# CHƯƠNG 5 TÍNH TOÁN VÀ THIẾT KẾ GIẢI THUẬT ĐIỀU KHIỂN TỰ ĐỘNG

### **5.1 Thành phần cơ bản của hệ thống điều khiển**

Một hệ thống điều khiển về cơ bản có ba phần chính: Khối tín hiệu vào, Khối điều khiển, Cơ cấu chấp hành.

Cơ cấu chấp hành

Tác động

Khối điều khiển

Thu thập dữ liệu

Khối tín hiệu vào

(sơ đồ tổng quan một hệ thống điều khiển)

Một cách tổng quát, đối tượng của hệ thống điều khiển được điều khiển để có các tín hiệu ra tuân theo qui định của tín hiệu vào thông qua các phần tử trong hệ thống điều khiển. Tín hiệu ở đây là các tác nhân (hay kích thích) từ nguồn năng lượng ngoài cấp đến hệ thống điều khiển để tạo ra các đáp ứng mong muốn. Tín hiệu ra là các đáp ứng thực sự từ hệ thống điều khiển, có thể giống hay không giống đáp ứng mong muốn được xác định bởi các tín hiệu vào.

### 5.2 **Bộ điều khiển vòng hở động cơ DC dẫn động**

Đối với đề tài nghiên cứu, do đặc thù phải sử dụng nhiều loại động cơ với những chức năng khác nhau nên đòi hỏi cần phối hợp các bộ điều khiển khác nhau phù hợp với đặc tính tải của động cơ (được trình bày ở chương 2), có thể là “đơn giản” và cả “phức tạp”. Về cơ bản, chúng tôi sử dụng 2 bộ điều khiển chính, tạm thời được gọi là ***Bộ điều khiển vòng hở*** (open - loop) và ***Bộ điều khiển vòng kín*** (closed-loop). Cả hai bộ điều khiển này cũng thường được sử dụng trong kĩ thuật điều khiển, ứng dụng thực tiễn trong các hệ thống tự động của các ngành điện tử, viễn thông, cơ khí, ..

***Bộ điều khiển vòng hở*** được sử dụng cho các động cơ sau trong đề tài:

- Động cơ DC máy bơm (*Normal DC Motor)* dùng cho bộ phận bơm thuốc.

- Động cơ DC giảm tốc (*Greared DC Motor*) dùng cho tải chính robot.

***Bộ điều khiển vòng kín*** được sử dụng cho các động cơ sau trong đề tài: Động cơ DC Servo (*DC Servo Motor*) cho bộ phận chuyển hướng robot.

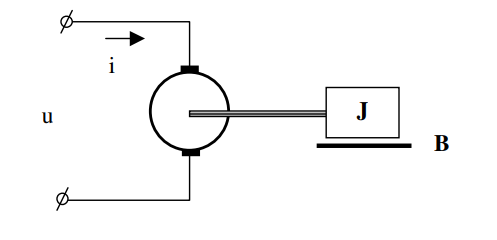
Bộ điều khiển vòng hở

Đặc thù điều khiển của các dạng động cơ *Stepper DC Motor, Normal DC Motor, Greared DC Motor* như đã trình bày ở trên về cơ bản là gần giống nhau. Hơn thế, động cơ DC giảm tốc là động cơ chính của hệ robot nên chúng tôi chỉ khảo sát và trình bày rõ về bộ điều khiển hở cho động cơ này.

Về cơ bản, ***Bộ điều khiển vòng hở*** có tác động điều khiển độc lập với tín hiệu ngõ ra. Ưu điểm hơn cả của bộ điều khiển này là có khả năng đạt độ chính xác bằng phương pháp thử (calibrate). Việc thử là thành lập hoặc tái thiết quan hệ giữa ngõ ra và ngõ vào theo mức chính xác mong muốn. Hệ thống điều khiển vòng hở có ưu điểm nối trổi là dễ thiết lập quy tắc điều khiển và gọn nhẹ. Tuy nhiên, độ chính xác điều khiển thấp và không đem đến khả năng đáp ứng tốt của hệ thống là nhược điểm cố hữu của bộ điều khiển dạng này.

|  |
| --- |
| Hình 5.2 Tổng quan về bộ điều khiển hở Tín hiệu tác động (u)  Biến được kiểm soát (c)  Tín hiệu tham chiếu (r) |
|  |

Để điều khiển động cơ *Greared DC Motor (tạm gọi là* ***động cơ DC****)*, chúng tôi sử dụng phương pháp **PWM** (*Điều chế độ rộng xung – Pulse Width Modulation*).

Trước hết, ta sẽ tìm hàm truyền đạt của động cơ, một mô hình của động cơ DC được khái quát với dạng như sau:

##### Hình 5.4 Hình sơ đồ hàm truyền động động cơ

Giả sử từ thông , J là moment quán tính quy về trục động cơ, B là hệ số ma sát của trục.

