ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC BÁCH KHOA

KHOA KĨ THUẬT GIAO THÔNG

Bộ môn : Ô-tô – Máy động lực

LUẬN VĂN TỐT NGHIỆP ĐẠI HỌC

THIẾT KẾ HỆ THỐNG ĐIỆN ĐIỀU KHIỂN BƯỚM GA ĐIỆN TỬ DÙNG ARDUINO VÀ LABVIEW

SVTH : Bùi Tuấn Anh

MSSV : 1510035

GVHD : TS. Trần Đăng Long

Thành phố Hồ Chí Minh, tháng 12 năm 2020

MỤC LỤC

DANH MỤC HÌNH

DANH MỤC BẢNG

LỜI NÓI ĐẦU

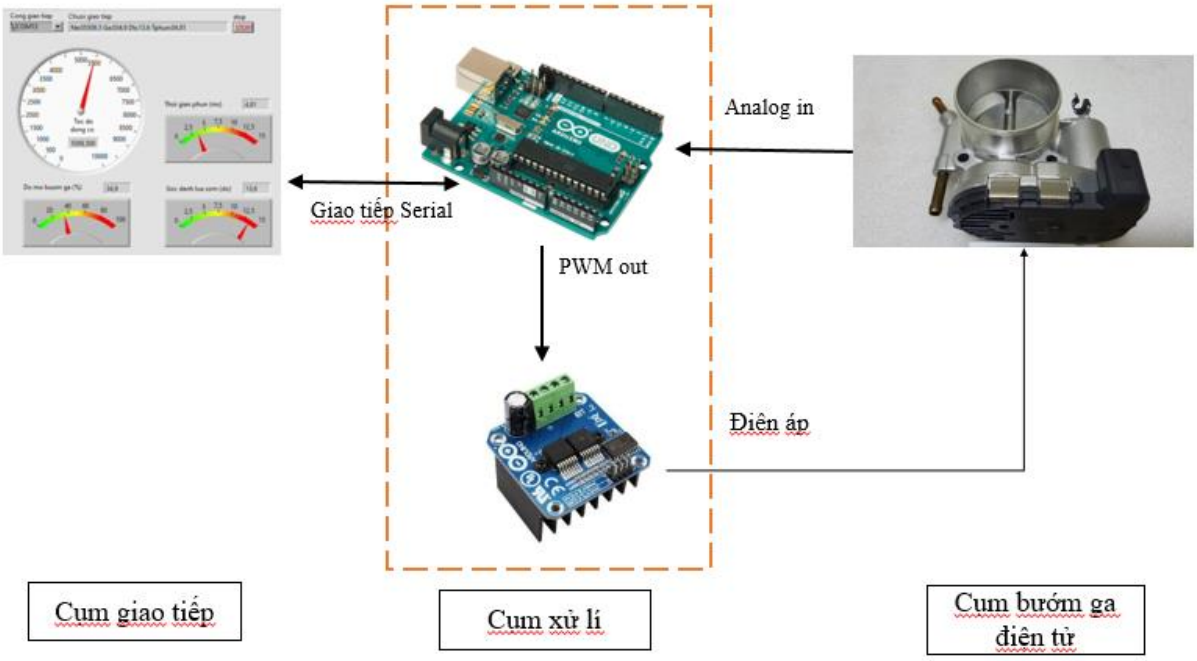
Động cơ đốt trong là một trong những phát minh mang tính cách mạng đối với lịch sử loài người, là động lực to lớn đưa thế giới tiến đến thời đại công nghiệp. Trải qua một quá trình phát triển kéo dài hàng trăm năm, bên cạnh việc liên tục nâng cao hiệu suất và công suất của động cơ, con người luôn không ngừng nỗ lực để nghiên cứu các giải pháp tối ưu hóa công việc điều khiển động cơ. Ở thời kì những năm 20 của thế kỉ 21, cùng với sự phát triển như vũ bão của công nghệ số, các giải pháp điều khiển động cơ bằng điện tử lập trình đang dần thay thế các loại điều khiển cơ khí truyền thống, mang đến khả năng điều khiển chính xác và thông minh hơn.

