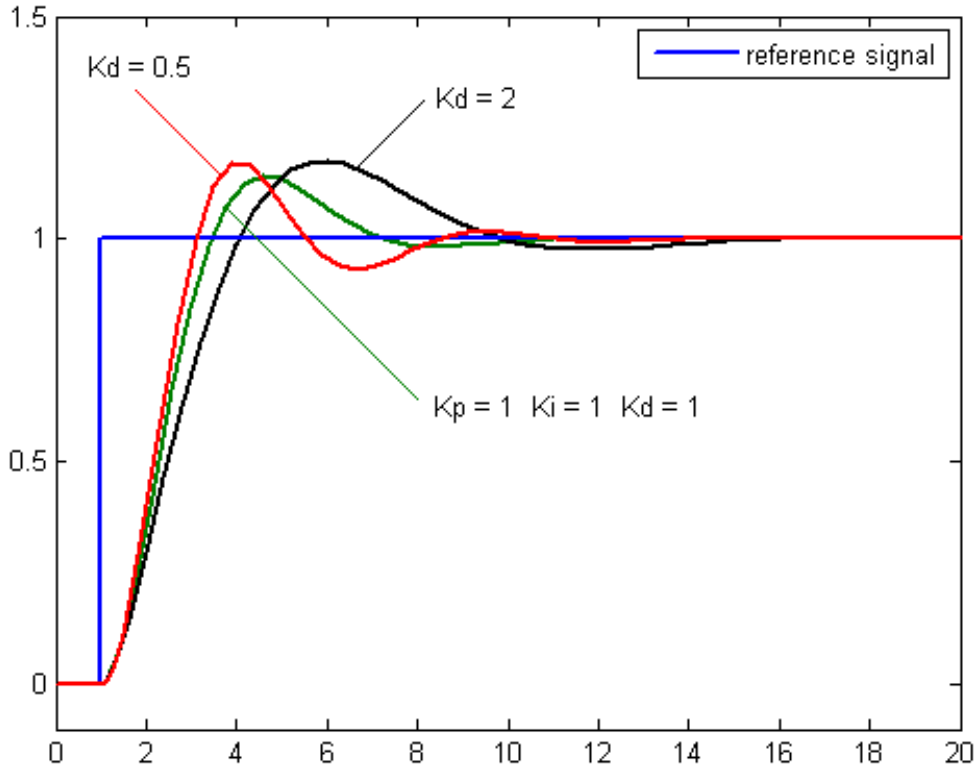
Giá trị của độ lợi tích phân (K_i) càng lớn thì sai số ổn định bị triệt tiêu càng nhanh, đổi lại độ vọt lố của hệ thống càng lớn. Các giá trị sai số được tích phân trong quá trình đáp ứng của hệ thống cần phải được triệt tiêu bằng tích phân trước khi tiến tới trạng thái ổn định.



Hình 1.3: Đáp ứng khâu tích phân của hệ thống với Ki, K_d không đổi

1.5.3 Độ lợi đạo hàm

Giá trị của độ lợi đạo hàm (K_d) càng lớn, độ vọt lố càng giảm nhưng lại làm chậm đáp ứng của hệ thống và có thể dẫn đến sự mất ổn định trong quá trình hoạt động của hệ thống.



Hình 1.4: Đáp ứng khâu đạo hàm của hệ thống với Ki, K_d không đổi

1.6 Một số phương pháp điểu chỉnh thông số PID

Một bộ điều khiển PID phụ thuộc rất nhiều vào ba thông số độ lợi. Nếu ba thông số K_p, K_i, K_d không được lựa chọn thích hợp, hệ thống sẽ hoạt động với độ tin cậy thấp, độ chính xác kém và dẫn đến sai lệch của hệ thống. Chính vì thế, cần xác định bộ số này sao cho phù hợp với hệ thống. Có nhiều phương pháp để điều chỉnh thông số PID cho một hệ thống.

1.6.1 Phương pháp điều chỉnh thủ công

Đây là phương pháp hiệu chỉnh đơn giản, dễ dàng đạt được một hệ thống có đáp ứng đầu ra như ý muốn và không cần thiết phải có mô hình chi tiết về toán học của hệ thống.

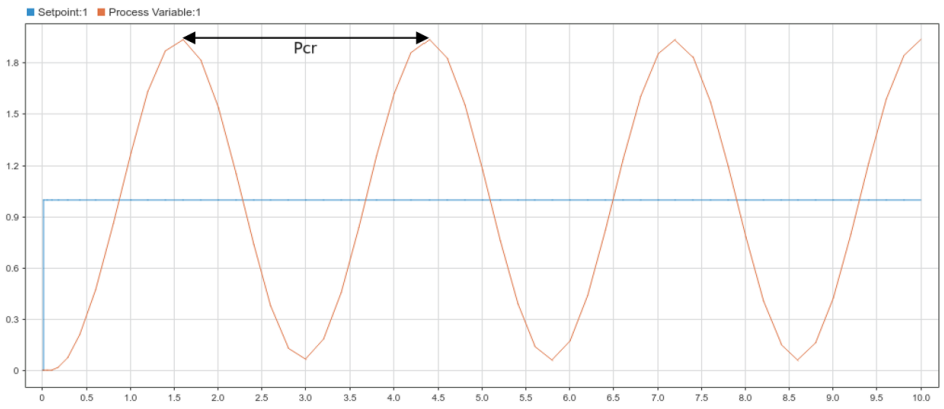
Tuy nhiên phương pháp này là mất nhiều thời gian và người hiệu chỉnh cần phải có nhiều kinh nghiệm. Các bước hiệu chỉnh:

- Đầu tiên chỉnh cả ba thông số K_p, K_i, K_d về 0.
- Tăng dần K_p cho đến khi đầu ra của vòng điều khiển dao động, sau đó K_p được đặt về xấp xỉ một nửa giá trị đó.
- Tiếp theo, tăng giá trị K_i đến giá trị phù hợp sao cho hệ thống đủ thời gian xử lý. Tuy nhiên, giá trị K_i quá lớn sẽ gây mất ổn định.
- Cuối cùng tăng K_d đến khi dao động của hệ thống bị loại bỏ. K_d quá lớn sẽ khiến đáp ứng dư và vọt lố.

1.6.2 Phương pháp Ziegler–Nichols 2

Phương pháp Ziegler–Nichols được đưa ra bởi John G. Ziegler và Nathaniel B. Nichols vào những năm 1940. Các bước hiệu chỉnh bằng phương pháp này bao gồm:

- Chỉnh K_p, K_i, K_d về 0.
- Tăng dần độ lợi K_p từ 0 cho đến khi đạt được độ lợi K_{cr} mà tại đó đầu ra của của vòng điều khiển dao động với biên độ không đổi.



Hình 1.5: Mô tả đầu ra của hệ thống dao động với chu kỳ không đổi

- Sau khi hệ dao động tuần hoàn, tiến hành xác định chu kỳ dao động P_{cr} của hệ thống. Lưu ý rằng, đơn vị thời gian được lấy theo đơn vị của thời gian lấy mẫu trong hệ thống.

- Từ các giá trị K_{cr} và P_{cr} tìm được, có thể xác định được các giá trị T_i và T_d theo bảng:

Type of Controller	K_p	T_i	T_d
P	$0.5K_{cr}$	∞	0
PI	$0.45K_{cr}$	$\frac{1}{1.2}P_{cr}$	0
PID	$0.6K_{cr}$	$0.5P_{cr}$	$0.125P_{cr}$

Hình 1.6: Bảng Ziegler-Nichols 2

- Dựa vào Ziegler-Nichols 2, các thông số K_i và K_d được xác định bởi:

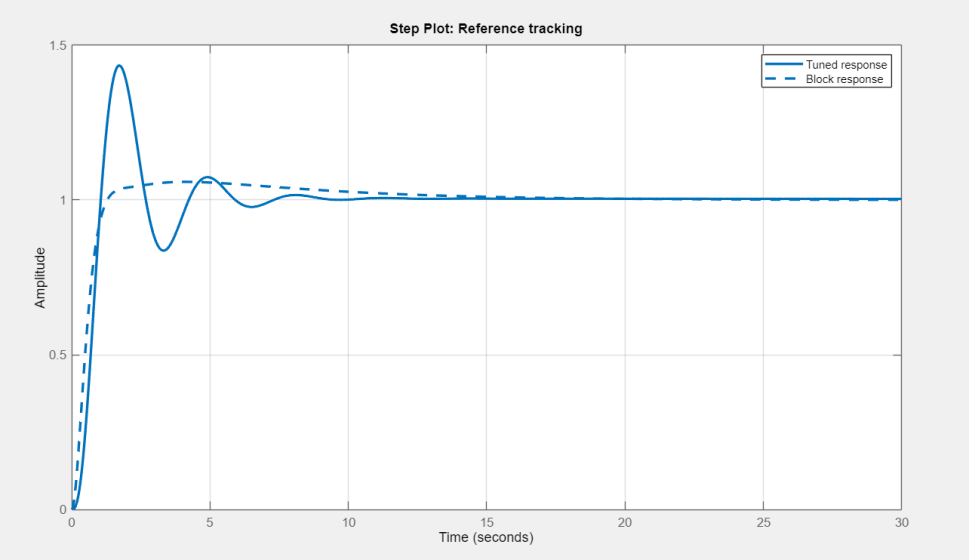
$$K_i = K_p/T_i \tag{1.10}$$

$$K_d = K_p.T_d \tag{1.11}$$

1.6.3 Phương pháp sử dụng phần mềm chuyên dụng

Đây là phương pháp điều chỉnh chắc chắn, chính xác. Đây là phương pháp giúp tối ưu hóa hiệu suất hệ thống, dễ dàng điều chỉnh và khả năng đáp ứng linh hoạt. Hạn chế của phương pháp này là tốn kém chi phí và cần hiểu rõ về chuyên môn cũng như hệ thống.

Dưới đây là mô tả phương pháp hiệu chỉnh độ lợi bằng cách sử dụng phần mềm Matlab Simulink. Đây là phương pháp mang lại độ tối ưu và tin cậy cao cho hệ thống. Tuy nhiên, để sử dụng phương pháp này thì cần phải đưa ra các phương trình toán học cho hệ thống, đây là một công việc đòi hỏi tính học thuật cao và khá khó khăn.



Hình 1.7: Bảng hiệu chỉnh độ lợi bằng phần mềm Matlab

Chương 2 THIẾT KẾ HỆ THỐNG

2.1 Giới thiệu

Dựa vào lý thuyết của thuật toán điều khiển PID, tiểu luận này đã xây dựng mô hình thực tế có sử dụng bộ điều khiển này đó là mô hình điều khiển vị trí góc quay động cơ DC Encoder.

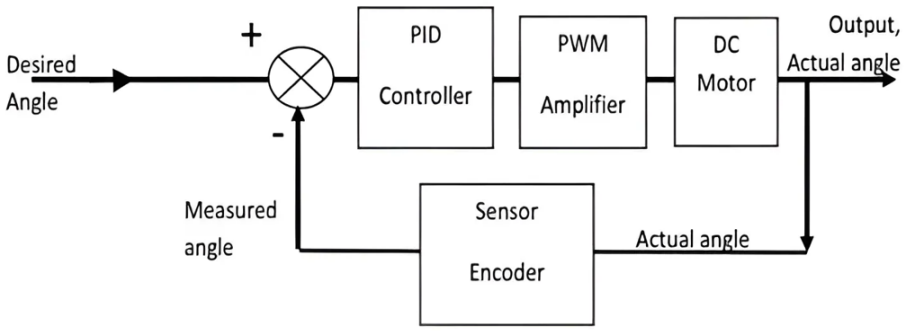
2.2 Điều khiển vị trí góc quay động cơ DC Encoder sử dụng thuật toán PID

2.2.1 Tổng quan

Bằng việc sử dụng thuật toán điều khiển PID, có thể điều khiển vị trí của động cơ DC Encoder dựa vào đĩa Encoder được gắn sẵn trên trục động cơ.

