Trường Đại học Bách Khoa TP.HCM
Khoa Khoa học và Kỹ thuật Máy tính

PID CONTROLLER

GVHD: Phạm Hoàng Anh

SVTH: Phan Gia Anh

Mục lục

Danh mục hình ảnh

I. PID CONTROLLER LÀ GÌ?

PID Controller hay **Bộ Điều Khiển Vi Tích Phân Tỉ Lệ**….- để cho đơn giản thì ta hãy cứ gọi là PID Controller – là một bộ điều khiển vòng kín phản hồi với mục đích tạo ra những tín hiệu ở một mức độ mà mình mong muốn. Nghe có vẻ phức tạp nhưng về ý tưởng nó cũng chỉ đơn giản là làm thế nào để có thể có những tín hiệu điều khiển "hoàn hảo" nhất có thể.

Để hiểu rõ hơn về PID Controller, ta hãy gọi nó một cách đúng đắn hơn: P-I-D Controller bởi vì thực chất PID Controller là một tổ hợp của ba bộ điều khiển khác nhau: ***Proportional, Integral*** *và* ***Derivative***

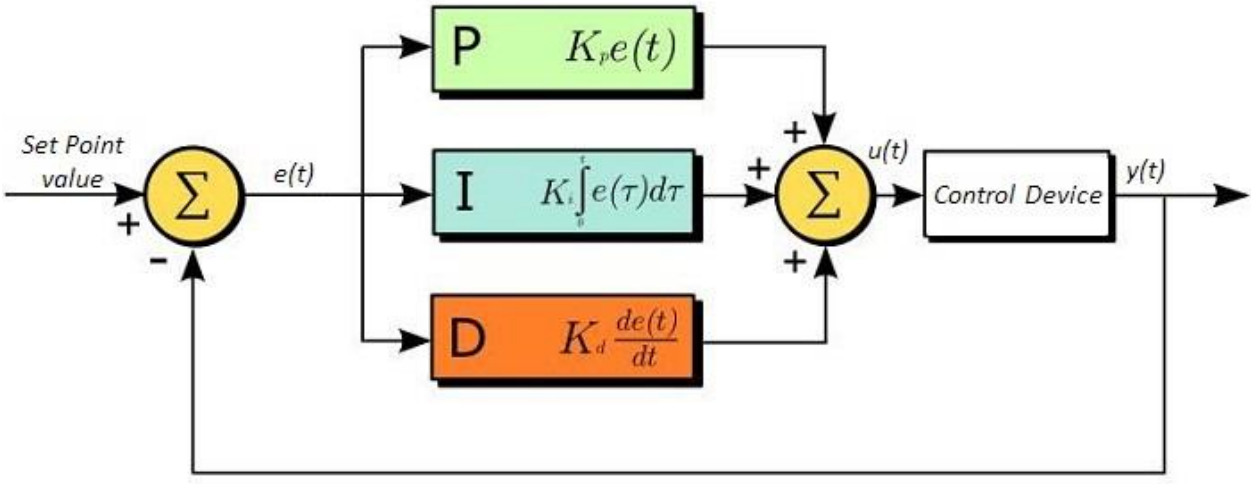

Figure 1. P-I-D Controller

Vấn đề đặt ra: Giả sử bạn đang tham gia Cuộc đua số, bạn muốn chiếc xe dò đường của mình phải thật xịn xò và chạy mượt mà nhất có thể. Nhưng liệu điều ấy có xảy ra ở thực tế hay không? Đương nhiên việc di chuyển của xe sẽ tùy thuộc vào những điều kiện như vị trí hiện tại của xe, tốc độ chạy,… Nhưng với những dữ liệu đó, làm thế nào bạn có thể đảm bảo tín hiệu điều khiển bạn đưa ra thật sự tốt?

Chúng ta sẽ đi vào từng khái niệm của các bộ điều khiển này và sau đó tìm hiểu tại sao chúng lại cần đi chung với nhau để tạo ra một tín hiệu điều khiển tốt.

II. P-CONTROLLER

Proportional Controller hay Bộ điều khiển tỉ lệ sẽ luôn tạo ra một tín hiệu output tỉ lệ với giá trị lỗi tức thời *e(t)* tại thời điểm đó.