Trong giai đoạn cách mạng 4.0 hiện nay, khi thế giới đang chuyển mình sang thời kỳ điện khí hóa, ngành công nghiệp ô tô cũng bước vào một cuộc cách mạng chưa từng có, đó là sự xuất hiện của xe điện tự hành đe dọa đến sự thống trị của những chiếc xe chạy bằng nhiên liệu hóa thạch. Nhưng ai biết được viễn cảnh nào sẽ đến trong tương lai. Do vậy, việc học tập và làm chủ kiến thức về lập trình đối với sinh viên ngành ô tô đang ngày càng trở nên quan trọng hơn bao giờ hết.

Nhóm em chọn đề tài "Lập trình điều khiển và số hóa 3D bướm ga điện tử" để nghiên cứu tìm hiểu về lập trình nhúng bằng Arduino cũng như phần mềm mô phỏng Labview. Trải qua một học kỳ thực hiện luận văn, trên cơ sở vận dụng kiến thức của những môn học chuyên ngành, đến nay đề tài của nhóm chúng em đã hoàn thành. Mặc dù có nhiều nguồn tài liệu để tham khảo, đặc biệt là trên internet, nhưng thời gian và kiến thức của em có hạn nên không tránh khỏi những sai sót trong quá trình thực hiện, rất mong nhận được sự góp ý và nhận xét từ phía hội đồng đánh giá.

Em xin gửi lời cảm ơn đến thầy Trần Đăng Long đã tận tình hướng dẫn, đưa ra gợi ý và chỉ ra những lỗi sai mà nhóm em mắc phải, giúp em sửa lỗi. Nhờ đó nhóm em mới có thể hoàn thành luận văn đúng hạn.

Chương I. GIỚI THIỆU ĐỀ TÀI



Hình 1.1. Sơ đồ bố trí thực tế hệ thống điều khiển điện

Với sự phát triển của công nghệ hiện đại trong lĩnh vực ô tô, xu hướng tự động hóa các quá trình hoạt động trên ô tô ngày càng thể hiện rõ lợi thế cạnh tranh giữa các doanh nghiệp sản xuất ô tô trên khắp thế giới. Trong đó, việc phát triển hệ thống điều khiển độ mở bướm ga đóng vai trò quan trọng trong việc tối ưu hóa hiệu suất làm việc của động cơ, giúp tiết kiệm nhiên liệu, tối ưu hóa lượng và thành phần vật chất khí thải. Vì thế việc thử nghiệm thiết kế một hệ thống điều khiển điện tử với nhiệm vụ điều khiển độ mở bướm ga là một việc làm thiết thực.

1. Đối tượng

Hệ thống điện – điện tử điều khiển bướm ga điện tử dùng Arduino và Labview.

2. Bài toán cần giải quyết

Để giải quyết yêu cầu đề bài, hệ thống phải giải quyết được các bài toán sau đây :

- Điều chỉnh được độ mở bướm ga mong muốn : phải xác lập được giá trị độ mở bướm ga cần đạt được

- Thiết kế giải thuật tính toán giá trị xung PWM điều khiển motor điện : cần tính toán được giá trị điện áp cần thiết để điều khiển motor điện để làm mở bướm ga.

- Thiết kế hệ thống điện điều khiển motor điện : từ giá trị điện áp đã tính toán, sử dụng giá trị điện áp đó để điều khiển motor điện.

3. Điều kiện làm việc

a) Điều kiện làm việc :

- Chịu tần số làm việc tương đối cao : bộ điều khiển và các cơ cấu chấp hành hoạt động ở tần số cao (100Hz), biến đổi liên tục.

b) Yêu cầu kĩ thuật :

- Đọc và xử lý được các tín hiệu điện áp đầu vào;

- Tính toán chính xác giá trị xung điện thích hợp điều khiển động cơ bướm ga;

- Điều khiển được độ mở bướm ga.