Phương trình quan hệ điện áp phần ứng:

Suy ra:

(5.1)

Phương trình quan hệ moment trên trục động cơ:

(5.2)

Thay (5.1) vào (5.2), ta được:

Vậy:

Với ; ;

Vậy hàm truyền đạt của động cơ là:

(5.3)

Ta sẽ thiết lập nguyên lí điều khiển động cơ với:

u: Tín hiệu vào là điện áp phần ứng động cơ

ω: Tín hiệu ra là góc quay của trục động cơ

(5.4)

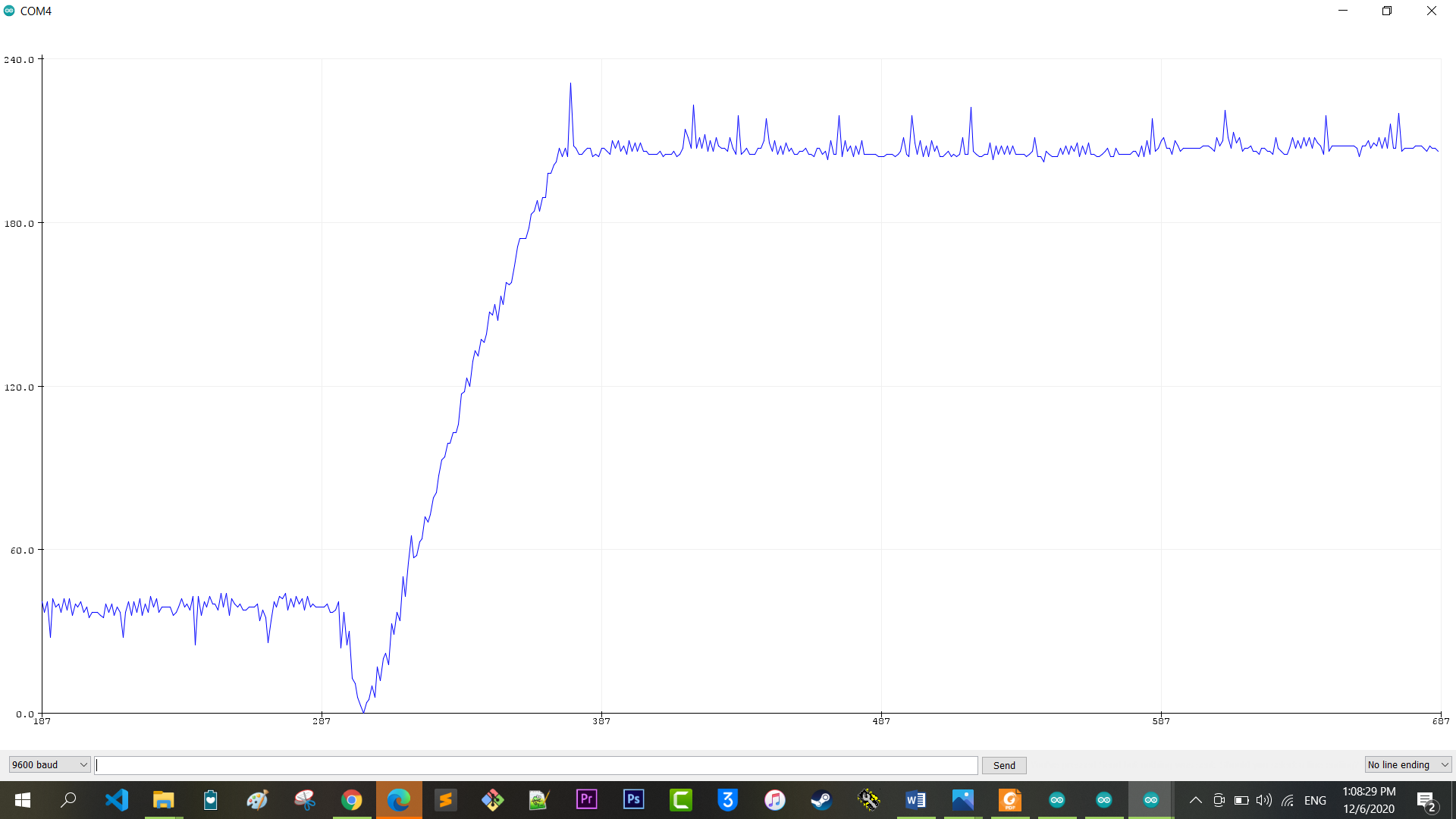
Trong đó:

(Time HIGH): Tần số động cơ được vận hành (s)

: Chu kì điều khiển (s)

e(t): Hàm truyền đạt

Quan hệ ngõ vào – ra này trên lí thuyết có thể xem là tuyến tính, tuy nhiên trong điều kiện thực tế, giá trị đáp ứng của động cơ DC sẽ là phi tuyến, đồ thị duty sẽ xuất hiện nhiễu gai.