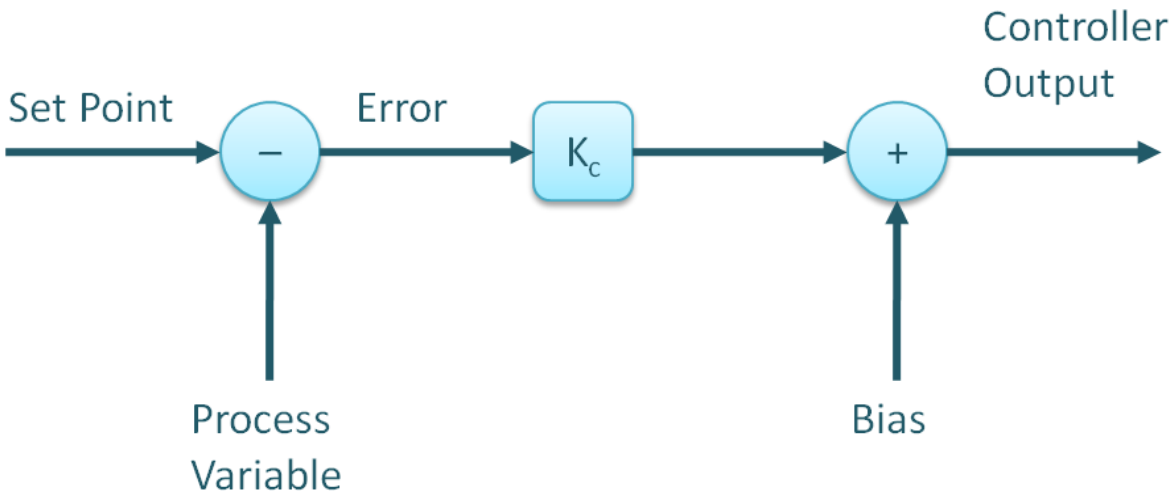

Figure 2. P-Controller

Để hiểu rõ hơn, ta hãy lấy một ví dụ:

Ví dụ: Ta lại đến với Cuộc đua số rồi, và bây giờ bạn đang có một chiếc xe để thử nghiệm một vài tuần trước khi cuộc thi chính thức diễn ra. Điều đầu tiên bạn muốn rằng chiếc xe ít nhất phải chạy được đến nơi nó muốn đã. Hãy giả sử chiếc xe đang cách đích đến 100m và giờ bạn muốn áp dụng P-Controller để điều khiển xe đua, hãy xem nó sẽ làm thế nào:

Những thông số được thiết lập sẽ là:

- **Set Point**: 100m, là mục tiêu bạn muốn đạt được.
- **Process Variable:** 0m, vị trí hiện tại của xe đua.
- $\boldsymbol{K_C}$**:** Hằng số tỉ lệ, giả sử ta muốn vận tốc sẽ phụ thuộc vào khoảng cách đến đích theo tương quan: Càng gần đích đến thì sẽ càng đi chậm lại. Khi đó giả sử đặt $\boldsymbol{K_C = 0.1.}$

Tạm thời là vậy, ta sẽ xem biểu hiện của xe đua nhé:

- Đầu tiên xe của chúng ta sẽ bắt đầu chạy, lúc này khoảng cách tới đích đến là 100 – 0 = 100(m), và đây chính là độ lệch(**Error**) mà ta mong muốn về 0.
- **Error** này sẽ kết hợp với hằng số $\boldsymbol{K_C = 0.1}$ để tạo một tín hiệu điều khiển **Controller Output** $\boldsymbol{y(0) = K_C * E(0) = 10}$
- Tín hiệu này giả sử được làm tín hiệu vận tốc cho xe đua, khi đó xe chúng ta sẽ bắt đầu chạy với vận tốc 10m/s.

- Tiếp tục như thế khi xe đi được x(m) thì vận tốc của xe sẽ là (100 - x)*0.1 (m/s) và đến khi xe đến được vị trí 100m, vận tốc sẽ là 0, và xe sẽ dừng lại!!!

Nghe có vẻ hoàn toàn ổn, đúng không nào?
Ta có thể nhận thấy rằng P-Controller sẽ phụ thuộc vào error, và điều đó cũng có 2 mặt:

- Tốt: Output sẽ càng lúc càng tiệm cận với mong muốn và error sẽ càng lúc càng tiến về 0, tính ổn định cao
- Xấu: Việc dựa vào error sẽ dẫn đến 1 hậu quả: error sẽ vẫn còn mãi do output chỉ có thể tồn tại khi error khác 0, nếu error bằng 0 output sẽ bằng 0.

Ta hãy xem qua 1 ví dụ để làm rõ vấn đề này:
Ví dụ: Chúc mừng bạn đã vượt qua vòng 1 của Cuộc Đua Số và giờ là vòng 2: Cuộc Bay Số. Thứ bạn cần điều khiển bây giờ là một chiếc drone, bạn mong muốn nó bay đến độ cao 100m và dừng ở đó. Nếu dùng P-Controller, điều gì sẽ xảy ra?