4. Giới hạn nội dung thực hiện

- Thiết kế giao diện điều khiển bằng Labview

- Thiết kế thuật toán, chương trình tính toán giá trị xung điện điều khiển bướm ga điện tử.

- Thiết kế một hệ thống điện – điện điện tử điều khiển bộ bướm ga điện tử

Chương II. CƠ SỞ LÝ THUYẾT

Để thiết kế được một hệ thống điều khiển bướm ga điện tử, trước hết ta phải hiểu được cấu tạo cũng như nguyên lí làm việc của các cụm, chi tiết chính trong hệ thống.

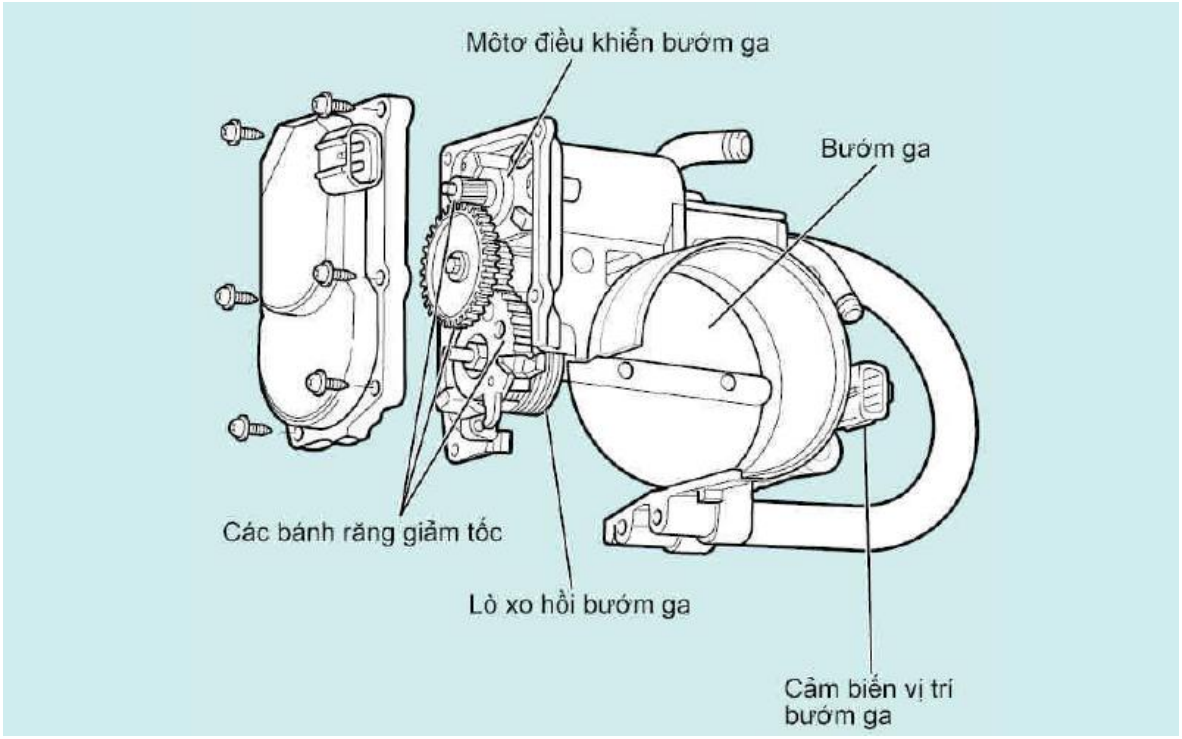
Ở chương này, chúng ta sẽ tìm hiểu các thành phần chính bộ điều khiển điện tử :

- Bộ bướm ga điện tử
- Mạch cầu H
- Board mạch Arduino.