2.2.2 Lý thuyết

Sử dụng động cơ DC giảm tốc GA25 Encoder. Động cơ có gắn thêm phần Encoder là một đĩa Encoder 11 xung, hai kênh A và B để có thể trả xung về vi điều khiển. Dựa vào giá trị xung trả về, có thể biết được vị trí hiện tại của đĩa. Từ đó có thể dễ dàng xác định vị trí hoặc vận tốc của động cơ. Cùng với việc sử dụng thuật toán điều khiển PID, vị trí hay tốc độ của động cơ được điều khiển một cách chính xác.



Hình 2.1: Sơ đồ khối hệ thống

2.2.3 Một số chức năng được sử dụng trong mô hình

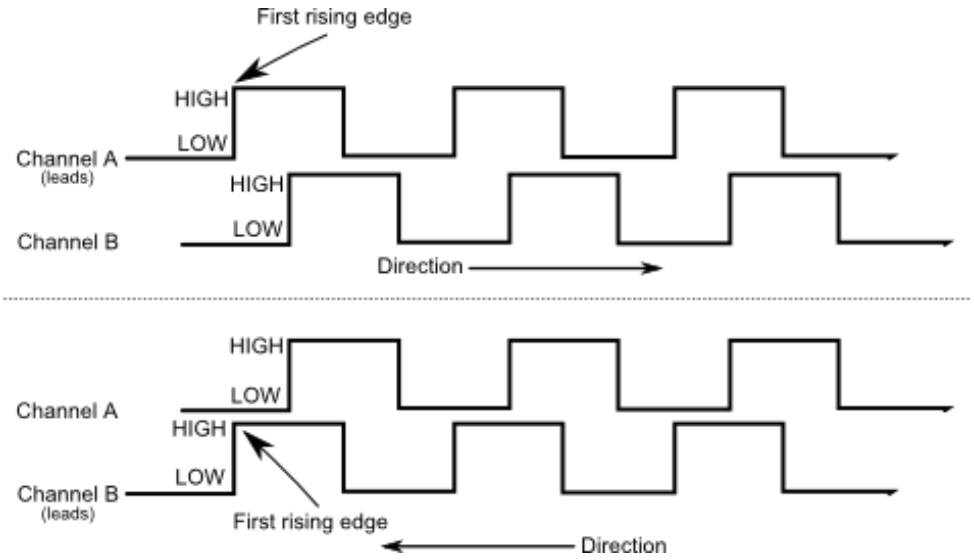
Ngắt ngoài - External Interrupt

Ngắt (interrupt) là những lời gọi hàm tự động khi hệ thống sinh ra một sự kiện. Những sự kiện này được nhà sản xuất vi điều khiển thiết lập bằng phần cứng và được cấu hình trong phần mềm bằng những tên gọi cố định. Ngắt giúp chương trình gọn nhẹ và xử lý nhanh hơn. Ví dụ với việc kiểm tra một nút nhấn có được nhấn hay không, việc kiểm tra trạng thái nút nhấn trong vòng lặp while(1) của chương trình có thể gây ảnh hưởng tới chương trình chính. Với việc sử dụng ngắt, chỉ cần nối nút nhấn đến đúng chân có hỗ trợ ngắt, sau đó cài đặt ngắt sẽ sinh ra khi trạng thái nút chuyển từ HIGH->LOW. Thêm một tên hàm sẽ gọi khi ngắt sinh ra. Như vậy đoạn chương trình ngắt sẽ cho biết trạng thái nút nhấn.

Có bốn trạng thái của ngắt bao gồm:

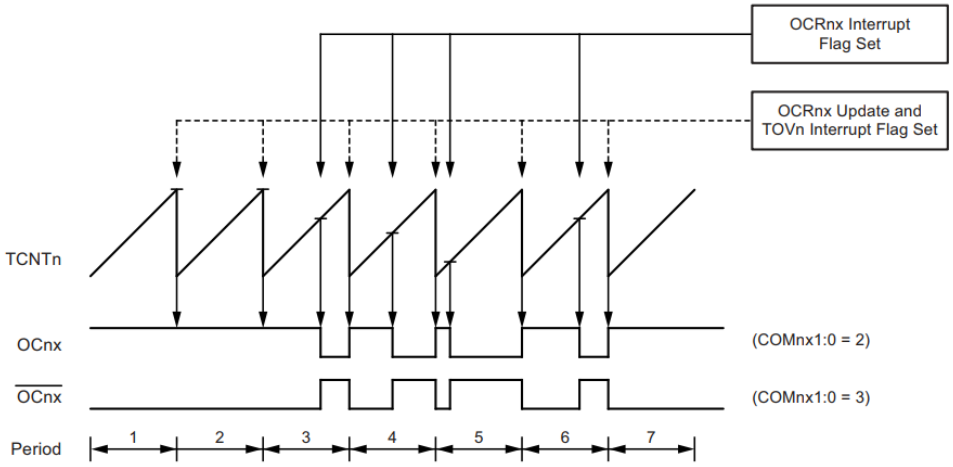
- LOW: kích hoạt liên tục khi trạng thái chân digital có mức thấp
- HIGH: kích hoạt liên tục khi trạng thái chân digital có mức cao.
- RISING: kích hoạt khi trạng thái của chân digital chuyển từ mức điện áp thấp sang mức điện áp cao.
- FALLING: kích hoạt khi trạng thái của chân digital chuyển từ mức điện áp cao sang mức điện áp thấp.

Trong mô hình này việc xác định chiều quay của động cơ sẽ được thực hiện bằng cách sử dụng ngắt INT0 (chân PD2) ở chế độ RISING, kênh A của Encoder được nối với chân PD2, kênh B được nối vào chân PD3. Như vậy mỗi khi ngắt được tạo ra, bằng cách đọc giá trị ở kênh B, có thể xác định được chiều quay của động cơ.



Hình 2.2: Xác định chiều quay của động cơ DC Encoder

Tạo xung Fast PWM



Hình 2.3: Tạo xung Fast PWM

WGM2:0 = 0b011 hoặc 0b111.

Counter chạy từ 0x00 tới TOP rồi được reset lại về 0x00.

TOP = MAX khi WGM2:0 = 0b011; TOP = OCRxA nếu WGM2:0 = 0b111

Khi CTNTx = OCRxn, tạo ra phản ứng tại chân OCxn. Phản ứng tại chân OCxn tùy thuộc vào giá trị của bit COMxn1:0

Ngắt được tạo ra (nếu cho phép) khi CTNTx = TOP.

COM0A1	COM0A0	Description
0	0	Normal port operation, OC0A disconnected.
0	1	WGM02 = 0: Normal port operation, OC0A disconnected. WGM02 = 1: Toggle OC0A on compare match.
1	0	Clear OC0A on compare match, set OC0A at BOTTOM, (non-inverting mode).
1	1	Set OC0A on compare match, clear OC0A at BOTTOM, (inverting mode).

Hình 2.4: Chế độ tạo xung Fast PWM

Tần số PWM tại đầu ra OCxn chỉ phụ thuộc vào tần số clock đưa vào Timer.

Giá trị trên thanh ghi so sánh OCRxn quyết định mức năng lượng được kích hoạt trong mỗi chu kì.

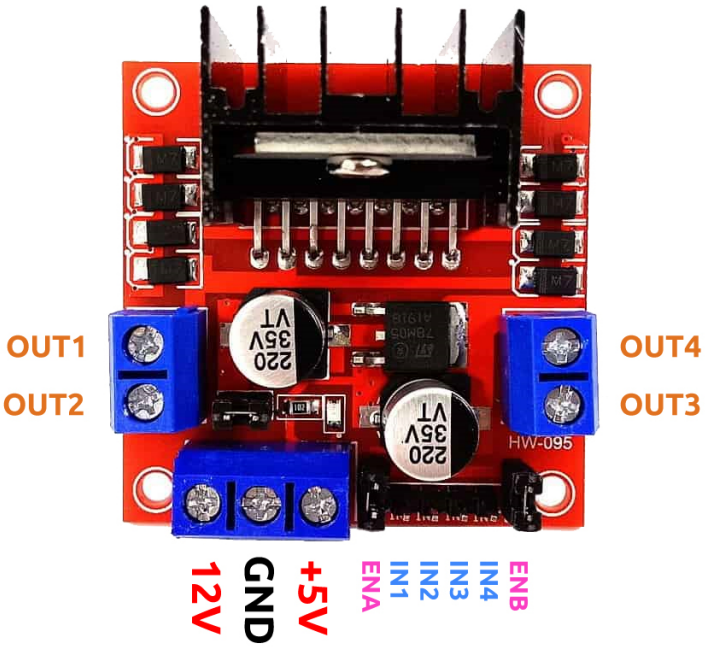
Công thức tính chu kì xung PWM:

$$f_{OCxA} = \frac{f_{IO}}{2.d.MAX} \tag{2.1}$$

Trong đó:

- d : hệ số chia clock của timer
- f_{IO} : Tần số clock của IO
- f_{OCxA} : Tần số xung tại đầu ra so sánh OCxA

Như vậy, bằng việc tạo xung Fast PWM, có thể dễ dàng điều khiển tốc độ quay của động cơ thông qua module L298N.