Figure 3. P-Controller cho drone?

Đúng vậy đấy, khi drone đã đạt tới độ cao 100m, error sẽ là 0 và đương nhiên output sẽ là 0, do đó ngay khi vừa đạt 100m thì drone sẽ **rơi xuống.**
Sau đó do error giờ đã khác 0 khiến tín hiệu output sẽ khác 0 và drone lại tiếp tục **bay lên,** drone giờ đây như 1 cái lò xo giữa không trung vì nó cần biết error tại thời điểm đó để quyết đinh tín hiệu điều khiển là gì.

Đây được gọi là **steady state error**, xảy ra khi hệ thống hoạt động ở trạng thái ổn định và tín hiệu lỗi luôn tồn tại trong suốt quá trình. Đương nhiên đối với vài hệ thống thì không bận tâm về điều này và

có thể hoàn toàn chỉ dùng P-Controller, tuy nhiên nếu chúng ta muốn tín hiệu điều khiển phải đạt được một mức độ chính xác nhất định thì thông số cuối cùng mà ta chưa nhắc ở trên sẽ tham gia quá trình này: **Bias.**

Bản thân P-Controller nếu dùng độc lập thì phải cần reset tự động hoặc chế độ thúc đẩy tín hiệu điều khiển để đảm bảo loại trừ được steady state error, tuy nhiên nếu chúng ta xem xét đến bộ điều khiển thứ 2 sau đây ta sẽ thấy những lợi ích mà nó mang lại.

Trước khi qua phần mới, hãy nhắc lại công thức P-Controller từ những gì đã xây dựng được:

$$P(t) = K_C e(t)$$

III. I-CONTROLLER VÀ PI-CONTROLLER

Như đã nói, I-Controller được dùng để khắc phục những gì P-Controller đã để lại: steady state error.

Một cách nói khác, ta có thể nói P-I-D tương đương với **Hiện tại – Quá khứ - Tương lai.** P-Controller tập trung vào error ở thời điểm hiện tại để sinh output và I-Controller sẽ tập trung vào dữ liệu error ở quá khứ để sinh output. Nghĩa là như thế nào?

I-Controller hay Integral Controller sẽ tích hợp tất cả những giá trị của error từ lúc bắt đầu đến thời điểm hiện tại để quyết định tín hiệu điều khiển

$$I(t) = K_I \int_0^t e(\tau) d\tau$$

Khi đó nếu có steady state error xảy ra thì error vẫn tồn tại và I-Controller vẫn sinh tín hiệu điều khiển dù tín hiệu P-Controller không còn ảnh hưởng hệ thống được nữa. Nếu error âm, tức output đã đi qua mức mong muốn, e(t) sẽ âm và khiến I(t) giảm, khiến output quay ngược trở về vị trí mong muốn.

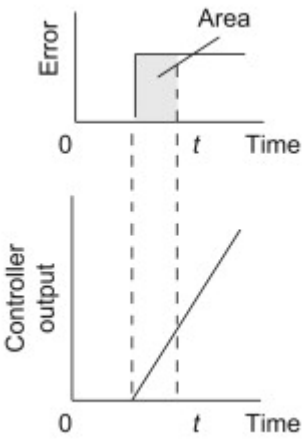

Figure 4. Output của I-Controller phụ thuộc error

PI-Controller sẽ là sự kết hợp của P-Controller và I-Controller, chủ yếu sẽ là P-Controller với bias là I-Controller để triệt tiêu steady state error

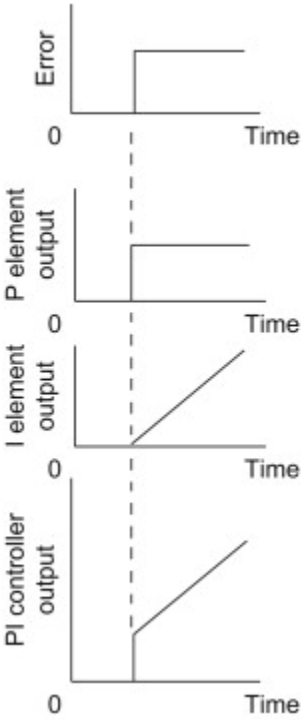

Figure 5. PI-Controller

Ta đã nhận thấy vấn đề của P-Controller ở **steady state error**, và đến với I-Controller ta sẽ nhận thấy một vấn đề mới: **integrator windup.**

Integrator windup sẽ xảy ra ở thực tế, khi thiết bị của chúng ta không đáp ứng được tín hiệu điều khiển của I-Controller dẫn đến bão hòa(**Saturation**)**.** Khi đó dù output I(t) vẫn tiếp tục tăng nhưng thiết bị không thể nào làm được gì hơn nữa. Và điều hay ho xảy ra là: một khi mà thiết bị đó có thể đáp ứng được rồi thì do output I(t) giờ nó đã quá lớn nên sẽ mất một khoảng thời gian, có thể rất dài, để tín hiệu có thể giảm về mức mong muốn.