1. Bướm ga điện tử :

1.1 Cấu tạo:

Bướm ga điện tử là một module điều khiển của động cơ phun nhiên liệu điện tử, có chức năng thay đổi lượng không khí nạp vào buồng đốt động cơ thông qua việc điều khiển độ mở cánh bướm ga bằng motor DC một chiều.



Hình 2.1. Cấu tạo cụm bướm ga điện tử [1]

Các bộ phận chính của 1 cụm bướm ga điện tử gồm có : Motor điều khiển bướm ga, các bánh răng giảm tốc, cảm biến vị trí bướm ga, (van) bướm ga và lo xo hồi vị bướm ga.

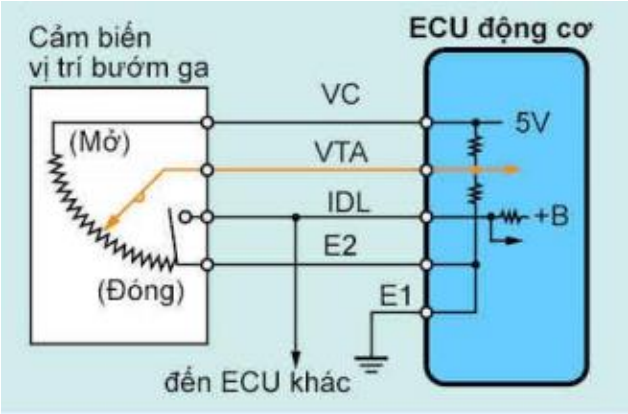
Khi cấp điện cho động cơ, moment từ động cơ sẽ truyền qua các bánh răng giảm tốc, làm quay bướm ga một góc nhất định. Khi moment của động cơ và moment phản lực của lò xo hồi vị cân bằng thì bướm ga sẽ ở một vị trí cố định. Khi ngừng cung cấp

điện cho motor, lo xo hồi vị sẽ có vài trò khép bướm ga lại. Sự thay đổi vị trí của bướm ga sẽ được đọc bởi cảm biến vị trí bướm ga và xuất ra ngoài ở dạng tín hiệu điện áp.

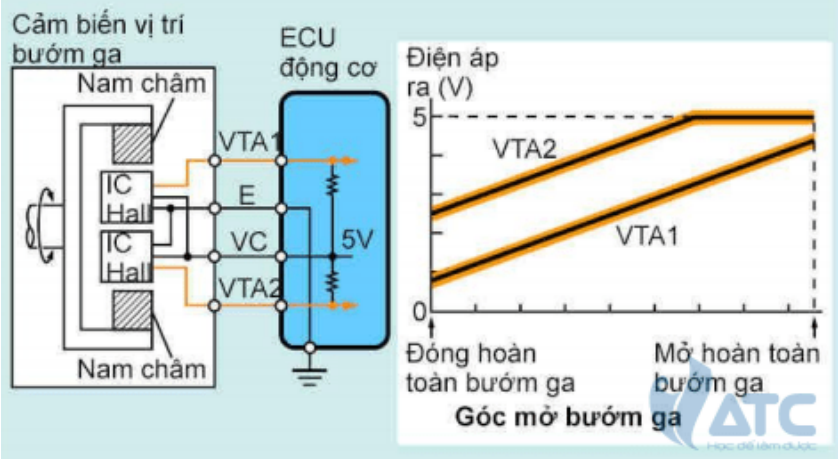
1.2 Cảm biến vị trí bướm ga:

Cảm biến vị trí bướm ga (TPS Sensor) được sử dụng để đo độ mở vị trí của cánh bướm ga để báo về hộp ECU. Từ đó, ECU sẽ sử dụng thông tin tín hiệu mà cảm biến vị trí bướm ga gửi về để tính toán mức độ tải của động cơ nhằm hiệu chỉnh thời gian phun nhiên liệu, cắt nhiên liệu, điều khiển góc đánh lửa sớm, điều chỉnh bù ga cầm chừng và điều khiển chuyển số.