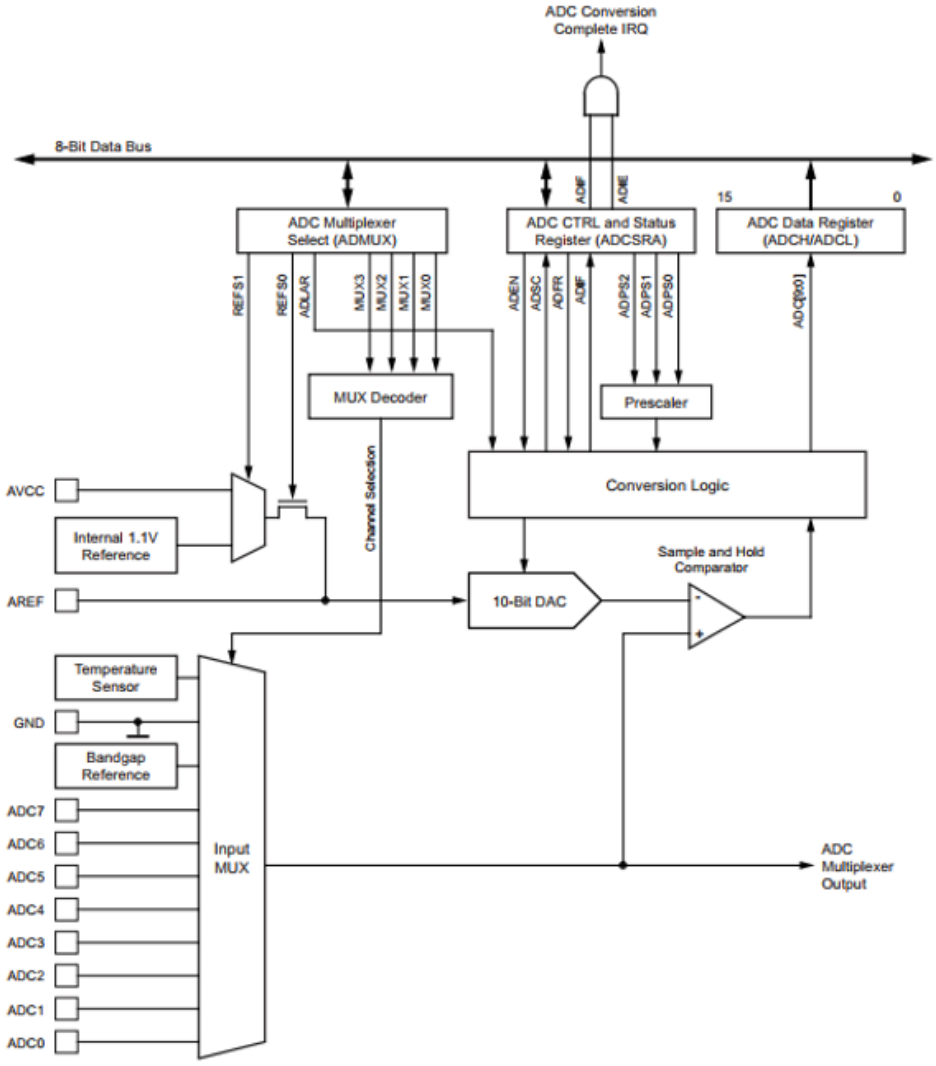


Hình 2.5: Module L298N

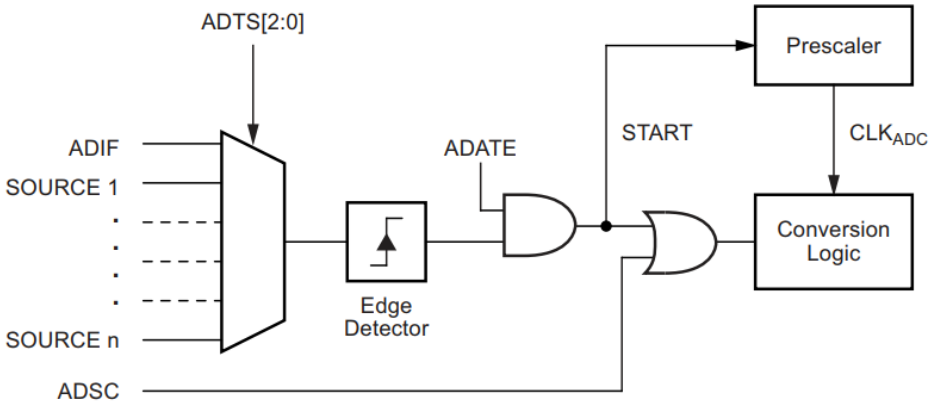
Chiều quay của động cơ được thay đổi bằng việc thay đổi giá trị trên chân PD6 nối với IN3 của module và PD7 nối với INT4 của module.

Analog to Digital Converter (ADC)

Sơ đồ khối ADC



Hình 2.6: Sơ đồ khối ADC trên ATmega328p



Hình 2.7: Hoạt động của khối ADC

ADTS (ADC Trigger Select): Chọn nguồn đầu vào cho bộ ADC

ADSC (ADC Start Conversion): Bắt đầu chuyển đổi

SOURCE: Nguồn đầu vào analog, ADC0 đến ADC5, tương ứng với PC0 đến PC5

ADATE (ADC Auto Trigger Enable): Tự động chuyển đổi

ADC cần clock trong tầm 50kH tới 200 kHz để đảm bảo độ phân giải tối đa 10 bit. Trong trường hợp không yêu cầu độ phân giải cao, cần tăng tốc độ lấy mẫu, ADC clock có thể được cấu hình cao hơn 200 kHz.

Việc thay đổi tốc độ ADC clock dựa trên các bit ADPS[2:0] trên thanh ghi ADCSRA.

Thời gian cho 1 lần chuyển đổi thường là 13 ADC clock. Ngoại trừ lần chuyển đổi đầu tiên cần 25 ADC clock.

Một số thanh ghi quản lý ADC:

- Thanh ghi ADMUX (ADC multiplexer Selection) Là thanh ghi lựa chọn điện áp tham chiếu, căn lề thanh ghi kết quả và lựa chọn kênh đầu vào cho ADC.
- Thanh ghi ADCSRA (ADC Control and Status Register A) Cấu hình và lựa chọn tốc độ lấy mẫu.
- Thanh ghi ADCSRB (ADC Control and Status Register B) Lựa chọn nguồn kích hoạt chế độ lấy mẫu tự động (Auto Conversion).
- Thanh ghi DIDR0 (Digital Input Disable) Loại bỏ đầu vào số cho các Input.
- Thanh ghi ADCL và ADCH (ADC Data Register) Thanh ghi chứa kết quả của quá trình chuyển đổi.

Giá trị của ADC nằm trong khoảng từ 0 - 1023 sau đó được chuyển về từ 0 - 360 ứng với 360 độ của động cơ. Giá trị ADC này sau đó được chuyển đổi tiếp thành giá trị của đĩa encoder tương ứng với góc quay của động cơ.

Giao tiếp TWI

Nhằm thuận tiện cho quá trình theo dõi hoạt động của hệ thống, mô hình sử dụng màn hình LCD1602 được gắn module I2C PCF8574, giao tiếp TWI với vi điều khiển. Các thống số như SP, PV hay giá trị góc quay sẽ được hiển thị lên LCD.

Chuẩn giao tiếp TWI được nghiên cứu và phát triển bởi Atmel, thực chất chính là chuẩn I2C của Philips Semiconductor với cái tên khác. Đây là giao tiếp sử dụng 2 dây kết nối là SCK (Serial Clock) và SDA (Serial Data), là chuẩn giao tiếp không đồng bộ và bán song công (half-duplex). Hai dây SCK và SDA cần điện trở kéo lên (Pull up resistor). Chế độ làm việc theo mô hình Multi-Master với cơ chế phân xử xung đột thiết bị chủ. Tốc độ clock cao nhất đạt 400 kHz. Slave được định địa chỉ 7 bit bằng lập trình, hỗ trợ General Call.

Các chế độ là việc của giao tiếp TWI:

- Master: Là thiết bị khởi tạo và kết thúc giao tiếp. Master sẽ tạo ra xung clock trên đường SCL. Master sẽ gửi địa chỉ yêu cầu kết nối với slave trên bus.
- Slave: Thiết bị được master gửi địa chỉ yêu cầu kết nối trên bus.
- Transmitter (Bộ truyền): Là thiết bị gửi dữ liệu lên trên bus.
- Receiver (Bộ nhận): Là thiết bị đọc dữ liệu trên bus.
- Như vậy sẽ có 4 chế độ cho thiết bị TWI là Master-transmit, Master-receive, Slave-transmit và Slave-receive.

Giao tiếp USART

USART (Universal Synchronous and Asynchronous serial Receiver and Transmitter): Là bộ truyền nhận thông tin nối tiếp, dữ liệu được truyền theo từng bit liên tục với nhau.

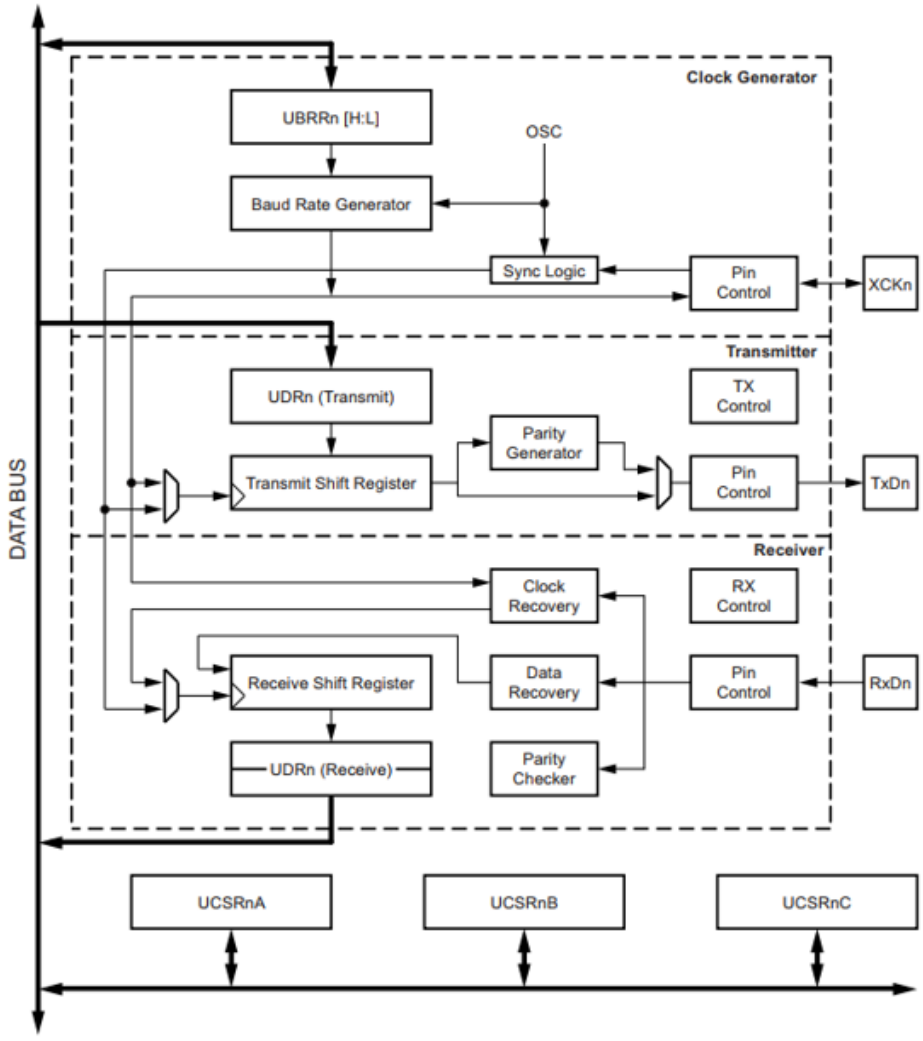
Có 2 chế độ: Đồng bộ và không đồng bộ

- Master: Không đồng bộ: Hai bên thu và nhận có nguồn clock riêng, dữ liệu được gửi theo từng byte (thêm bit start, stop và parity). Thường dung trong truyền dữ liệu giữa các thiết bị khác nhau.

- Slave: Đồng bộ: Hai bên thu và nhận sử dụng chung 1 nguồn clock, dữ liệu được gửi liên tục không cần bit start, stop và parity. Thường dung trong truyền dữ liệu on board.

Giao thức hoạt động ở chế độ song công (Full duplex), hỗ trợ khung truyền tin 5,6,7,8 hoặc 9 bit dữ liệu, 1 hoặc 2 bit dừng (stop bit), hỗ trợ kiểm tra lỗi (Check parity). Có 3 ngắt: Ngắt TX, ngắt TX data Register empty và ngắt RX.

Sơ đồ khối USART:



Hình 2.8: Sơ đồ khối USART

Chia làm 3 khối chính: Clock Generator, Transmitter và Receiver.

Chân XCK (PD4) chỉ dung trong chế độ đồng bộ.

Chân TX (PD1) là chân truyền tín hiệu (Transmit).

Chân RX (PD0) là chân nhận tín hiệu (Receiver).

Giao tiếp USART được sử dụng nhằm phục vụ xử lý số liệu và vẽ đồ thị hoạt động của hệ thống.