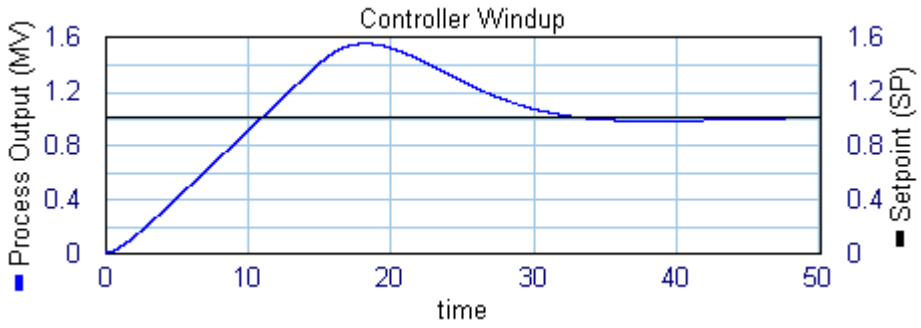

Figure 6. Integrator Windup

Việc kìm hãm độ lớn của các tín hiệu điều khiển là điều cần thiết và Bộ điều khiển tiếp theo sẽ có nhiệm vụ đó.

IV. D-CONTROLLER VÀ PID CONTROLLER

D-Controller hay Derivative Controller sẽ dùng dữ liệu error trong suốt quá trình mà dự đoạn được trong tương lai error sẽ đi theo chiều hướng nào dựa vào tốc độ thay đổi của e(t).

$$D(t) = K_D \frac{de(t)}{dt}$$

Theo công thức trên, ta sẽ thấy D(t) sẽ tùy thuộc vào độ biến đổi của e(t) mà sinh ra output cho tín hiệu điều khiển.

- Nếu e(t) giảm: D(t) sẽ là một tín hiệu có dấu âm và kìm hãm sự tăng của tín hiệu tổng PID.
- Nếu e(t) tăng: D(t) sẽ là một tín hiệu dương và thúc đẩy tín hiệu điều khiển để khiến cho e(t) giảm về 0.
- Tốc độ tăng của e(t) sẽ ảnh hưởng trực tiếp đến độ lớn của D(t).

Khi ta kết hợp 3 bộ điều khiển lại thì ta sẽ có một bộ điều khiển PID hoàn chỉnh:

- P-Controller: giúp tín hiệu điều khiển luôn tiến gần đến kết quả mong muốn đạt được, hệ thống ổn định nhưng sẽ luôn có steady state error do output luôn phụ thuộc vào lỗi.
- I-Controller: triệt tiêu trạng thái steady state error nhưng đôi lúc sẽ gây ra hiện tượng overshooting và integrator windup.
- D-Controller: giúp kìm hãm những tín hiệu khác, tạo output thích hợp nhất tùy theo mức độ và trạng thái lỗi.

Bộ điều khiển PID trên thực tế sẽ có kết hợp những mạch như anti windup(sử dụng bộ kiểm tra bão hòa) và bộ chống nhiễu để đảm bảo kết quả tốt nhất.