Cảm biến vị trí bướm ga dùng cho bướm ga điện tử có 2 loại chính: loại tuyến tính và loại phần tử Hall.



a) Loại biến trở



b) Loại Hall và đồ thị đặc tính chung

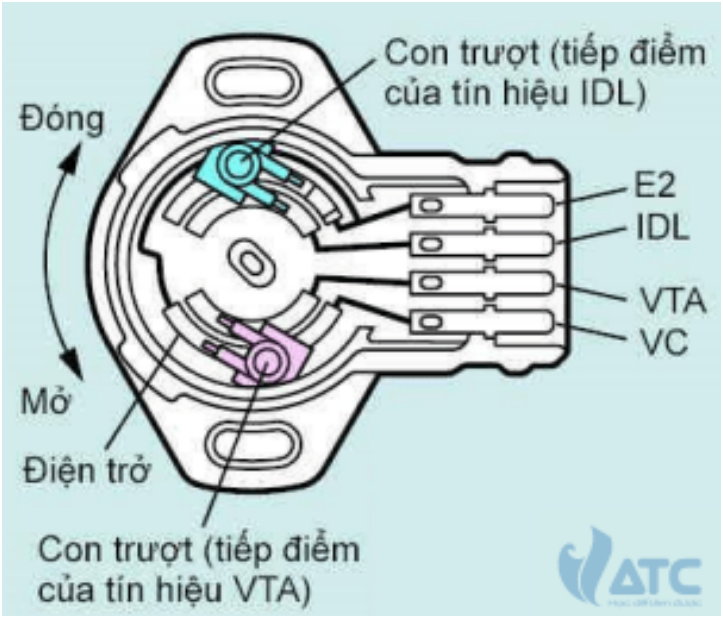
Hình 2.2 Sơ đồ nguyên lý và đường đặc tính của cảm biến vị trí bướm ga loại tuyến tính [1]

a) Cảm biến sử dụng biến trở

b) Cảm biến sử dụng IC Hall

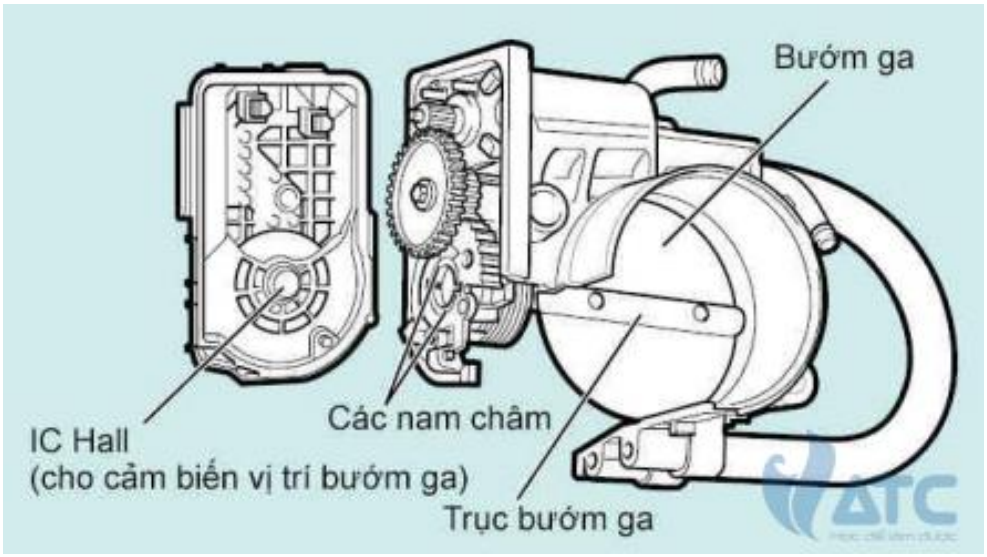
Nguyên lí chung cảm biến vị trí bướm ga : khi bướm ga mở ra/đóng lại do đạp chân ga, cảm biến bướm ga sẽ ghi lại sự đóng mở đó bằng cách chuyển hóa góc mở bướm ga thành một giá trị điện áp.