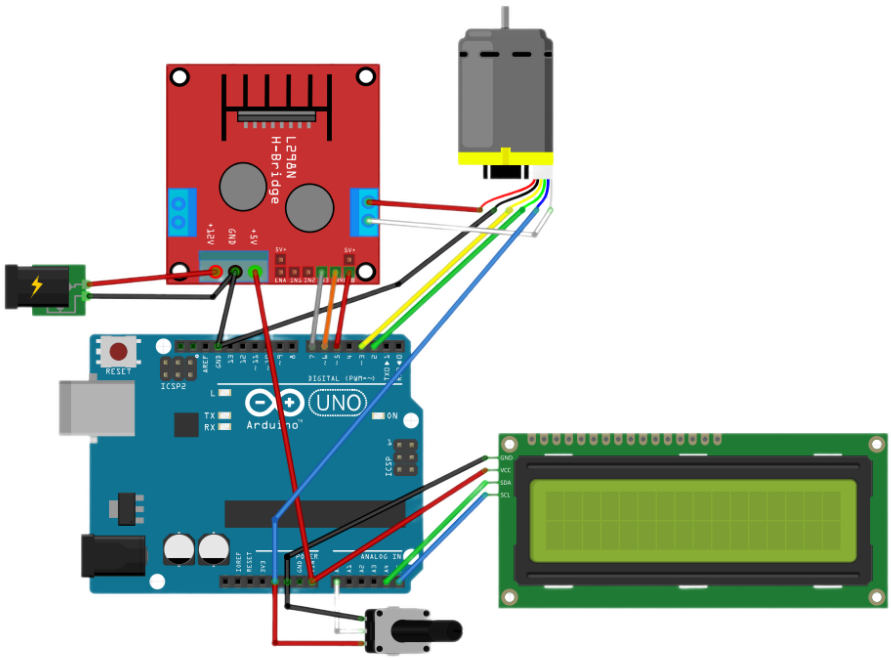
2.2.4 Ảnh mô hình

Vỏ hộp và các linh kiện được thiết kế và in 3D. Mô hình điều khiển vị trí góc động cơ. Có thể thay đổi giá trị SP bằng cách vặn chiết áp. Giá trị SP sau đó được hiển thị lên màn hình LCD1602, đồng thời động cơ cũng quay một góc có độ lớn đúng bằng độ lớn của giá trị SP. Mỗi vạch trên đĩa ứng với 10 độ, mũi trên trên trục động cơ sẽ giúp xác định góc quay của động cơ.

Hình 2.9: Mô mình điều khiển vị trí góc quay động cơ

2.2.5 Sơ đồ linh kiện

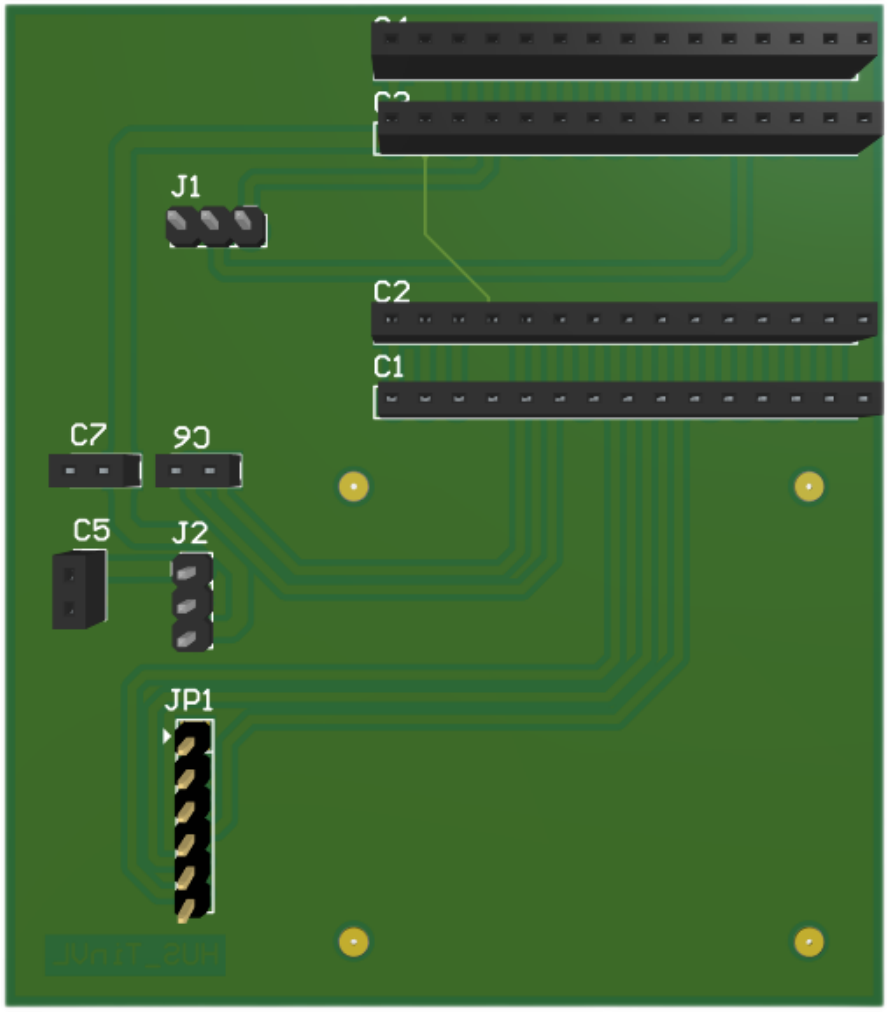
Mô hình được sử dụng một nguồn 12V cung cấp nguồn nuôi cho động cơ, hai chân ENA và ENB của L298N cần được nối với chân xung của vi điều khiển, chân GND cần được nối chung cho toàn mạch. Chi tiết sơ đồ chân nối được ghi ở phần phụ lục B



Hình 2.10: Sơ đồ chân mô mình điều khiển vị trí góc quay động cơ

2.2.6 Thiết kế mạch in

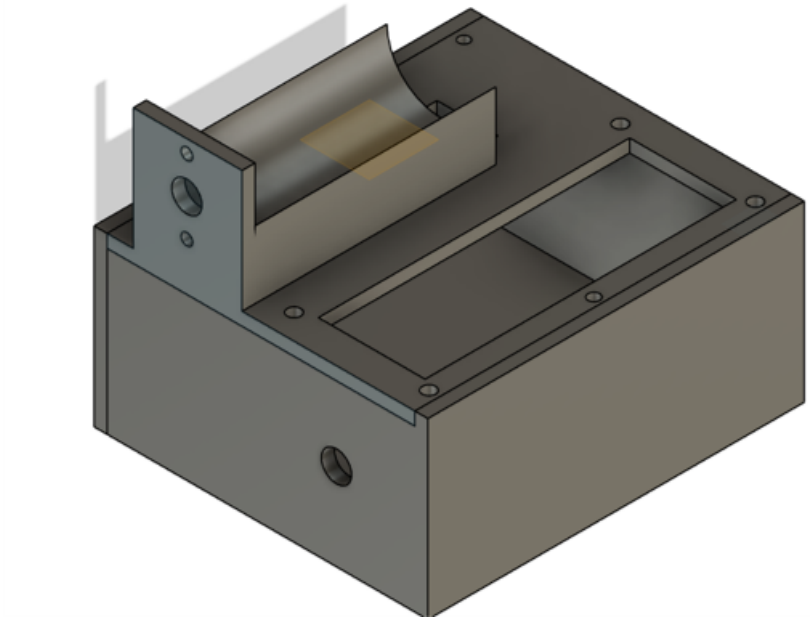
Mạch in của mô hình được thiết kế bằng phần mềm Altium.



Hình 2.11: Thiết kế mạch in mô mình điều khiển vị trí góc quay động cơ DC Encoder

2.2.7 Thiết kế vỏ hộp

Vỏ hộp của mô hình được thiết kế bằng phần mềm Fusion360 sau đó được in 3D.



Hình 2.12: Vỏ hộp của mô hình được vẽ 3D

2.2.8 Phương pháp tìm K_p, K_i, K_d

Với bộ điều khiển PID

Sử dụng phương pháp Ziegler-Nichols 2 để tìm độ lợi cho hệ thống, cùng với đó là hiệu chỉnh thủ công và tiến hành khảo sát để tìm ra ba hằng số độ lợi cho mô hình. Ở mô hình này, ba hằng số độ lợi K_p, K_i và K_d lần lượt là 6,75 ; 12,14 và 0,22.

Với bộ điều khiển PD

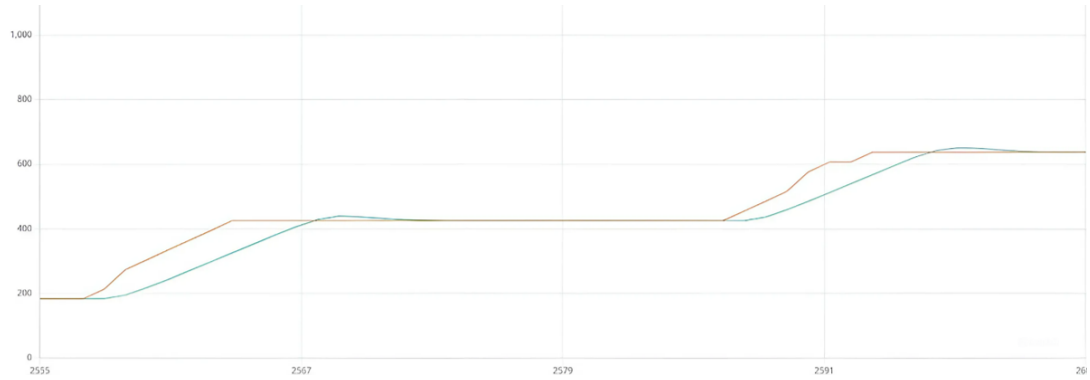
Bằng cách lược bỏ khâu tích phân trong bộ điều khiển PID, bộ điều khiển PD được xây dựng cho mô hình điều khiển vị trí động cơ DC Encoder. Tiến hành tìm thông số K_p, K_d cho hệ thống. Ở đây, hai giá trị K_p và K_d lần lượt được xác định bằng 6,7 và 0,2.

2.2.9 Kết quả

Hệ thống hoạt động ổn định với các giá trị góc được đặt ra. Khi ta thay đổi setpoint bằng cách vặn chiết áp, giá trị này sẽ được hiển thị lên màn hình LCD, đồng thời động cơ cũng quay một góc bằng giá trị setpoint. Cụ thể trong từng bộ điều khiển PD và PID sẽ cho các kết quả như sau.

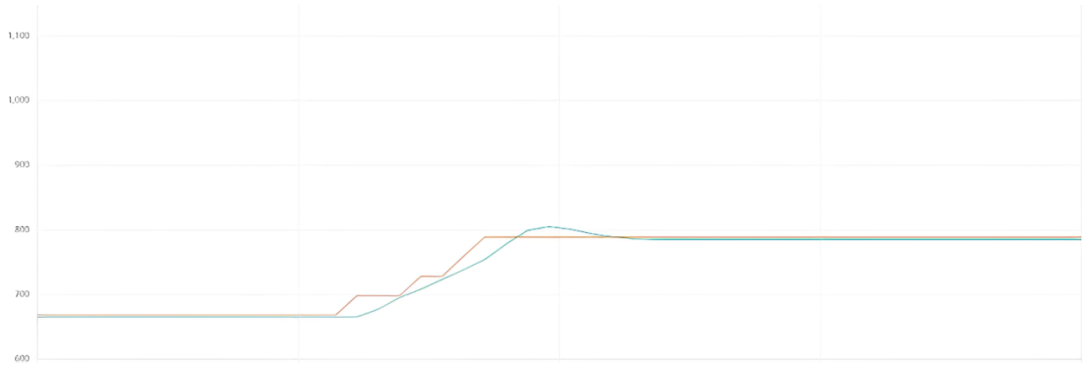
Với bộ điều khiển PD

Khảo sát hoạt động của hệ thống với bộ điều khiển PD cho thấy hệ thống hoạt động tương đối chính xác, phản hồi nhanh.