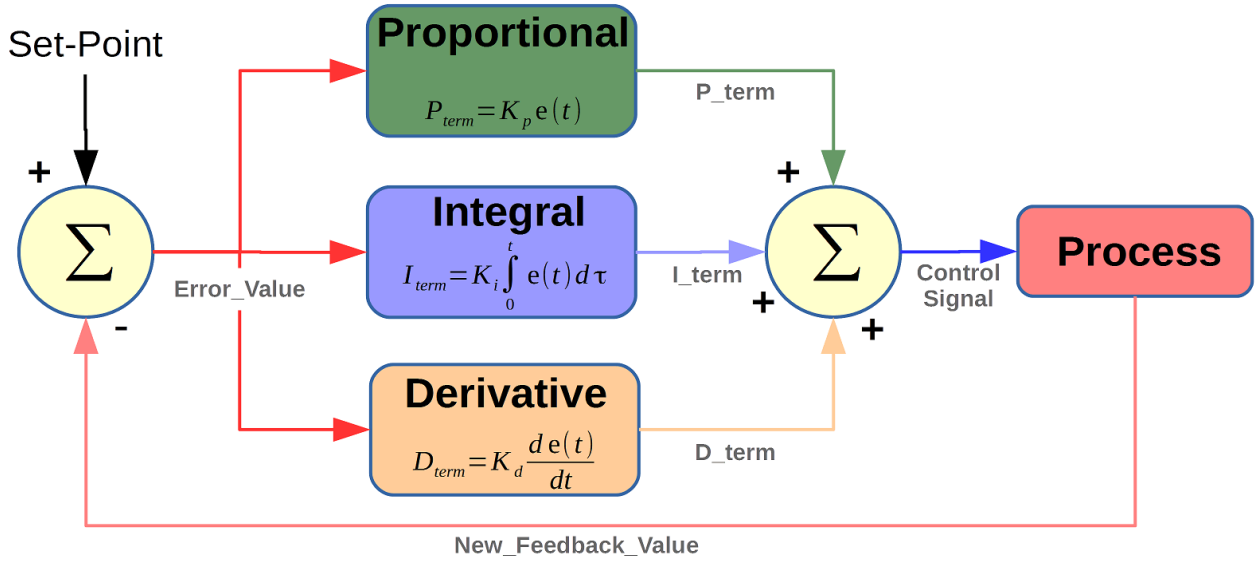

Figure 7. PID Controller

V. DSP TRONG VIỆC HIỆN THỰC PID CONTROLLER

Việc xử lý tín hiệu analog là một điều khó nhằn và số thực cũng vậy, do đó đưa về những tín hiệu rời rạc để xử lí là một ý tưởng không tệ. Hãy xem xét cách để hiện thực PID Controller nhé.

Đầu tiên, ta hãy quay trở về công thức gốc của PID:

$$u(t) = K_P e(t) + K_I \int_0^{\tau} e(\tau) d\tau + K_D \frac{de(t)}{dt}$$

Để sử dụng công thức này dưới dạng rời rạc cần lưu ý vài đặc điểm sau đây:

- Việc ghi nhớ dữ liệu từ đầu đến cuối sẽ tốn rất nhiều dữ liệu nên với việc tiếp cận thực tế theo hướng lấy mẫu theo chu kỳ, ta chỉ nên giữ lại một số lần gần nhất và bỏ đi những lần quá xa.
- Độ chính xác khi lấy mẫu như vậy sẽ không bằng được khi hiện thực dùng tín hiệu analog và dữ kiện được lưu từ đầu, tuy nhiên nếu tần số lấy mẫu đủ cao thì sai số sẽ không đáng kể.

1. Lấy mẫu(Sampling):

Lựa chọn tần số lấy mẫu *f* tùy theo mong muốn độ chính xác của kết quả.

Chu kỳ lấy mẫu:

$$\boldsymbol{T = \frac{1}{f}}$$

Tín hiệu error:

$$\boldsymbol{e(k) = e\left(\frac{t}{T}\right), k \in \mathbb{N}}$$

Tín hiệu P-Controller: Lấy mẫu ở thời điểm hiện tại

$$\boldsymbol{P(k) = K_P e(k)}$$

Tín hiệu I-Controller: Chính là diện tích của e(t) tạo nên, là tổng của nhiều hình thang vuông tạo bởi e(k) và e(k-1), cạnh hình thang là hằng số T, do lấy mẫu đều

$$\boldsymbol{I(k) = K_I \sum_{i=1}^{k} \frac{e(i) + e(i-1)}{2} * T}$$

Tín hiệu D-Controller: Tạo nên bởi độ biến thiên của e(t)

$$\boldsymbol{D(k) = K_D \frac{e(k) - e(k-1)}{T}}$$

2. Tạo hàm đệ quy

Với ý tưởng giữ những trạng thái gần nhất, ta nên tạo hàm đệ quy, với những biểu thức sampling ở trên, ta có công thức rời rạc cho PID Controller:

$$u(k) = K_P e(k) + K_I \sum_{i=1}^{k} \frac{e(i) + e(i-1)}{2} * T + K_D \frac{e(k) - e(k-1)}{T}$$

$$\Rightarrow u(k-1) = K_P e(k-1) + K_I \sum_{i=1}^{k-1} \frac{e(i) + e(i-1)}{2} * T + K_D \frac{e(k-1) - e(k-2)}{T}$$

$$\Rightarrow u(k) - u(k-1) =$$

$$K_P e(k) - K_P e(k-1) + K_I \frac{e(k) + e(k-1)}{2} * T + K_D \frac{e(k) - 2e(k-1) + e(k-2)}{T}$$