Đối với cảm biến loại biến trở, sự thay đổi vị trí bướm ga sẽ tương ứng với sự thay đổi vị trí con chạy trên biến trở.



Hình 2.3. Cảm biến bướm ga loại biến trở [1]

Đối với cảm biến loại Hall, 2 IC Hall sẽ đọc tín hiệu từ các nam châm được bố trí đồng trục với van bướm ga.



Hình 2.4. Cảm biến bướm ga loại Hall [1]

Giá trị đầu ra của cảm biến vị trí bướm ga sẽ bao gồm 2 giá trị điện áp

+ VTA1 : dùng để xác định độ mở bướm ga

+ VTA2 : dùng để kiểm tra sai số của VTA1. Ở trạng thái cảm biến hoạt động tốt, sai khác giữa giá trị VTA1 và 0,8.VTA2 được giữ ở khoảng 1.11V. [2]

2. Mạch cầu H :

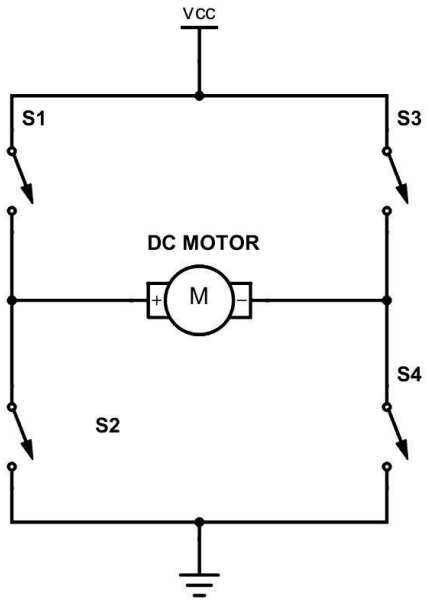
Để điều khiển chiều quay và moment quay cho motor điện, ta sử dụng mạch cầu H.

2.1 Cấu tạo:

Mạch cầu H gồm 4 công tắc 1 chiều S1, S2, S3, S4 là các transistor. Trong đó :

- S1 và S2 tạo nên nửa cầu trái, điều khiển cực dương của motor .

- S3 và S4 tạo nên nửa cầu phải, điều khiển cực âm của motor.



Hình 2.5. Sơ đồ nguyên lý cấu tạo mạch cầu H

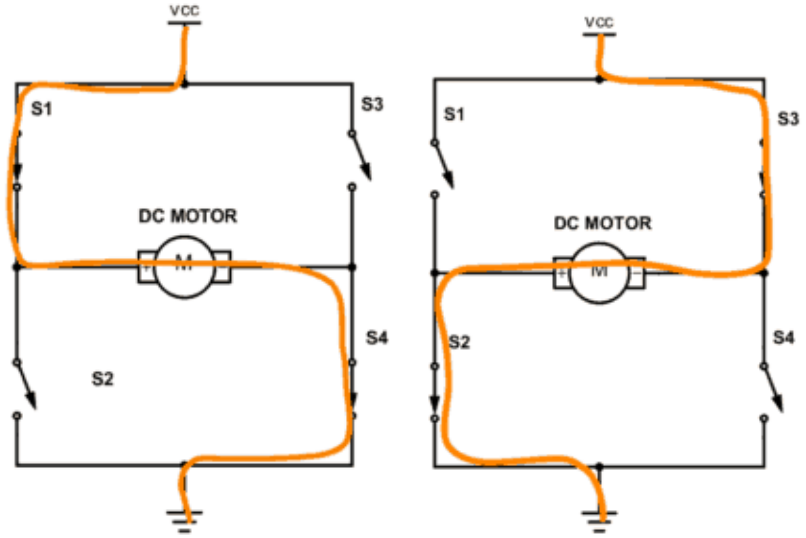
2.2. Nguyên lí hoạt động cầu H :

Để điều khiển mạch cầu H, ta thay đổi điện áp điều khiển các công tắc S1, S2, S3, S4. Bằng cách thay đổi điện áp điều khiển các công tắc S1, S2, S3, S4 ta có thể điều khiển chiều quay và moment quay của motor.