Nhiễu phi tuyến

Vùng hoạt động

Timing

Duty

Vùng “điểm chết” (dead zone) của động cơ

##### Hình 5.5 Đồ thị đáp ứng dạng phi tuyến của động cơ DC trong thực tế

Theo như sơ đồ nguyên lí của hệ thống được thiết kế từ đầu, động cơ DC hoàn toàn được điều khiển từ xung PPM từ tay cầm. Do đó chúng ta sẽ xét sự ảnh hưởng của xung này đến duty của động cơ theo thời gian. Xung PPM từ tay cầm FlySky FS – TH9X được trả về giá trị timing (ms) ( thời gian tín hiệu ở mức cao hoặc xung thấp)

Khảo sát lí thuyết từ MCU bằng hàm ***pulseIn(channel, timeout)***, trong đó

- Channel: Kênh tín hiệu trả về từ bộ thu Reciever (kênh số 4)

- Timeout: Thời gian chờ

Hàm ***pulseIn()*** sẽ trả về data là một giá trị số (s) biến thiên tuyến tính từ thấp đến cao. Do đó chúng tôi dễ dàng mô hình phương thức điều khiển từ tay cầm về động cơ DC bằng bảng tham số của 3 trường hợp điều khiển như sau:

|  |  |  |  |
| --- | --- | --- | --- |
| Trạng thái tay cầm | Gạt xuống thấp | Trạng thái ban đầu | Gạt lên cao |
| Xung PPM(s) - HIGH | 1050 | 1490 - 1510 | 1910 |
| Duty Cycle (%) | 100 | 0 | 100 |
| Điện áp cung cấp cho DC Motor (V) | 24 | 0 | 24 |
| Trạng thái vận hành động cơ DC | Nhanh – Chạy nghịch | Chờ (Standby) - OFF | Nhanh – Chạy thuận |

###### Bảng 5.1 Bảng mô hình phương thức điều khiển từ tay cầm về động cơ DC bằng bảng tham số của 3 trường hợp điều khiển

Việc chuyển đổi từ tín hiệu xung PPM từ tay cầm FlySky FS – TH9X sang tín hiệu Duty như mong muốn được thực hiện theo tiến trình như sau:

Hàm đọc PPM ***pulseIn()***: Hàm số đọc tín hiệu PPM từ bộ thu Reciever.

Hàm chặn biên ***constrain()***: Hàm số ràng buộc giá trị nằm trong khoảng cho trước thường được sử dụng trong MCU Mega2560.

|  |
| --- |
| Hình 5.6 Tiến trình xử lí tín hiệu PPM thành tín hiệu PWM |

Cú pháp ***constrain (x, a, b)***;

* x: giá trị cần xét
* a: Chặn dưới (a là giá trị nhỏ nhất của khoảng)
* b: Chặn trên (b là giá trị lớn nhất của khoảng)

Hàm chuyển đổi tuyến tính ***map()***: Hàm số chuyển đổi thang đo từ giá trị này sang thang đo khác, giá trị trả về của hàm này là một số nguyên.

Cú pháp ***map(value, a1, a2, b1, b2);***

* value: Giá trị cần chuyển đổi
* a1, a2: Giới hạn trên dưới của thang đo cần chuyển đổi
* b1, b2: Giới hạn trên dưới của thang đó hiện tại

Hàm chiều đế độ rộng xung ***analogWrite()***: Hàm số xuất tín hiệu analog (phát xung PWM) thường được dùng trên MCU Mega2560

Cú pháp ***analogWrite([chân phát xung PWM], [giá trị xung PWM])****;*

Giá trị xung PWM nằm khoảng 0 đến 255, tương ứng mức Duty Cycle từ 0% đến 100%

Từ phương trình điều khiển hở cho *Greared DC Motor* (5.4) và quan hệ giữa tín hiệu PPM - Duty Cycle của động cơ DC Motor đã trình bày ở trên, ta có thể sử dụng để điều khiển lực thông qua Duty Cycle.

Tuy nhiên, khả năng điều khiển động cơ bằng phương pháp PWM thông thường đem đến độ chính xác **không cao**. Do đó, nó chỉ phù hợp cho các cơ cấu không đòi hỏi độ chính xác cao trong đề tài: Cơ cấu tải chính, Cơ cấu nâng hạ dàn phun, Bộ phận máy bơm. Vì bản thân độ chính xác của hệ thống cơ khí là không lí tưởng như điều kiện của phương trình và bản thân của phương trình điều khiển lực theo Duty Cycle chỉ chính xác tương đối do các sai số trong đo lường.

### 5.4 **Bộ điều khiển PID/PI/PD**

1. **Tổng quan**

Bộ điều khiển PID viết tắt của Proportional (tỉ lệ) – Integral (tích phân) – Derivative (đạo hàm), được xem là bộ điều khiển hồi tiếp được sử dụng nhiều nhất trong ứng dụng điều khiển tự động với độ chính xác, đáp ứng nhanh với độ ổn định cao. PID là từ đề cập đến 3 khâu hoạt động trên tín hiệu sai số để tạo ra một tín hiệu điều khiển.