Và cuối cùng ta có dạng đệ quy của PID Controller rời rạc:

$$u(k) = u(k-1) + K_1 e(k) + K_2 e(k-1) + K_3 e(k-2)$$

Trong đó:

$$K_1 = K_P + \frac{TK_I}{2} + \frac{K_D}{T}$$

$$K_2 = -K_P + \frac{TK_I}{2} - 2\frac{K_D}{T}$$

$$K_3 = \frac{K_D}{T}$$

Sau khi có được phương trình rời rạc, xây dựng mô hình hồi quy là khá dễ dàng, tuy nhiên bước khó khăn tiếp theo là việc xử lí các giá trị số thực của e(t), để rời rạc hóa giá trị của e(t) thì sẽ tùy vào dung lượng bộ nhớ bạn cung cấp cũng như mức độ chính xác sau dấu phẩy để đảm bảo độ nhạy của hệ thống điều khiển.

VI. MÔ PHỎNG PID CONTROLLER

Để dễ hiểu hơn về PID Controller, hãy đi qua một số mô phỏng và xem kết quả để nhận xét nhé:

Mô hình mô phỏng: Dùng PID module của Xcos trong Scilab và hiện thực có dùng biến đổi Laplace(Phần này chưa hoàn thiện do chưa xác định đúng Ts và N)