2.2.1. Điều khiển chiều quay :

Để điều khiển chiều quay động cơ với mạch cầu H, ta điều khiển từng cặp transistor hay nói cách khác là cặp khóa (S1,S4) và (S2,S3)

- Trường hợp điều khiển quay theo chiều thuận : Ta đóng cặp khóa (S1, S4). Lúc này dòng điện sẽ đi từ nguồn qua S1 đến cực dương của động cơ rồi từ cực âm qua S4 đến mass.
- Trường hợp điều khiển quay theo chiều nghịch : Ta đóng cặp khóa (S2, S3). Lúc này dòng điện sẽ đi từ nguồn qua S3 đến cực âm động cơ rồi từ cực dương qua S2 về mass.



a) b)

Hình 2.6. Sơ đồ nguyên lí hoạt động mạch cầu H.

a) Trường hợp điều khiển quay theo chiều thuân
b) Trường hợp điều khiển quay theo chiều nghịch

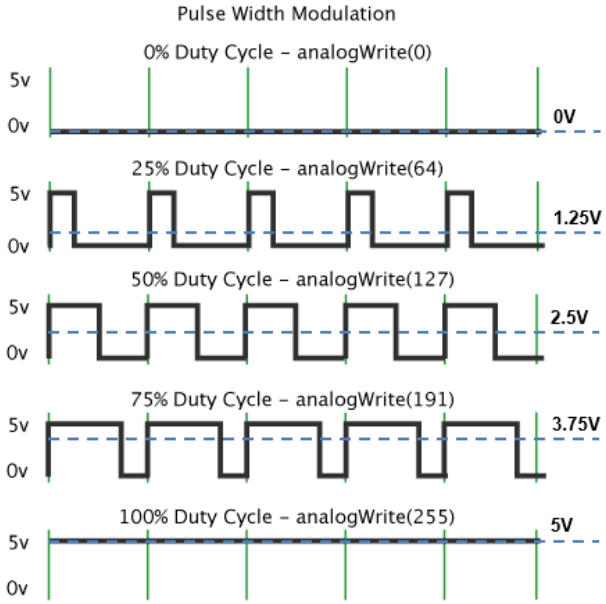
2.2.2. Điều khiển moment quay động cơ – tín hiệu PWM :

Để thay đổi moment quay của động cơ, ta phải thay đổi giá trị điện áp điều khiển các công tắc ở phía nguồn (S1, S3). Giá trị điện áp điều trên sẽ được xác lập bởi một tín hiệu PWM.

Đặc trưng của một tín hiệu PWM

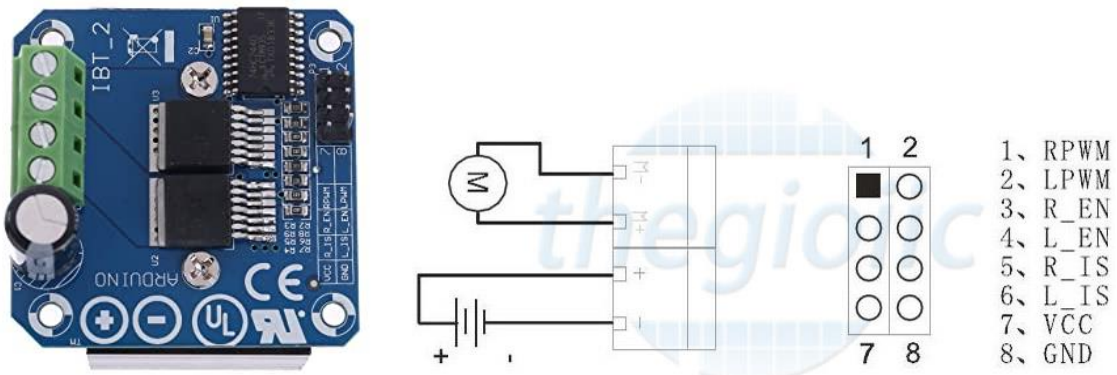
- Chu kì xung : chu kì xung của một tín hiệu PWM là một hằng số
- Độ rộng xung : thời gian (i %) xung hoạt động ở mức điện áp cao. Trong thời gian còn lại, xung làm việc ở điện áp thấp. Giá trị của độ rộng xung là từ 0 – 100% chu kì xung.
- Điện áp trung bình : bằng tích của độ rộng xung với điện áp cực đại ở mức điện áp cao. $U_{tb} = U_{max} * i$ %