Nếu u(t) là tín hiệu điều khiển gửi tới hệ thống, y(t) là đầu ra đo được và r(t) là đầu ra mong muốn, thì sai số theo dõi được hình thành là . Việc triển khai bộ điều khiển với sự đáp ứng nhanh thì đơn giản hóa là sử dụng hàm tuyến tính:

Viết lại với hệ thống điều khiển động cơ Servo:

Trong đó:

: Hàm ngõ ra (hàm )

: Độ lợi tỉ lệ

: Độ sai lệch ()

: Thời gian

là một hằng số (dương) nào đó mà chúng ta gọi là P (Proportional Gain), là sai số cần điều khiển. Mục tiêu ở đây là đưa (sai số) tiến về 0 càng nhanh càng tốt. Chúng ta nhận thấy, nếu lớn thì sự tác động nhanh của bộ điều khiển cũng lớn. Tuy nhiên, do quán tính nên việc điều khiển nhanh này dễ gây ra hiện tượng mất ổn định, điều này xảy ra do lực quán tính của động cơ và lực điều khiển tạo ra 2 dao động liên tiếp nhau, điều này gây ra dao động mất kiểm soát và ảnh hưởng cực kì xấu đến chất lượng điều khiển cũng như tuổi thọ của động cơ.

Như vậy, chúng ta sẽ sử dụng đạo hàm của sai số để làm tăng giá trị nhưng ngược chiều lực của động cơ. Đạo hàm này sẽ hãm lại vọt lố (overshoot) trong quá trình điều khiển. Đây chính là khâu D (Derivative Gain) trong bộ PID. Chúng ta thu được bộ điều khiển như sau:

Trong đó:

: Hàm ngõ ra

: Độ lợi vi phân

: Độ sai lệch ()

: Thời gian

Trong đó, chính là vận tốc thay đổi của sai số theo thời gian và là một hằng số không âm, gọi là khâu D. Thành phần khâu D này làm giảm overshoot của đối tượng điều khiển khi nó tiến về trạng thái cân bằng – vị trí ổn định. Lực F bao gồm 2 thành phần >= 0 (thành phần P) và <= 0 (thành phần D). Đôi khi khâu D có giá trị lớn hơn hẳn thành phần P và lực F của động cơ đổi chiều, hãm tốc lại, lúc này yếu tố cần điều khiển (vị trí, tốc độ,…) của đối tượng giảm mạnh tại vị trí cân bằng. Điều này dẫn đến vấn đề nảy sinh là nếu khâu D lớn hơn thành phần P hoặc bản thân thành phần P nhỏ thì sẽ xảy ra hiện tượng khi tiến gần đến vị trí cân bằng (thực tế là trong trường hợp này sẽ không bao giờ đến vị trí cân bằng) thì động cơ đã dừng, khâu D về 0 (vì không còn thay đổi nữa). Khi này lực của động cơ là F = , thêm vào đó, và lúc này đều nhỏ nên không thắng được ma sát tĩnh của động cơ (điểm chết - Dead Zone của động cơ). Sai số lúc này được gọi là sai số trạng thái tĩnh (Steady State Error). Để tránh xảy ra trường hợp này, người ta lại thêm một thành phần có khả năng cộng dồn sai số.

Khi steady state error xảy ra, 2 thành phần P và D mất tác dụng điều khiển, thành phần điều khiển mới này sẽ cộng dồn sai số theo thời gian, do đó lực F sẽ tăng lên theo thời gian. Đến một lúc nào đó, lực F đủ lớn để thắng ma sát tĩnh, đẩy đối tượng đến vị trí mong muốn. Thành phần cộng dồn này chính là I (Intergral Gain) trong bộ điều khiển PID. Đến đây, chúng ta đã hoàn thiện bộ điều khiển PID với dạng tổng quát như sau:

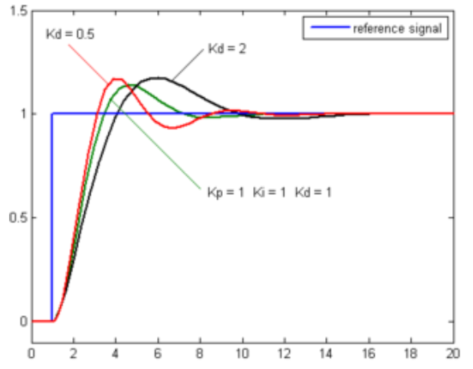
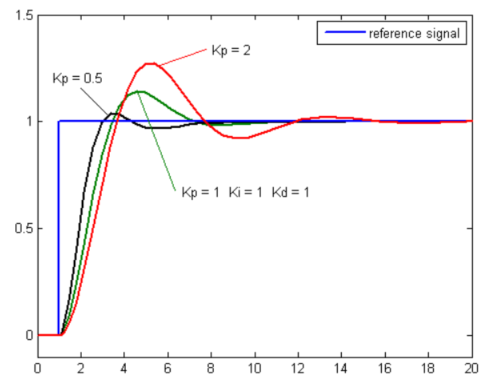
Trong đó:

: Hàm ngõ ra

: Độ lợi tích phân

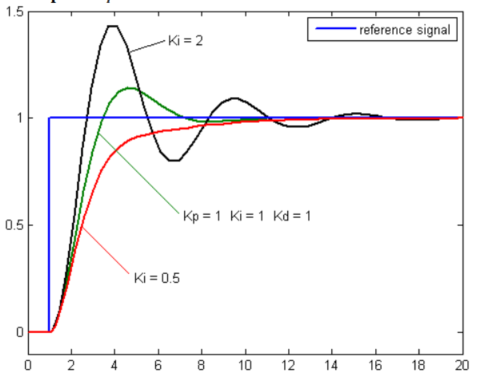
: Độ sai lệch ()

: Thời gian

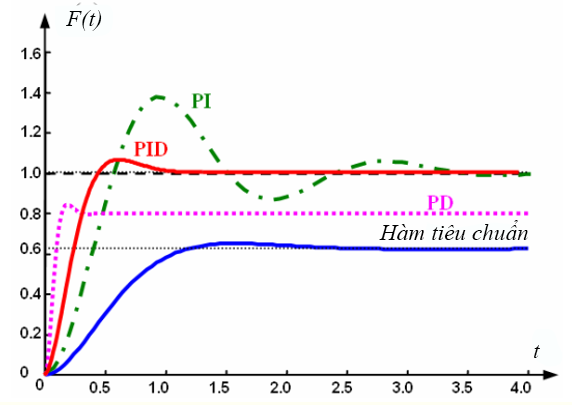
Tuy nhiên, tùy vào mục đích sử dụng cụ thể, có thể tùy chỉnh sử dụng bộ PID, PD, PI bằng cách cho các hệ số không dùng về 0.

##### Hình 5.8 Đồ thị hàm số theo thời gian, ba giá trị Kd (Ki, Kp là hằng số)

##### Hình 5.9 Đồ thị hàm số theo thời gian, ba giá trị Kp (Ki, Kd là hằng số)



##### Hình 5.10 Đồ thị hàm số theo thời gian, ba giá trị Ki (Kd, Kp là hằng số)

****

##### Hình 5.11 Đồ thị mô phỏng đáp ứng của bộ PID, PD, PI

Đối với đề tài nghiên cứu, chúng tôi sử dụng bộ điều khiển PI vì những lí do sau:

* Giảm tính phức tạp của bộ điều khiển
* Robot di chuyển chậm với tốc độ dao động 0.72 ~ 1.08km/h do đó không cần yêu cầu xử lí bẻ lái động cơ nhanh. Yêu cầu về thời gian xác lập không cao.
* Robot cần bẻ lái với góc rộng từ 30 độ đên -30 độ.

Bộ PI tuy có thời gian xác lập chậm nhưng đem đến khả năng đáp ứng chính xác, do đó sẽ lựa chọn phù hợp cho đề tài để giảm thiểu độ phức tạp và gánh nặng cho MCU. Đến đây, công việc chính của việc thiết kế là chuyển lí thuyết của hàm PID vào mã nguồn của MCU và lựa chọn các hệ số , , sao cho hợp lí nhất.

Việc lựa chọn , , thực ra đòi hỏi tuân theo các tiêu chuẩn ổn định (stability criteria) hoặc các phương pháp chọn hệ số (Ziegler – Nichols, bằng tay, bằng phần mềm,…). Tuy nhiên, việc áp dụng các tiêu chuẩn này thường phức tạp và mất thời gian, thay vào đó, chúng tôi sử dụng phương án chọn hệ số bằng tay (chọn theo kinh nghiệm).

Quả khảo sát đã thực nghiệm được xem xét ở chương số 6 và kinh nghiệm chọn hệ số , , đã tổng hợp được, chúng tôi xin được tóm tắt cơ bản cách chọn hệ số như sau:

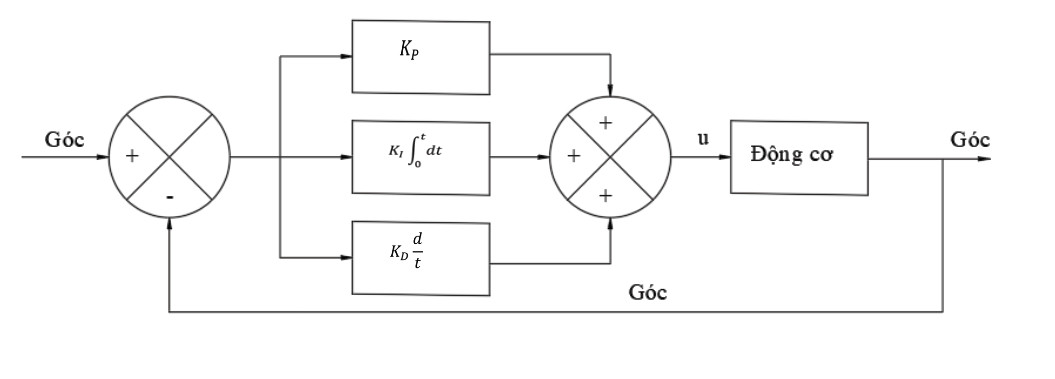
* Đặt hệ số  **, = 0.**
* Chọn hệ số trước, thử hệ thống với đối tượng thật (động cơ DC Servo) và đọc các giá trị hồi tiếp (Encoder, biến trở..). Chọn giá trị nhỏ nhất cho . Điều chỉnh giảm sao cho thời gian đáp ứng đủ nhanh và overshoot nhỏ trong mức cho phép. Việc chọn ngay từ ban đầu được xem là quan trọng nhất, **kinh nghiệm là tăng đến khi overshoot thì giảm đi 1/2, tiến hành tăng .**
* **Nếu sử dụng bộ PD thì không chọn và ngược lại, sử dụng bộ PI thì không chọn .**