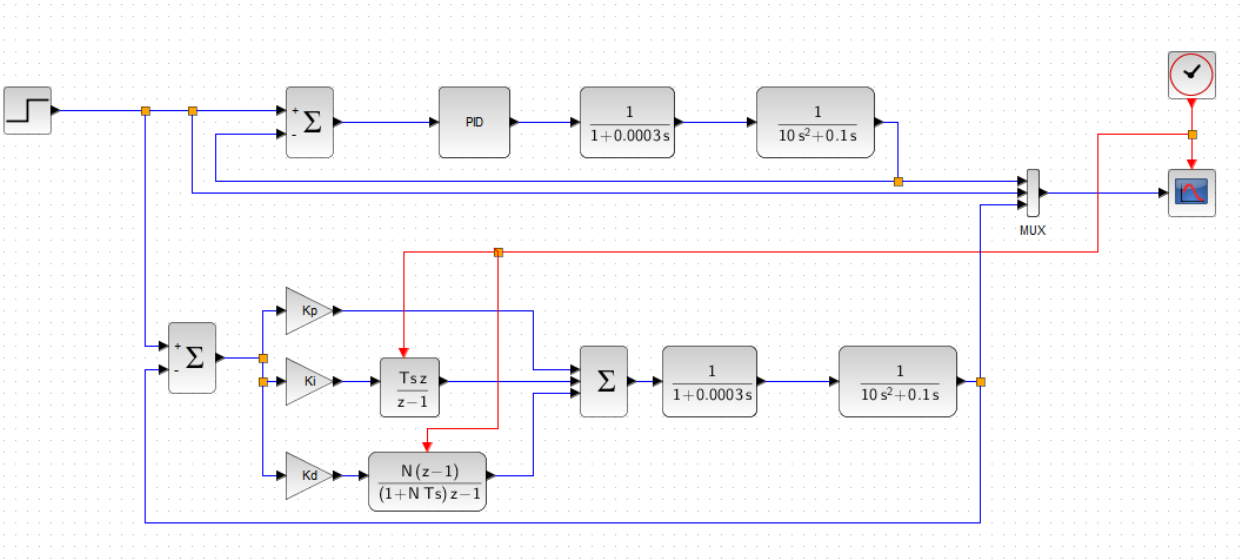

Figure 8. Mô phỏng PID

1. $K_P = 200, K_I = 400, K_D = 42$

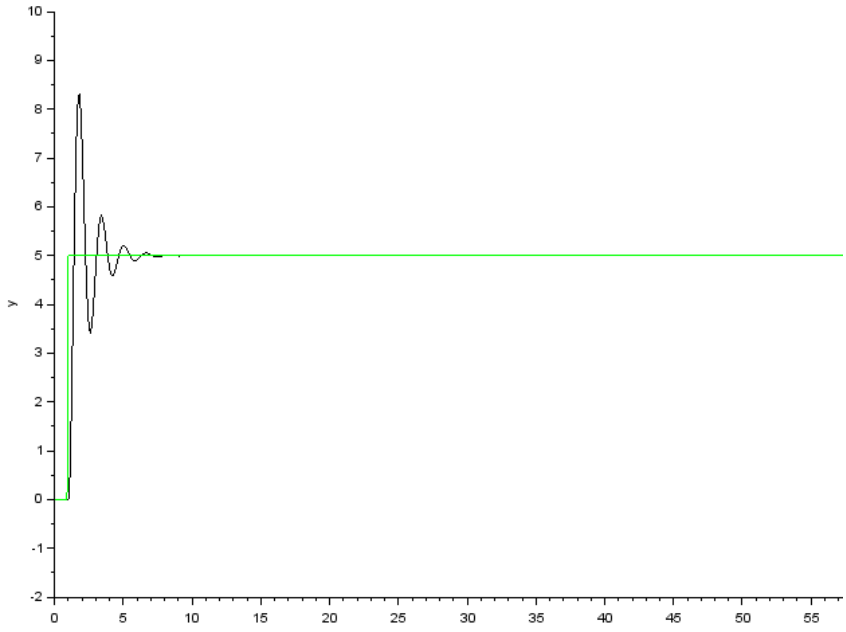

Figure 9. Tín hiệu dao động nhiều

Đây là tín hiệu của một bộ PID hoàn chỉnh với các hằng số như trên, ta thấy rằng tín hiệu điều khiển sẽ hướng về một đường chuẩn màu xanh(tín hiệu mong muốn), tuy nhiên do Kd trong trường hợp này không lớn so với Kp và Ki nên độ dao động tín hiệu vẫn còn(do Kd không kìm hãm kịp sự tăng của hai tín hiệu điều khiển kia).

2. $K_P = 200, K_I = 400, K_D = 150$

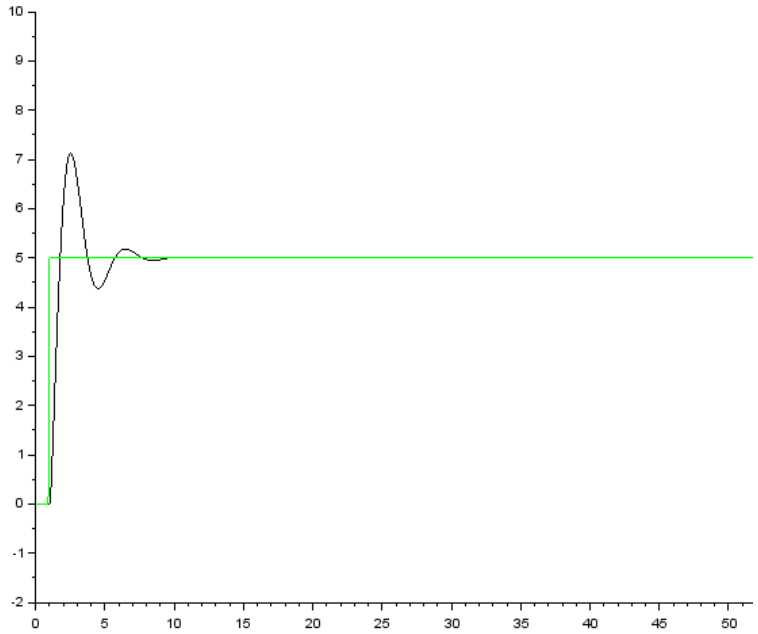

Figure 10. Tín hiệu dao động đã giảm

Khi Kd tăng lên, độ dao động cũng đã giảm và tín hiệu ổn định hơn, nhưng do Ki quá lớn nên tính trạng **overshooting** vẫn diễn ra