Hình 2.13: Hoạt động của hệ thống với bộ điều khiển PD (1)

Tuy nhiên trong một số trường hợp giá trị PV khác giá trị SP thì hệ thống không thể tính toán để điều khiển giá trị PV = SP do bộ điều khiển không có thành phần tích phân.

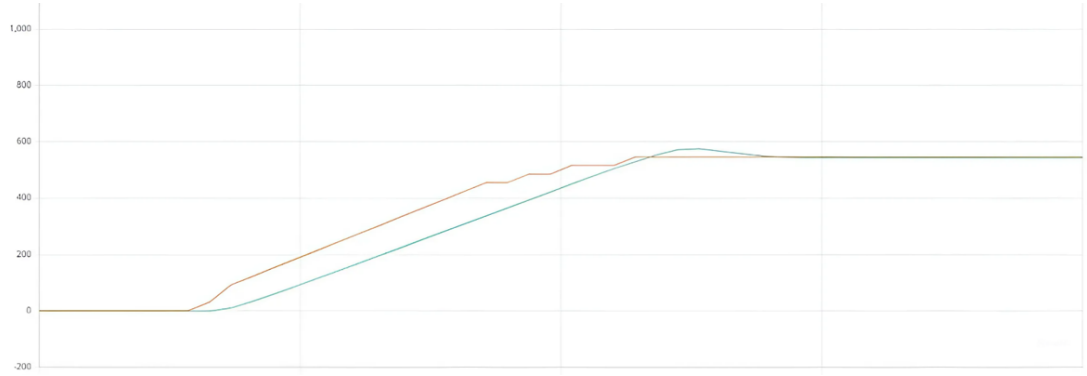


Hình 2.14: Hoạt động của hệ thống với bộ điều khiển PD (2)

Đổi lại hệ thống hoạt động với độ ổn định cao, không xảy ra hiện tượng dao động.

Với bộ điều khiển PID

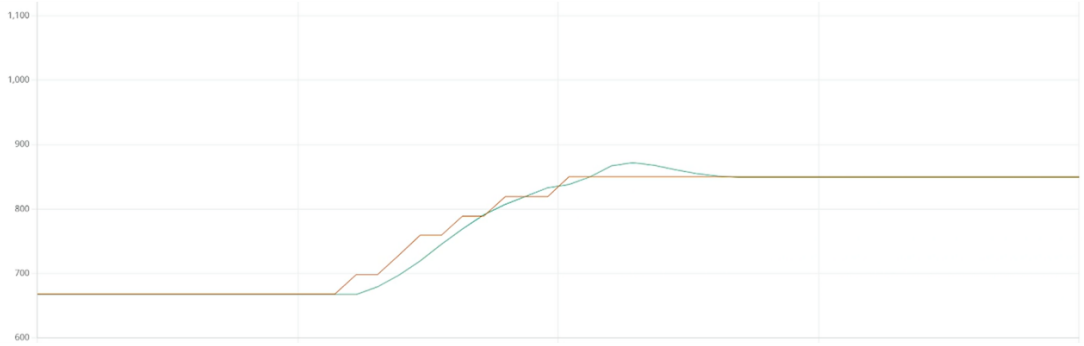
Với bộ điều khiển PID, hệ thống cho thấy độ chính xác cao trong quá trình hoạt động.



Hình 2.15: Hoạt động của hệ thống với bộ điều khiển PID (1)

Trong trường hợp giá trị PV chưa bằng SP, thành phần tích phân sẽ tích lũy giá trị sai số và tiếp tục điều khiển động cơ sao cho hai giá trị này bằng nhau. Điều này sẽ giúp cho hệ thống hoạt động với độ chính xác cao.

Tuy vậy, việc tích lũy tích phân sai số cũng sẽ khiến hệ thống mất đi tính ổn định, trong một số trường hợp động cơ còn xảy ra hiện tượng dao động. Nguyên nhân là do ba hằng số độ lợi chưa hoàn toàn tối ưu với hệ thống.



Hình 2.16: Hoạt động của hệ thống với bộ điều khiển PID (2)

Chương 3 KẾT QUẢ

3.1 Kết luận chung

Sau thời gian tìm hiểu về lý thuyết của bộ điều khiển PID cùng với ứng dụng của bộ điều khiển này vào mô hình điều khiển vị trí góc quay động cơ DC Encoder, báo cáo này đã đạt được một số kết quả.

3.1.1 Bộ điều khiển PID

- Lịch sử hình thành, cơ cấu hoạt động của thuật toán điều khiển PID.
- Hiểu rõ về bộ điều khiển PID dạng song song, các thành phần và vai trò của chúng trong bộ điều khiển.
- Tìm hiểu về một số phương pháp tìm độ lợi K_p K_i K_d giúp tối ưu độ ổn định của các hệ thống trong quá trình hoạt động.
- Tính hiệu quả, độ tin cậy của bộ điều khiển PID, các ứng dụng của bộ điều khiển này trong đời sống con người.
- Xây dựng một số mô hình sử dụng thuật toán PID nhằm chứng minh cơ cấu hoạt động, tính hiệu quả của thuật toán.

3.1.2 Các mô hình thực tế sử dụng thuật toán PID

Mô hình sử dụng thuật toán PID đều hoạt động một cách ổn định. Phản hồi của hệ thống nhanh khi bị tác động bởi các yếu tố khách quan. Bộ điều khiển này đã chứng minh được tính ổn định và tối ưu của nó trong việc điều khiển một số hệ thống. Các hệ thống trong quá trình hoạt động đều đạt được tính ổn định cao, đồng thời cho thấy tính đơn giản và hiệu quả của thuật toán PID.

3.1.3 Các chức năng của vi điều khiển

- Học được kỹ năng đọc datasheet của linh kiện cần sử dụng.
- Hiểu về lập trình nhúng AVR, chức năng của các thanh ghi trong vi điều khiển ATmega328p.
- Thành thạo trong việc sử dụng các chức năng khác của Timer như: tạo xung PWM. Sử dụng ngắt để xử lý các tình huống trong lập trình vi điều khiển.
- Hiểu rõ và ứng dụng các giao thức được sử dụng phổ biến như TWI, USART...
- Học được cách viết chương trình nhằm tối ưu hóa hoạt động của vi điều khiển.

3.2 Một số vấn đề còn tồn tại của đề tài

Cùng với các kết quả đạt được, đề tài này còn tồn tại một số vấn đề. Mô hình và bộ điều khiển PID còn ở dạng đơn giản. Chưa mô hình hóa được hệ thống tổng thể và phương trình toán học cụ thể cho từng mô hình. Ngoài ra, chưa tìm hiểu và khảo sát các phương pháp tìm độ lợi khác.

3.3 Hướng phát triển của đề tài

Bên cạnh kết quả thu được và một số hạn chế, đề tài này vẫn có thể được phát triển hơn nữa trong tương lai. Nâng cấp các cảm biến đang sử dụng trong các mô hình, sử dụng các chip xử lí tốc độ cao nhằm tối ưu hơn nữa quá trình tính toán và các hệ thống. Tìm hiểu về một số phương thức truyền dữ liệu không dây để có thể xử lý hệ thống. Nghiên cứu sâu hơn về bộ điều khiển PID, các dạng khác của bộ điều khiển này như: Dạng Laplace, dạng nối tiếp/ tương hỗ... Có thể biểu diễn các mô hình dưới dạng phương trình toán học. Sử dụng thuật toán PID trong các mô hình lớn hơn như: điều khiển lò nhiệt, chế tạo drone, xe segway...

Tài liệu tham khảo

[1] Karl Johan Åström and Tore Hägglund. "The future of PID control". In: *Control engineering practice* 9.11 (2001), pp. 1163–1175.

[2] Karl Johan Åström. "Control system design lecture notes for me 155a". In: *Department of Mechanical and Environmental Engineering University of California Santa Barbara* 333 (2002).

[3] Rakesh P Borase, DK Maghade, SY Sondkar, et al. "A review of PID control, tuning methods and applications". In: *International Journal of Dynamics and Control* 9 (2021), pp. 818–827.

[4] Tan Kok Kiong, Wang Qing-Guo, Hang Chang Chieh, et al. *Advances in PID control*. Springer, 1999.

[5] Feng Lin, Robert D Brandt, and George Saikalis. "Self-tuning of PID controllers by adaptive interaction". In: *Proceedings of the 2000 American Control Conference. ACC (IEEE Cat. No. 00CH36334)*. Vol. 5. IEEE. 2000, pp. 3676–3681.

[6] Ramon Vilanova and Antonio Visioli. *PID control in the third millennium*. Vol. 75. 417. Springer, 2012.

[7] Sara Zak Ardemis Boghossian James Brown. "P, I, D, PI, PD, and PID control". In: *University of Michigan* (). URL: https://eng.libretexts.org.

[8] Prof. Bill Messner Prof. Dawn Tilbury. "Control Tutorials for MATLAB and Simulink". In: *University of Michigan, Carnegie Mellon* (). URL: https://ctms.engin.umich.edu/CTMS/index.php?aux=Home.