-Tăng từ bé đến lớn và ngăn không cho overshoot xuất hiện.

-Tăng để loại bỏ hẳn overshoot, tăng từ từ kết hợp thử sai để chọn giá trị thích hợp. Khả năng hiện tượng Steady State Error sẽ xảy ra, lúc này dừng chọn . Theo kinh nghiệm, giá trị hệ số .

Như đã trình bày, chúng ta cũng có thể chọn các hệ số , , bằng các tiêu chuẩn ổn định hoặc bằng phương pháp hiệu chỉnh Ziegler – Nichols.

1. **Ứng dụng bộ điều khiển PI vào DC Servo Motor trong Robot**

Bộ PI sẽ được chọn sử dụng điều khiển trong đề tài, nhưng trước hết cần có phương pháp chuyển đổi (porting) lí thuyết điều khiển PI về dạng số rời rạc để thuận tiện cho việc lập trình trên MCU. Ở đây, để tổng quát hóa, **chúng tôi vẫn sẽ trình bày cách thức chuyển đổi số cả bộ PID, mặc định khi chọn hệ số chúng tôi sẽ cho bằng 0.**

Encoder

##### Hình 5.12 Mô hình điều khiển động cơ Servo dẫn hướng

Dạng mô hình điều khiển PID liên tục có được:

(1)

Trong đó:

* (hệ số tỉ lệ)
* (hệ số tích phân),
* (hệ số vi phân),
* : Thời gian tích phân

Chuyển đổi thành bộ PID rời rạc:

- Rời rạc hóa khâu tích phân:

- Rời rạc hóa khâu vi phân:

Thay vào bộ điều khiển PID liên tục (1), ta được:

Trong đó:

Đặt **;**

Khi này, hàm ngõ ra điều khiển sẽ có dạng:

Công thức hàm điều khiển PID số:

) (2)

Trong đó:

: Sai số hiện tại

: Sai số trước lần thứ

: Sai số trước lần thứ

Tín hiệu điều khiển ba khâu của bộ điều khiển PID là tổng của ba khâu rời rạc P, I, D. Mã giả rời rạc của bộ điều khiển được minh họa như sau:

*previous\_error = 0*

*integral = 0*

*loop:*

*error = setpoint - measured\_value*

*integral = integral + error \* dt*

*derivative = (error - previous\_error) / dt*

*output = Kp \* error + Ki \* integral + Kd \* derivative*

*previous\_error = error*

*wait(dt)*

*goto loop*

Trên thực tế sử dụng, đôi khi người ta sử dụng bộ PID số mà chỉ lấy mẫu sai số hiện tại , bỏ qua các sai số quá khứ, điều này là không sai nhưng chưa phát huy hết tính chính xác và khả năng lấy mẫu của bộ điều khiển. Bộ PID số theo công thức (2) trên có thành phần I càng ngày càng nhỏ và phù hợp hơn. Do đó với đặc thù của đề tài, chúng tôi điều khiển động cơ DC Servo với setpoint thay đổi lớn và liên tục (trong trường hợp này là điều khiển vị trí encoder với độ phân giải lên đến 1000 xung/vòng), nếu chỉ dùng phương pháp lấy mẫu một lần thì sai số tích lũy cộng dồn sẽ rất lớn, lớn đến một mức nào đó mà khi đã xác lập rồi thành phần I vẫn dương dẫn đến vọt lố, sau một thời gian vọt lố, thành phần I mới giảm dần về trạng thái cân bằng. Sử dụng bộ PID số theo công thức (2) sẽ hạn chế được điểm yếu này.

**Thời gian lấy mẫu**: Bộ PID ngoài việc phụ thuộc vào các hệ số , , thì còn phụ thuộc vào chu kì lấy mẫu T (**chu kì T càng nhỏ là càng tốt**). Trong MCU Mega2560, chúng tôi sử dụng cơ chế ngắt (Interrupt) với hàm ***attachInterrupt()*** và thư viện định thời ***timer1.h****(open source)* để định thời gian lấy mẫu.