3. $K_P = 200, K_I = 50, K_D = 150$

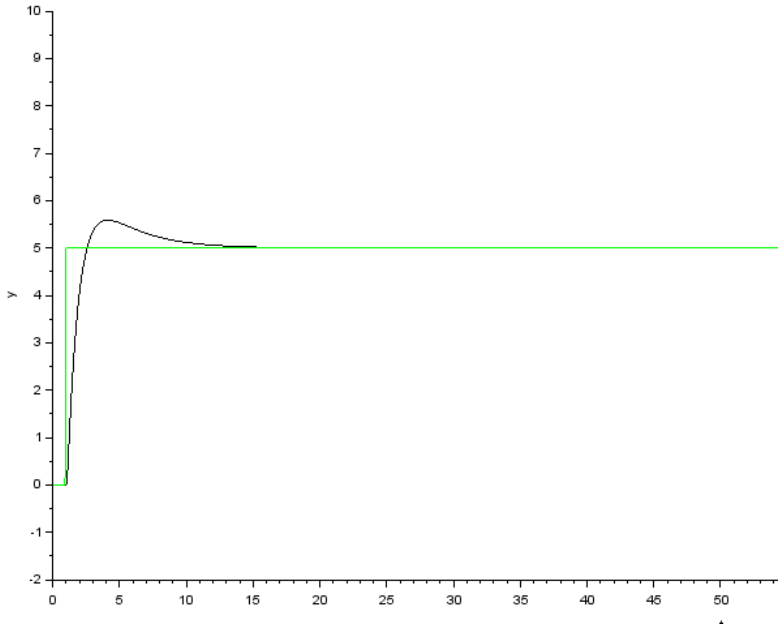

Figure 11. Không còn dao động

4. $\boldsymbol{K_P = 200, K_I = 10, K_D = 100}$

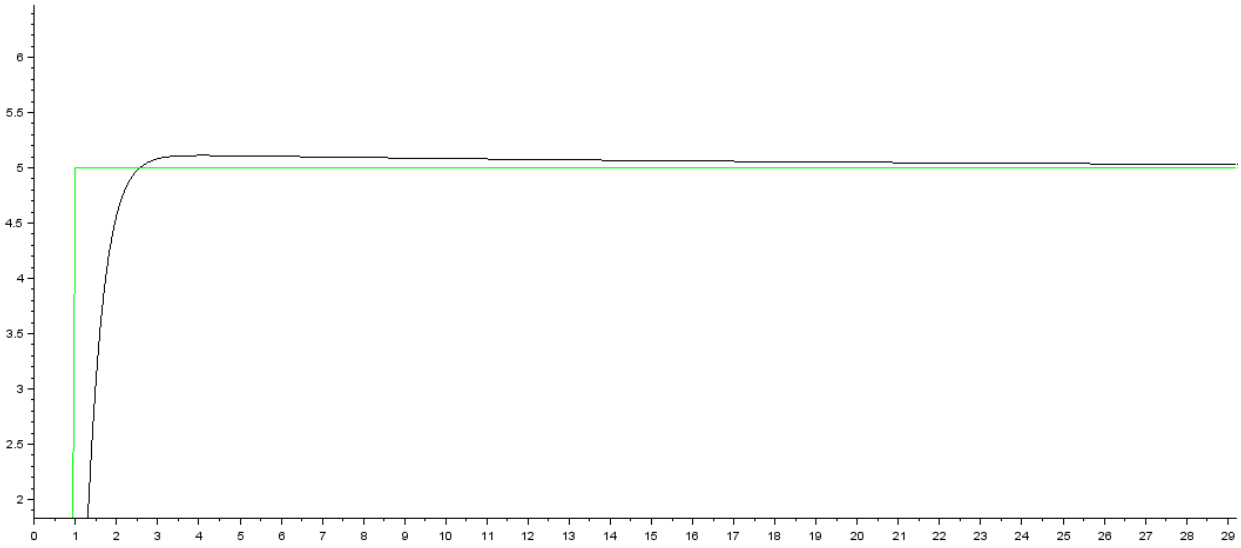

Figure 12. Trạng thái steady state error

Trạng thái tín hiệu rất ổn định nhưng vẫn không thể bằng được tín hiệu xanh(tín hiệu mong muốn) do Ki quá nhỏ so với Kp.

VII. PHỤ LỤC VÀ TÀI LIỆU THAM KHẢO

Việc điều chỉnh các hệ số cần một sự chính xác và liên quan mật thiết, các bạn có thể tìm hiểu thêm về vấn đề này ở:
https://robotics.stackexchange.com/questions/167/what-are-good-strategies-for-tuning-pid-loops
https://www.youtube.com/watch?v=sFOEsA0Irjs
https://en.wikipedia.org/wiki/Ziegler%E2%80%93Nichols_method

[1] The Working Principle of a PID Controller for Beginners
https://www.elprocus.com/the-working-of-a-pid-controller/
[2] Understanding PID Control
https://www.youtube.com/watch?v=wkfEZmsQqiA&list=PLn8PRpmsu08pQBgjxYFXSsODEF3Jqmm-y
[3] Using a Fixed-Point Digital Signal Processor as a PID Controller
http://www.ni.com/pdf/academic/us/journals/lv02_65.pdf
[4] Hardware Demo of a Digital PID Controler
https://www.youtube.com/watch?v=fusr9eTceEo

Truy cập lần cuối ngày 6/1/2020.