Phụ lục A Các chương trình của đề tài

A.1 Chương trình chính (main.c)

```
/*
* GccApplication1.c
*
* Created: 5/23/2024 3:04:35 PM
* Author : letung
*/
#define F_CPU 16000000UL
#define deltaT 0.04
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <stdio.h>
#include <math.h>
#include <stdbool.h>
#include "uart.h"
#include "ADCLib.h"
#include "hd44780pcf8574.h"


char addr = PCF8574_ADDRESS;  //0x27
float ADC_value = 0;
float setpoint = 0;
float pos = 0;

int b;
char str_sp[5];
char str_pv[5];
char str_angle[5];

long prevT = 0;
float eprev = 0;
float eintegral = 0;

bool Saturation = false;

// PID parameters
float kp = 6.75;
float ki = 12.14;
//float ki = 0;
float kd = 0.22;

ISR(INT0_vect){
    b = PIND & (1 << PD3);
    if (b) pos++;
    else pos--;
}
void setMotor(int dir, int pwmVal) {
    OCR0B = pwmVal;
    if (dir == 1) {
        PORTD |= (1 << PD7);
        PORTD &= ~(1 << PD6);
    } else if (dir == -1) {
        PORTD &= ~(1 << PD7);
        PORTD |= (1 << PD6);
    } else {
        PORTD &= ~(1 << PD7);
        PORTD &= ~(1 << PD6);
    }
```

```
59
60  }
61  int main(void)
62  {
63      HD44780_PCF8574_Init(addr);
64      HD44780_PCF8574_DisplayClear(addr);
65      HD44780_PCF8574_DisplayOn(addr);
66      HD44780_PCF8574_DrawString(addr, "Embedded System");
67      _delay_ms(1000);
68      HD44780_PCF8574_DisplayClear(addr);
69      HD44780_PCF8574_PositionXY(addr, 0, 0);
70      HD44780_PCF8574_DrawString(addr,"SP:");
71      HD44780_PCF8574_PositionXY(addr, 0, 1);
72      HD44780_PCF8574_DrawString(addr,"PV:");
73      //_delay_ms(100);
74      uart_init(9600);
75      ADC_init(0);
76
77      DDRD |= (1 << PD5) | (1 << PD6) | (1 << PD7);
78      TCCR0A |= (1 << WGM00) | (1 << WGM01) | (1 << COM0B1); // Fast PWM, non-
            inverting mode
79      TCCR0B |= (1 << CS00) | (1 << CS01); // Prescaler 64
80
81      EICRA=(0<<ISC11) | (0<<ISC10) | (1<<ISC01) | (1<<ISC00);
82      EIMSK=(0<<INT1) | (1<<INT0);
83      EIFR=(0<<INTF1) | (1<<INTF0);
84      sei();
85
86
87
88      while (1)
89      {
90
91          read_ADC();
92          ADC_value = ADCW;
93          ADC_value = 10*round(ADC_value*36/1023);
94          setpoint = round((11 * 99.3 * ADC_value)/360);
95          int e = setpoint - pos;
96          float dedt = (e - eprev) / deltaT;
97          if (!Saturation) {
98              eintegral += (e + eprev) * deltaT / 2;
99          }
100         // PID calculation
101         float u = kp * e + ki * eintegral + kd * dedt;
102
103         float pwr = fabs(u);
104         if (pwr > 255) {
105             pwr = 255;
106             Saturation = true;
107         } else {
108             Saturation = false;
109         }
110
111         // Motor direction
112         int dir;
113         if (u < 0) {
114             dir = -1;
115         } else {
116             dir = 1;
117         }
118
119         // Set motor
120         setMotor(dir, pwr);
121
122         // Store previous error
123         eprev = e;
124
```

```
        sprintf(str_angle, "%4.0f",ADC_value);
        sprintf(str_sp, "%4.0f",setpoint);
        sprintf(str_pv, "%4.0f",pos);
        HD44780_PCF8574_PositionXY(addr, 4, 0);
        HD44780_PCF8574_DrawString(addr,str_angle);
        HD44780_PCF8574_PositionXY(addr, 10, 0);
        HD44780_PCF8574_DrawString(addr,str_sp);
        HD44780_PCF8574_PositionXY(addr, 10, 1);
        HD44780_PCF8574_DrawString(addr,str_pv);


        uart_putstring("Setpoint:");
        uart_putstring(str_sp);
        uart_putstring("    ");
        uart_putstring(" PV:");
        uart_putstring(str_pv);
        uart_putstring("\n");
    }
}
```

A.2 Thư viện ADCLib.h

```

    #include <avr/io.h>
    #include <util/delay.h>
    #include <stdlib.h>
    #include <stdio.h>
    #include <avr/interrupt.h>


    void ADC_init(unsigned int pin){

        ADMUX = 0b01000000 | pin;
        ADCSRA = 0b10000111;
        DIDR0 |= 0x01 << pin;
        _delay_ms(10);
    }

    int read_ADC(){
        ADCSRA |= 0b01000000;
        while((ADCSRA & 0b00010000) == 0);
        ADCSRA |= 0b00010000;
        return ADCW;
    }
```

A.3 Thư viện uart.h

```
#ifndef _UARTLIB_
#define _UARTLIB_
#define fosc 16000000
#include <avr/io.h>
//#include <avr/interrupt.h>
#include <util/delay.h>
void uart_init(unsigned int BAUDRATE)
{
    //Config BAUD Rate
    unsigned int n = fosc/BAUDRATE/16 - 1;
    UBRR0H = n>>8;
    UBRR0L = n;
    //Config mode and data frame
    //Asynchronous mode, 8 data bit, 1 stop bit, no Parity
    UCSR0C = 0b00000110;
```

```
    //Enable transmiter and receiver, RX interupt
    UCSR0B = 0b10011000;
    sei();
}
void uart_putchar(unsigned char data)
{
    while (!(UCSR0A & 0b00100000)); //wait for data register empty
    UDR0 = data;
}
void uart_putstring(char *str)
{
    while (*str)
    {
        uart_putchar(*str);
        //if see the line feed, add carriage return
        if (*str == '\n')
        uart_putchar('\r');
        str++;
    }
}
void uart_put_int(unsigned int value)
{
    unsigned char buf[8];
    int index = 0,i,j;
    j = value;
    do {
        buf[index] = j%10 + 48;//chuyen gia tri sang ki tu
        j = j/10;
        index +=1;
    } while(j);

    for (i = index; i>0; i--)
    uart_putchar(buf[i-1]);
}

#endif
```

A.4 Thư viện TWI.h

```
    #include <stdio.h>
    #include <avr/io.h>

    #ifndef __TWI_H__
    #define __TWI_H__

    // define register for TWI communication
    #if defined(__AVR_ATmega16__) || defined(__AVR_ATmega328P__)
    #define TWI_TWAR TWAR // TWI (Slave) Address Register
    #define TWI_TWBR TWBR // TWI Bit Rate Register
    #define TWI_TWDR TWDR // TWI Data Register
    #define TWI_TWCR TWCR // TWI Control Register
    #define TWI_TWSR TWSR // TWI Status Register
    #endif

    // TWI status mask
    #define TWI_STATUS                    (TWI_TWSR & 0xF8)
    #define TWI_WRITE                     0
    #define TWI_READ                      1

    // TWI CLK frequency
    //  @param TWBR
    //  @param Prescaler
    //    TWPS1 TWPS0  - PRESCALER
    //      0     0    -     1
```

```
26      //      0     1     -     4
27      //      1     0     -    16
28      //      1     1     -    64
29      #define TWI_FREQ(BIT_RATE, PRESCALER) { TWI_TWBR = BIT_RATE; TWI_TWSR |= (
            TWI_TWSR & 0x03) | PRESCALER; }
30
31      // TWI start condition
32      // (1 <<  TWEN) - TWI Enable
33      // (1 << TWINT) - TWI Interrupt Flag - must be cleared by set
34      // (1 << TWSTA) - TWI Start
35      #define TWI_START()                   { TWI_TWCR = (1 << TWEN) | (1 << TWINT) |
            (1 << TWSTA); }
36
37      // TWI MASTER enable with NACK
38      // (1 <<  TWEN) - TWI Enable
39      // (1 << TWINT) - TWI Interrupt Flag - must be cleared by set
40      #define TWI_MSTR_ENABLE_NACK()        { TWI_TWCR = (1 << TWEN) | (1 << TWINT); }
41
42      // TWI MASTER enable with ACK
43      // (1 <<  TWEN) - TWI Enable
44      // (1 << TWINT) - TWI Interrupt Flag - must be cleared by set
45      // (1 <<  TWEA) - TWI Master Receiver will return ACK
46      #define TWI_MSTR_ENABLE_ACK()         { TWI_TWCR = (1 << TWEN) | (1 << TWINT) |
            (1 << TWEA); }
47
48      // TWI stop condition
49      // (1 <<  TWEN) - TWI Enable
50      // (1 << TWINT) - TWI Interrupt Flag - must be cleared by set
51      // (1 << TWSTO) - TWI Stop
52      #define TWI_STOP()                    { TWI_TWCR = (1 << TWEN) | (1 << TWINT) |
            (1 << TWSTO); }
53
54      // TWI test if TWINT Flag is set
55      #define TWI_WAIT_TILL_TWINT_IS_SET()  { while (!(TWI_TWCR & (1 << TWINT))); }
56
57      // definitions
58      #define TWI_STATUS_INIT       0xFF
59      #define TWI_SUCCESS              0
60      #define TWI_ERROR                1
61      #define TWI_ERROR_NONE           0
62
63      // ++++++++++++++++++++++++++++++++++++++++
64      //
65      //        M A S T E R   M O D E
66      //
67      // ++++++++++++++++++++++++++++++++++++++++
68      // Master Mode - Transmitter / Receiver
69      #define TWI_START_ACK         0x08  // A START condition has been transmitted
70      #define TWI_REP_START_ACK     0x10  // A repeated START condition has been
            transmitted
71      #define TWI_FLAG_ARB_LOST     0x38  // Arbitration lost in SLA+W or NOT ACK bit
72      // Master Transmitter Mode
73      #define TWI_MT_SLAW_ACK       0x18  // SLA+W has been transmitted; ACK has been
            received
74      #define TWI_MT_SLAW_NACK      0x20  // SLA+W has been transmitted; NOT ACK has
            been received
75      #define TWI_MT_DATA_ACK       0x28  // Data byte has been transmitted; ACK has
            been received
76      #define TWI_MT_DATA_NACK      0x30  // Data byte has been transmitted; NOT ACK
            has been received
77      // Master Receiver Mode
78      #define TWI_MR_SLAR_ACK       0x40  // SLA+R has been transmitted; ACK has been
            received
79      #define TWI_MR_SLAR_NACK      0x48  // SLA+R has been transmitted; NOT ACK has
            been received
80      #define TWI_MR_DATA_ACK       0x50  // Data byte has been received; ACK has been
             received
```

```
81  #define TWI_MR_DATA_NACK      0x58  // Data byte has been received; NOT ACK has
        been received
82
83  // ++++++++++++++++++++++++++++++++++++++++
84  //
85  //         S L A V E   M O D E
86  //
87  // ++++++++++++++++++++++++++++++++++++++++
88  // Slave Receiver Mode
89  #define TWI_SR_SLAW_ACK       0x60  // Own Slave address has been received; ACK
        returned
90  #define TWI_SR_ALMOA_ACK      0x68  // Arbitration Lost in SLA+R/W as Master;
        Own Slave address has been received; ACK returned
91  #define TWI_SR_GCALL_ACK      0x70  // General call address has been received;
        ACK returned
92  #define TWI_SR_ALMGA_ACK      0x78  // Arbitration lost in SLA+R/W as Master;
        General call address has been received; ACK returned
93  #define TWI_SR_OA_DATA_ACK    0x80  // Previously addressed with own SLA+W; data
         has been received; ACK returned
94  #define TWI_SR_OA_DATA_NACK   0x88  // Previously addressed with own SLA+W; data
         has been received; NOT ACK returned
95  #define TWI_SR_GC_DATA_ACK    0x90  // Previously addressed with general call;
        data has been received; ACK returned
96  #define TWI_SR_GC_DATA_NACK   0x98  // Previously addressed with general call;
        data has been received; NOT ACK returned
97  #define TWI_SR_STOP_RSTART    0xA0  // A STOP condition or repeated START
        condition has been received while still addressed as Slave
98  // Slave Transmitter Mode
99  #define TWI_ST_OA_ACK         0xA8  // Own SLA+R has been received; ACK has been
         returned
100 #define TWI_ST_ALMOA_ACK      0xB0  // Arbitration lost in SLA+R/W as Master;
        own SLA+R has been received; ACK has been received
101 #define TWI_ST_DATA_ACK       0xB8  // Data byte in TWDR has been transmitted;
        ACK has been received
102 #define TWI_ST_DATA_NACK      0xC0  // Data byte in TWDR has been transmitted;
        NOT ACK has been received
103 #define TWI_ST_DATA_LOST_ACK  0xC8  // Last data byte in TWDR has been
        transmitted (TWEA = '0'); ACK has been received
104
105
106 extern char _twi_error_stat;
107
108
109 void TWI_Init (void);
110
111
112 void TWI_MT_Start (void);
113
114
115 void TWI_Transmit_SLAW (char);
116
117
118 void TWI_Transmit_SLAR (char);
119
120
121 void TWI_Transmit_Byte (char);
122
123
124 char TWI_Receive_Byte (void);
125
126
127 void TWI_Stop (void);
128
129 void TWI_Error (char, char);
130
131 #endif
```