Cú pháp hàm ngắt lấy mẫu Encoder: ***timer1.attachInterrupt( interrupt, ISR, mode )***

Trong đó:

* + ***Interrupt***: Số thứ tự chân ngắt, ở đây sử dụng 1 chân ngắt số 2 cho kênh A của Encoder, chân digital thông thường số 4 sử dụng cho kênh B của Encoder.
  + ***ISR(Interrupt Service Routine)***: Trình phục vụ ngắt, tên hàm gọi đến khi sự kiện ngắt xảy ra sẽ được thực thi (ở đây là chuỗi hàm target xử lí tính toán PI).
  + ***Mode***: Kiểu kích hoạt ngắt

+ ***LOW***: Kích hoạt liên tục khi trạng thái chân digital mức thấp.

+ ***HIGH***: Kích hoạt liên tục khi trạng thái chân digital 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.

Cú pháp đặt thời gian Timer: ***Timer1.initialize(time)***¸trong đó, time là thời gian lấy mẫu (mili giây).

Ở đây chúng tôi sử dụng cấu hình: ***attachInterrupt( channelA, PIDCompute(), HIGH )***. Trong đó, ***channelA*** được cấu hình là chân digital số 2 để đọc xung channel A từ đĩa encoder quang của động cơ DC Servo, ***PIDCompute()*** là chuỗi hàm xử lí bộ điều khiển sau ngắt, kiểu kích hoạt ngắt là ***HIGH***.

***Bật cờ, cài định thời:  
Timer1.initialize()***

***Cho phép ngắt: interrupts()***

Thanh ghi MCU

***Hủy ngắt:  
noInterrupts()***

*Chu trình xử lí   
hàm ISR*

*Hàm ngắt đọc Encoder:  
Timer1.attachInterrupt()*

`

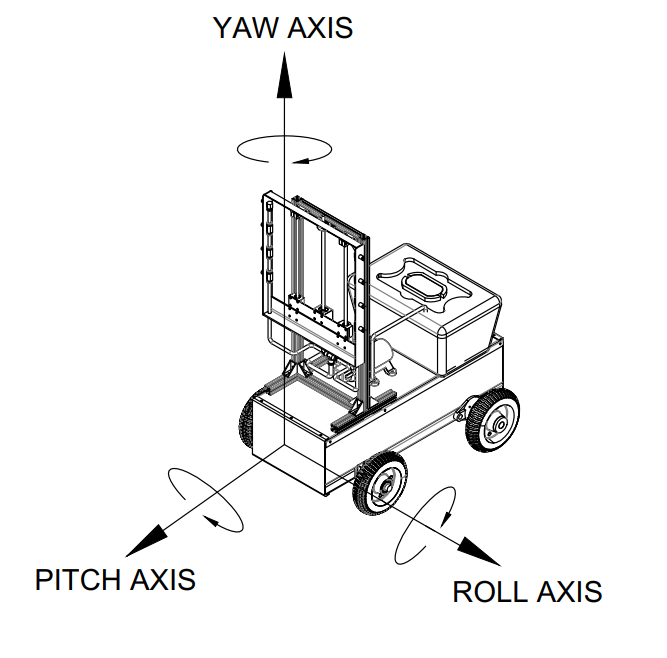
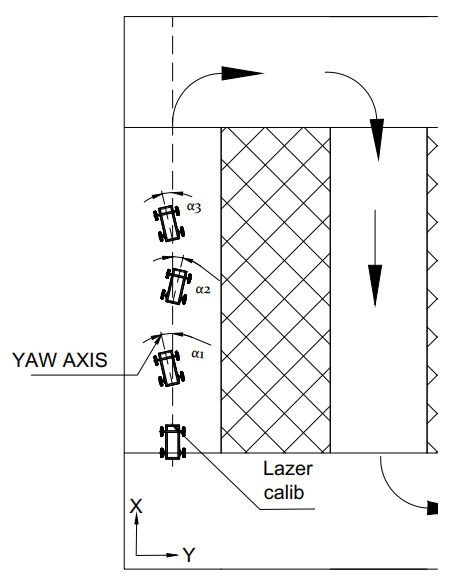
Chu kì

##### Hình 5.13 Sơ đồ hàm ngắt

Các hàm ngắt và định thời này sẽ được đặt kết hợp trong phần setup của chương trình (bên ngoài chương trình chính) và bên trong chương trình chính. Hình bên trên mô tả quá trình sự kiện ngắt diễn ra trên chương trình với một chu kì điều khiển T.