A.5 Thư viện TWI.c

```
#include "twi.h"

/* @var error status */
char _twi_error_stat = TWI_ERROR_NONE;

/**
* @desc    TWI init - initialize frequency
*
* @param   void
*
* @return  void
*/
void TWI_Init(void)
{
    // ++++++++++++++++++++++++++++++++++++++++++++
    // Calculation fclk:
    //
    // fclk = (fcpu)/(16+2*TWBR*4^Prescaler)
    // ---------------------------------------------
    // Calculation TWBR:
    //
    // TWBR = {(fcpu/fclk) - 16 } / (2*4^Prescaler)
    // ++++++++++++++++++++++++++++++++++++++++++++
    // @16MHz
    // @param1 value of TWBR,
    //    fclk = 400 kHz; TWBR = 3
    //    fclk = 200 kHz; TWBR = 8
    //    fclk = 100 kHz; TWBR = 18
    // @8MHz
    // @param1 value of TWBR,
    //    fclk = 200 kHz; TWBR = 3
    //    fclk = 100 kHz; TWBR = 8
    // @param2 value of Prescaler = 1
    TWI_FREQ(8, 1);
}

/**
* @desc    TWI MT Start
*
* @param   void
*
* @return  void
*/
void TWI_MT_Start(void)
{
    // init status
    char status = TWI_STATUS_INIT;
    // START
    // ---------------------------------------------
    // request for bus
    TWI_START();
    // wait till flag set
    TWI_WAIT_TILL_TWINT_IS_SET();
    // status read
    status = TWI_STATUS;
    // test if start or repeated start acknowledged
    if ((status != TWI_START_ACK) && (status != TWI_REP_START_ACK)) {
        // error status
        TWI_Error(status, TWI_START_ACK);
    }
}

/**
```

```
65      * @desc    TWI Send address + write
66      *
67      * @param   char
68      *
69      * @return  void
70      */
71     void TWI_Transmit_SLAW(char address)
72     {
73         // init status
74         char status = TWI_STATUS_INIT;
75         // SLA+W
76         // ---------------------------------------------
77         TWI_TWDR = (address << 1);
78         // enable
79         TWI_MSTR_ENABLE_ACK();
80         // wait till flag set
81         TWI_WAIT_TILL_TWINT_IS_SET();
82         // status read
83         status = TWI_STATUS;
84         // find
85         if (status != TWI_MT_SLAW_ACK) {
86             // error status
87             TWI_Error(status, TWI_MT_SLAW_ACK);
88         }
89     }
90
91     /**
92     * @desc    TWI Send address + read
93     *
94     * @param   char
95     *
96     * @return  void
97     */
98     void TWI_Transmit_SLAR(char address)
99     {
100        // init status
101        char status = TWI_STATUS_INIT;
102        // SLA+R
103        // ---------------------------------------------
104        TWI_TWDR = (address << 1) | TWI_READ;
105        // enable
106        TWI_MSTR_ENABLE_ACK();
107        // wait till flag set
108        TWI_WAIT_TILL_TWINT_IS_SET();
109        // status read
110        status = TWI_STATUS;
111        // find
112        if (status != TWI_MR_SLAR_ACK) {
113            // error status
114            TWI_Error(status, TWI_MR_SLAR_ACK);
115        }
116    }
117
118    /**
119    * @desc    TWI Transmit data
120    *
121    * @param   char
122    *
123    * @return  void
124    */
125    void TWI_Transmit_Byte(char data)
126    {
127        // init status
128        char status = TWI_STATUS_INIT;
129        // DATA SEND
130        // ---------------------------------------------
131        TWI_TWDR = data;
```

```
        // enable
        TWI_MSTR_ENABLE_ACK();
        // wait till flag set
        TWI_WAIT_TILL_TWINT_IS_SET();
        // status read
        status = TWI_STATUS;
        // send with success
        if (status != TWI_MT_DATA_ACK) {
            // error status
            TWI_Error(status, TWI_MT_DATA_ACK);
        }
    }

    /**
    * @desc    TWI Receive 1 byte
    *
    * @param   void
    *
    * @return  char
    */
    char TWI_Receive_Byte(void)
    {
        // init status
        char status = TWI_STATUS_INIT;
        // DATA RECEIVE
        // ---------------------------------------------
        // enable with NACK
        TWI_MSTR_ENABLE_NACK();
        // wait till flag set
        TWI_WAIT_TILL_TWINT_IS_SET();
        // status read
        status = TWI_STATUS;
        // send with success
        if (status != TWI_MR_DATA_NACK) {
            // error status
            TWI_Error(status, TWI_MR_DATA_NACK);
        }
        // received data
        return TWI_TWDR;
    }


    void TWI_Stop(void)
    {
        // End TWI
        // -------------------------------------------------
        // send stop sequence
        TWI_STOP();
        // wait for TWINT flag is set
        //  TWI_WAIT_TILL_TWINT_IS_SET();
    }


    void TWI_Error(char status, char expected)
    {

    }
```

A.6 Thư viện LCD I2C - hd44780pcf8574.h

```
    #ifndef __HD44780PCF8574_H__
    #define __HD44780PCF8574_H__

```

```
#include <avr/io.h>
#include <avr/pgmspace.h>

#define PCF8574_SUCCESS         0
#define PCF8574_ERROR           1
#define PCF8574_ADDRESS      0x27


#define PCF8574_PIN_RS       0x01
#define PCF8574_PIN_RW       0x02
#define PCF8574_PIN_E        0x04
#define PCF8574_PIN_P3       0x08
#define PCF8574_PIN_DB4      0x10
#define PCF8574_PIN_DB5      0x20
#define PCF8574_PIN_DB6      0x40
#define PCF8574_PIN_DB7      0x80

#define HD44780_BUSY_FLAG    HD44780_DB7
#define HD44780_INIT_SEQ     0x30
#define HD44780_DISP_CLEAR   0x01
#define HD44780_DISP_OFF     0x08
#define HD44780_DISP_ON      0x0C
#define HD44780_CURSOR_ON    0x0E
#define HD44780_CURSOR_BLINK 0x0F
#define HD44780_RETURN_HOME  0x02
#define HD44780_ENTRY_MODE   0x06
#define HD44780_4BIT_MODE    0x20
#define HD44780_8BIT_MODE    0x30
#define HD44780_2_ROWS       0x08
#define HD44780_FONT_5x8     0x00
#define HD44780_FONT_5x10    0x04
#define HD44780_POSITION     0x80

#define HD44780_SHIFT        0x10
#define HD44780_CURSOR       0x00
#define HD44780_DISPLAY      0x08
#define HD44780_LEFT         0x00
#define HD44780_RIGHT        0x04

#define HD44780_ROWS         2
#define HD44780_COLS         16

#define HD44780_ROW1_START   0x00
#define HD44780_ROW1_END     HD44780_COLS
#define HD44780_ROW2_START   0x40
#define HD44780_ROW2_END     HD44780_COLS

// set bit
#define SETBIT(REG, BIT) { REG |= (1 << BIT); }
// clear bit
#define CLRBIT(REG, BIT) { REG &= ~(1 << BIT); }
// set port / pin if bit is set
#define SET_IF_BIT_IS_SET(REG, PORT, DATA, BIT) { if((DATA & BIT) > 0) { SETBIT(
    REG, PORT); } }

char HD44780_PCF8574_Init (char);

void HD44780_PCF8574_E_pulse (char);

void HD44780_PCF8574_SendInstruction (char, char);

void HD44780_PCF8574_SendData (char, char);

void HD44780_PCF8574_CheckBF (char);

void HD44780_PCF8574_Send_4bits_M4b_I (char);

```

```
71  void HD44780_PCF8574_Send_8bits_M4b_I (char, char, char);
72
73  void HD44780_PCF8574_DisplayClear (char);
74
75  void HD44780_PCF8574_DisplayOn (char);
76
77  void HD44780_PCF8574_CursorOn (char);
78
79  void HD44780_PCF8574_CursorBlink (char);
80
81  void HD44780_PCF8574_DrawChar (char, char);
82
83  void HD44780_PCF8574_DrawString (char, char *);
84
85  char HD44780_PCF8574_PositionXY (char, char, char);
86
87  char HD44780_PCF8574_Shift (char, char, char);
88
89  #endif
```

A.7 Thư viện LCD I2C - hd44780pcf8574.c

```
1   #include <stdio.h>
2   #include <util/delay.h>
3   #include <avr/io.h>
4   #include "twi.h"
5   #include "hd44780pcf8574.h"
6
7   // +---------------------------+
8   // |         Power on          |
9   // | Wait for more than 15 ms  |   // 15 ms wait
10  // | after VCC rises to 4.5 V  |
11  // +---------------------------+
12  //               |
13  // +---------------------------+
14  // |  RS R/W DB7 DB6 DB5 DB4   |
15  // |   0   0   0   0   1   1   |   // Initial sequence 0x30
16  // | Wait for more than 4.1 ms |   // 4.1 ms us writing DATA into DDRAM or CGRAM
17  // +---------------------------+
18  //               |
19  // +---------------------------+
20  // |  RS R/W DB7 DB6 DB5 DB4   |
21  // |   0   0   0   0   1   1   |   // Initial sequence 0x30
22  // | Wait for more than 0.1 ms |   // 100 us writing DATA into DDRAM or CGRAM
23  // +---------------------------+
24  //               |
25  // +---------------------------+
26  // |  RS R/W DB7 DB6 DB5 DB4   |   // Initial sequence 0x30
27  // |   0   0   0   0   1   1   |   // 37 us writing DATA into DDRAM or CGRAM 4
        us tadd - time after busy flag disapeared
28  // | Wait for more than 45 us  |   // 37 us + 4 us = 41 us * (270/250) = 45us
29  // +---------------------------+
30  //               |
31  // +---------------------------+   // 4bit mode 0x20 !!! MUST BE SET TIME, BF
        CHECK DOESN'T WORK CORRECTLY !!!
32  // |  RS R/W DB7 DB6 DB5 DB4   |   //
33  // |   0   0   0   0   1   0   |   // 37 us writing DATA into DDRAM or CGRAM 4
        us tadd - time after busy flag disapeared
34  // | Wait for more than 45 us  |   // !!! MUST BE SET DELAY TIME, BUSY FLAG
        CHECK DOESN'T WORK CORRECTLY !!!
35  // +---------------------------+
36  //               |
37  // +---------------------------+
38  // |  RS R/W DB7 DB6 DB5 DB4   |   // Display off 0x28
```

```
    // |   0   0   0   0   1   0   |   //
    // |   0   0   1   0   0   0   |   //
    // |    Wait for BF Cleared    |   // Wait for 50us
    // +---------------------------+
    //               |
    // +---------------------------+
    // |  RS R/W DB7 DB6 DB5 DB4   |   // Display clear 0x01
    // |   0   0   0   0   0   0   |   //
    // |   0   0   0   0   0   1   |   //
    // |    Wait for BF Cleared    |   // Wait for 50us
    // +---------------------------+
    //               |
    // +---------------------------+
    // |  RS R/W DB7 DB6 DB5 DB4   |   // Entry mode set 0x06
    // |   0   0   0   0   0   0   |   //
    // |   0   0   0   1   1   0   |   // shift cursor to the left, without text
        shifting
    // |    Wait for BF Cleared    |   // Wait for 50us
    // +---------------------------+

    /**
    * @desc    LCD init - initialisation routine
    *
    * @param   char
    *
    * @return  char
    */
    char HD44780_PCF8574_Init (char addr)
    {
        // delay > 15ms
        _delay_ms(16);

        // Init TWI
        TWI_Init();

        // TWI: start
        // -------------------------
        TWI_MT_Start();

        // TWI: send SLAW
        // -------------------------
        TWI_Transmit_SLAW(addr);

        // DB7 BD6 DB5 DB4 P3 E RW RS
        // DB4=1, DB5=1 / BF cannot be checked in these instructions
        // ------------------------------------------------------------------
        HD44780_PCF8574_Send_4bits_M4b_I(PCF8574_PIN_DB4 | PCF8574_PIN_DB5);
        // delay > 4.1ms
        _delay_ms(5);

        // DB4=1, DB5=1 / BF cannot be checked in these instructions
        // ------------------------------------------------------------------
        HD44780_PCF8574_Send_4bits_M4b_I(PCF8574_PIN_DB4 | PCF8574_PIN_DB5);
        // delay > 100us
        _delay_us(110);

        // DB4=1, DB5=1 / BF cannot be checked in these instructions
        // ------------------------------------------------------------------
        HD44780_PCF8574_Send_4bits_M4b_I(PCF8574_PIN_DB4 | PCF8574_PIN_DB5);
        // delay > 45us (=37+4 * 270/250)
        _delay_us(50);

        // DB5=1 / 4 bit mode 0x20 / BF cannot be checked in these instructions
        // ------------------------------------------------------------------
        HD44780_PCF8574_Send_4bits_M4b_I(PCF8574_PIN_DB5);
        // delay > 45us (=37+4 * 270/250)
        _delay_us(50);
```

```
105
106         // TWI Stop
107         TWI_Stop();
108
109         // 4 bit mode, 2 rows, font 5x8
110         HD44780_PCF8574_SendInstruction(addr, HD44780_4BIT_MODE | HD44780_2_ROWS |
                HD44780_FONT_5x8);
111
112         // display off 0x08 - send 8 bits in 4 bit mode
113         HD44780_PCF8574_SendInstruction(addr, HD44780_DISP_OFF);
114
115         // display clear 0x01 - send 8 bits in 4 bit mode
116         HD44780_PCF8574_SendInstruction(addr, HD44780_DISP_CLEAR);
117
118         // entry mode set 0x06 - send 8 bits in 4 bit mode
119         HD44780_PCF8574_SendInstruction(addr, HD44780_ENTRY_MODE);
120
121         // return success
122         return PCF8574_SUCCESS;
123     }
124
125     /**
126     * @desc    LCD E pulse
127     *
128     * @param   char
129     *
130     * @return  void
131     */
132     void HD44780_PCF8574_E_pulse (char data)
133     {
134         // E pulse
135         // -----------------------------------
136         TWI_Transmit_Byte(data | PCF8574_PIN_E);
137         // PWeh delay time > 450ns
138         _delay_us(0.5);
139         // E down
140         TWI_Transmit_Byte(data & ~PCF8574_PIN_E);
141         // PWeh delay time > 450ns
142         _delay_us(0.5);
143     }
144
145     /**
146     * @desc    LCD send 4bits in 4 bit mode
147     *
148     * @param   char
149     *
150     * @return  void
151     */
152     void HD44780_PCF8574_Send_4bits_M4b_I (char data)
153     {
154         // Send upper nibble, E up
155         // -----------------------------------
156         TWI_Transmit_Byte(data);
157         // E pulse
158         HD44780_PCF8574_E_pulse(data);
159     }
160
161     /**
162     * @desc    LCD send 8bits in 4 bit mode
163     *
164     * @param   char
165     * @param   char
166     *
167     * @return  void
168     */
169     void HD44780_PCF8574_Send_8bits_M4b_I (char addr, char data, char annex)
170     {
```

```
171         // upper nible with backlight
172         char up_nibble = (data & 0xF0) | annex;
173         // lower nibble with backlight
174         char low_nibble = (data << 4) | annex;
175 
176         // TWI: start
177         // ------------------------
178         TWI_MT_Start();
179 
180         // TWI: send SLAW
181         // ------------------------
182         TWI_Transmit_SLAW(addr);
183 
184         // Send upper nibble, E up
185         // ---------------------------------
186         TWI_Transmit_Byte(up_nibble);
187         // E pulse
188         HD44780_PCF8574_E_pulse(up_nibble);
189 
190         // Send lower nibble, E up
191         // ---------------------------------
192         TWI_Transmit_Byte(low_nibble);
193         // E pulse
194         HD44780_PCF8574_E_pulse(low_nibble);
195 
196         // TWI Stop
197         TWI_Stop();
198     }
199 
200     /**
201     * @desc    LCD check BF
202     *
203     * @param   char
204     *
205     * @return  void
206     */
207     void HD44780_PCF8574_CheckBF (char addr)
208     {
209     }
210 
211     /**
212     * @desc    LCD Send instruction 8 bits in 4 bits mode
213     *
214     * @param   char
215     * @param   char
216     *
217     * @return  void
218     */
219     void HD44780_PCF8574_SendInstruction (char addr, char instruction)
220     {
221         // send instruction
222         HD44780_PCF8574_Send_8bits_M4b_I(addr, instruction, PCF8574_PIN_P3);
223         // check BF
224         //HD44780_PCF8574_CheckBF(addr);
225         _delay_ms(50);
226     }
227 
228     /**
229     * @desc    LCD Send data 8 bits in 4 bits mode
230     *
231     * @param   char
232     * @param   char
233     *
234     * @return  void
235     */
236     void HD44780_PCF8574_SendData (char addr, char data)
237     {
```

```
238         // send data
239         // data/command -> pin RS High
240         // backlight -> pin P3
241         HD44780_PCF8574_Send_8bits_M4b_I(addr, data, PCF8574_PIN_RS | PCF8574_PIN_P3
                );
242         // check BF
243         //HD44780_PCF8574_CheckBF(addr);
244         //_delay_ms(50);
245     }
246 
247     /**
248     * @desc    LCD Go to position x, y
249     *
250     * @param   char
251     * @param   char
252     * @param   char
253     *
254     * @return  char
255     */
256     char HD44780_PCF8574_PositionXY (char addr, char x, char y)
257     {
258         if (x > HD44780_COLS || y > HD44780_ROWS) {
259             // error
260             return PCF8574_ERROR;
261         }
262         // check which row
263         if (y == 0) {
264             // send instruction 1st row
265             HD44780_PCF8574_SendInstruction(addr, (HD44780_POSITION | (
                    HD44780_ROW1_START + x)));
266         } else if (y == 1) {
267             // send instruction 2nd row
268             HD44780_PCF8574_SendInstruction(addr, (HD44780_POSITION | (
                    HD44780_ROW2_START + x)));
269         }
270         // success
271         return PCF8574_SUCCESS;
272     }
273 
274     /**
275     * @desc    LCD display clear
276     *
277     * @param   char
278     *
279     * @return  void
280     */
281     void HD44780_PCF8574_DisplayClear (char addr)
282     {
283         // Diplay clear
284         HD44780_PCF8574_SendInstruction(addr, HD44780_DISP_CLEAR);
285     }
286 
287     /**
288     * @desc    LCD display on
289     *
290     * @param   char
291     *
292     * @return  void
293     */
294     void HD44780_PCF8574_DisplayOn (char addr)
295     {
296         // send instruction - display on
297         HD44780_PCF8574_SendInstruction(addr, HD44780_DISP_ON);
298     }
299 
300     /**
301     * @desc    LCD cursor on, display on
```

```
302     *
303     * @param   char
304     *
305     * @return  void
306     */
307    void HD44780_PCF8574_CursorOn (char addr)
308    {
309        // send instruction - cursor on
310        HD44780_PCF8574_SendInstruction(addr, HD44780_CURSOR_ON);
311    }
312
313    /**
314    * @desc    LCD cursor blink, cursor on, display on
315    *
316    * @param   char
317    *
318    * @return  void
319    */
320    void HD44780_PCF8574_CursorBlink (char addr)
321    {
322        // send instruction - Cursor blink
323        HD44780_PCF8574_SendInstruction(addr, HD44780_CURSOR_BLINK);
324    }
325
326    /**
327    * @desc    LCD draw char
328    *
329    * @param   char
330    * @param   char
331    *
332    * @return  void
333    */
334    void HD44780_PCF8574_DrawChar (char addr, char character)
335    {
336        // Draw character
337        HD44780_PCF8574_SendData(addr, character);
338    }
339
340    /**
341    * @desc    LCD draw string
342    *
343    * @param   char
344    * @param   char *
345    *
346    * @return  void
347    */
348    void HD44780_PCF8574_DrawString (char addr, char *str)
349    {
350        unsigned short int i = 0;
351        // loop through chars
352        while (str[i] != '\0') {
353            // draw individual chars
354            HD44780_PCF8574_DrawChar(addr, str[i++]);
355        }
356    }
357
358    /**
359    * @desc    Shift cursor / display to left / right
360    *
361    * @param   char addr
362    * @param   char item {HD44780_CURSOR; HD44780_DISPLAY}
363    * @param   char direction {HD44780_RIGHT; HD44780_LEFT}
364    *
365    * @return  char
366    */
367    char HD44780_PCF8574_Shift (char addr, char item, char direction)
368    {
```

```
        // check if item is cursor or display or direction is left or right
        if ((item != HD44780_DISPLAY) && (item != HD44780_CURSOR)) {
            // error
            return PCF8574_ERROR;
        }
        // check if direction is left or right
        if ((direction != HD44780_RIGHT) && (direction != HD44780_LEFT)) {
            // error
            return PCF8574_ERROR;
        }
        // cursor shift
        if (item == HD44780_CURSOR) {
            // right shift
            if (direction == HD44780_RIGHT) {
                // shit cursor to right
                HD44780_PCF8574_SendInstruction(addr, HD44780_SHIFT | HD44780_CURSOR
                    | HD44780_RIGHT);
            } else {
                // shit cursor to left
                HD44780_PCF8574_SendInstruction(addr, HD44780_SHIFT | HD44780_CURSOR
                    | HD44780_LEFT);
            }
            // display shift
        } else {
            // right shift
            if (direction == HD44780_RIGHT) {
                // shit display to right
                HD44780_PCF8574_SendInstruction(addr, HD44780_SHIFT |
                    HD44780_DISPLAY | HD44780_RIGHT);
            } else {
                // shit display to left
                HD44780_PCF8574_SendInstruction(addr, HD44780_SHIFT |
                    HD44780_DISPLAY | HD44780_LEFT);
            }
        }
        // success
        return PCF8574_SUCCESS;
    }
```

Phụ lục B Sơ đồ chân nối

B.1 Sơ đồ chân nối mô hình điều khiển vị trí góc quay động cơ DC Encoder

ARDUINO NANO	DC ENCODER	L298N	LCD I2C	CHIẾT ÁP	NGUỒN 12V
2	Enc_A				
3	Enc_B				
5		ENB			
6		IN4			
7		IN3			
A0				OUTPUT	
A4			SDA		
A5			SCL		
Vin		5V	Vcc		
5V	Vcc			Vcc	
GND	GND		GND	GND	GND
		12V			12V
	Motor+	OUT3			
	Motor-	OUT4			

Hình B.1: Sơ đồ chân nối mô hình điều khiển vị trí góc quay động cơ DC Encoder