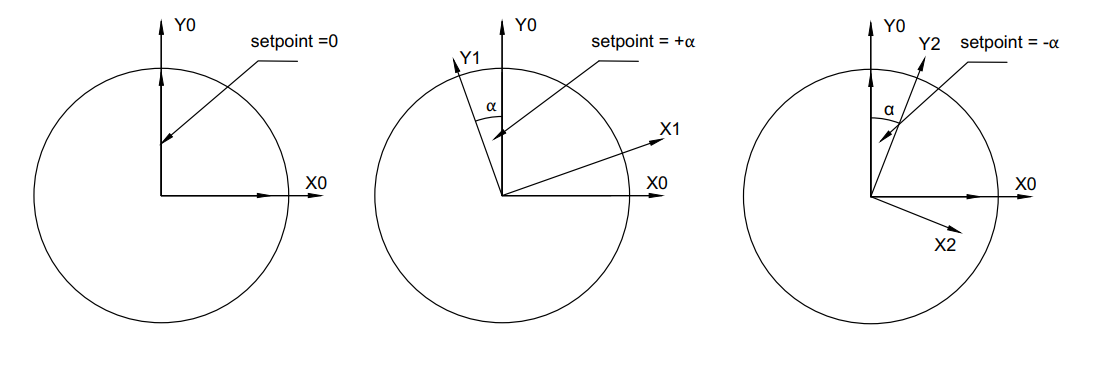
**5.1 Điều khiển động cơ dẫn hướng với MPU6050**

Như đã trình bày ở mục **4.2.1.2 - Phương án 2: Sử dụng cảm biến gia tốc góc nghiêng GY-521 6DOF MPU6050**, chúng tôi đã đề cập đến việc sử dụng bộ nhớ đệm DMP (Digital Motion Processor). Bộ nhớ này kết hợp dữ liệu từ con quay hồi chuyển và gia tốc kế để tính toán định hướng di chuyển của robot.

((Mô hình hệ tọa độ Roll – Pitch – Yaw gắn trên robot)

(mô hình vận hành robot với góc lệch trục Yaw)

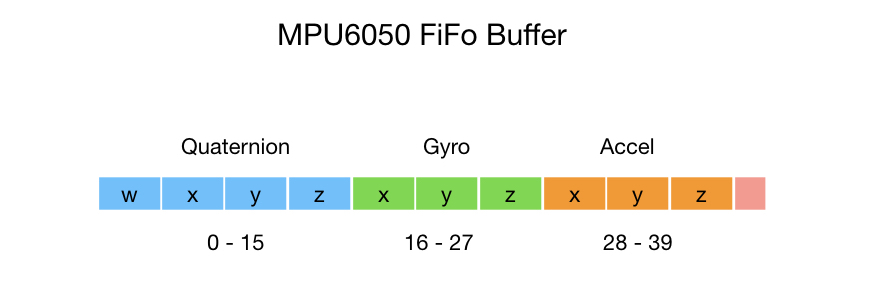
Theo đó, với việc phát hiện góc lệch trục Yaw của MPU trong quá trình thực hiện, ta thực hiện điều khiển vị trị động cơ dẫn hướng theo setpoint cài đặt cũng chính bằng đối trọng góc lệch của trục Yaw. Hình dưới đây mô tả mối quan hệ điều khiển giữa hệ tọa độ gắn trên MPU (robot) và góc điều khiển dẫn hướng của robot.



(Phương án đặt setpoint theo góc lệch Yaw trên robot)

Hệ tọa độ gốc (X0,Y0) là hệ trục được MPU xác định khi calib, hệ tọa độ (X1, Y1) và hệ tọa độ (X2, Y2) là 2 trường hợp giả định khi robot lệch khỏi hệ tọa độ gốc các góc lần lượt là α1 và α2. Như hình vẽ mô tả, khi MPU đo đạc góc lệch là α thì góc setpoint đặt cho động cơ là –α, ngược lại, góc setpoint sẽ được đặt là α.

Về cơ chế lấy mẫu tín hiệu, MPU6065 có bộ nhớ DMP được tích hợp các phần bù (offsets) và cơ chế lấy mẫu riêng. Đây cũng chính là dạng của bộ dữ liệu FIFO bên trong

DMP.

(Bộ đệm FIFO bên trong MPU 6050)

Cơ chế trích xuất tín hiệu DMP được minh họa bằng đoạn mã như sau:

*//Kiểm tra ngắt sẵn sàng trên DMP*

**if** (mpuIntStatus **&** 0x02) {

*//Đợi dữ liệu có chiều dài hợp lệ*

**while** (fifoCount **<** packetSize) fifoCount **=** mpu.getFIFOCount();

*// Đọc gói tin từ FIFO*

mpu.getFIFOBytes(fifoBuffer, packetSize);

fifoCount **-=** packetSize;

*//Lấy giá trị từ mảng dữ liệu*

mpu.dmpGetQuaternion(**&**q, fifoBuffer);

mpu.dmpGetGravity(**&**gravity, **&**q);

mpu.dmpGetYawPitchRoll(ypr, **&**q, **&**gravity);

**float** Roll **=** ypr[0] **\*** 180**/**M\_PI;

**float** Pitch **=** ypr[1] **\*** 180**/**M\_PI;

**float** Yaw **=** ypr[2] **\*** 180**/**M\_PI;

}

DMP cập nhật dữ liệu cảm biến vào các bộ nhớ đệm và có cơ chế trích xuất dữ liệu bên trong thư viện giao tiếp. Do đó, khi cần polling gọi dữ liệu, chúng ta chỉ cần truy xuất đến một ô nhớ trong mảng dữ liệu của bộ nhớ DMP.