Bằng các thay đổi độ rộng xung PWM, ta có thể thay đổi điện áp trung bình của một tín hiệu PWM từ 0 đến 100% giá trị điện áp cực đại, tức là từ 0 - Vcc



Hình 2.7. Các tín hiệu PWM với độ rộng xung khác nhau.

2.3. Mạch cầu H BTS7960:



a) b)

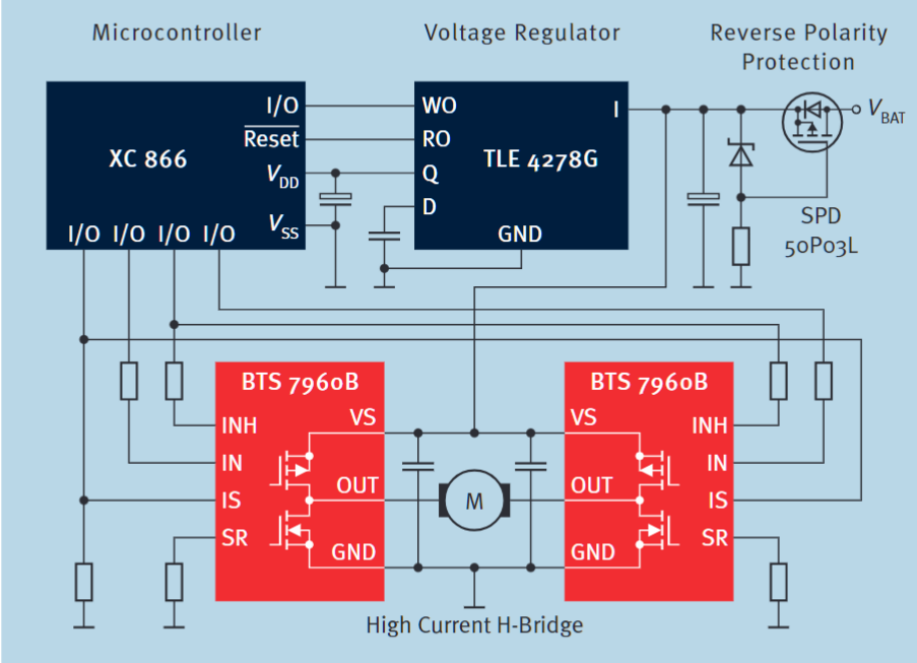
Hình 2.8. Mạch cầu H BTS 7960 [5]

a) Mạch cầu H BTS7960

b) Sơ đồ chân mạch cầu H BTS7960

Để đơn giản việc điều khiển cầu H, ta sử dụng mạch cầu H BTS7960.

Với mạch cầu H BTS7960, thay vì điều khiển trực tiếp 4 công tắc, ta chỉ cần điều khiển tín hiệu chiều quay và độ rông xung PWM cho IC, từ IC sẽ điều khiển 4 công tắc tương ứng với chiều quay và độ rộng xung ta cấp vào.



Hình 2.9. Sơ đồ nguyên lí Mạch cầu H BTS7960 [5]

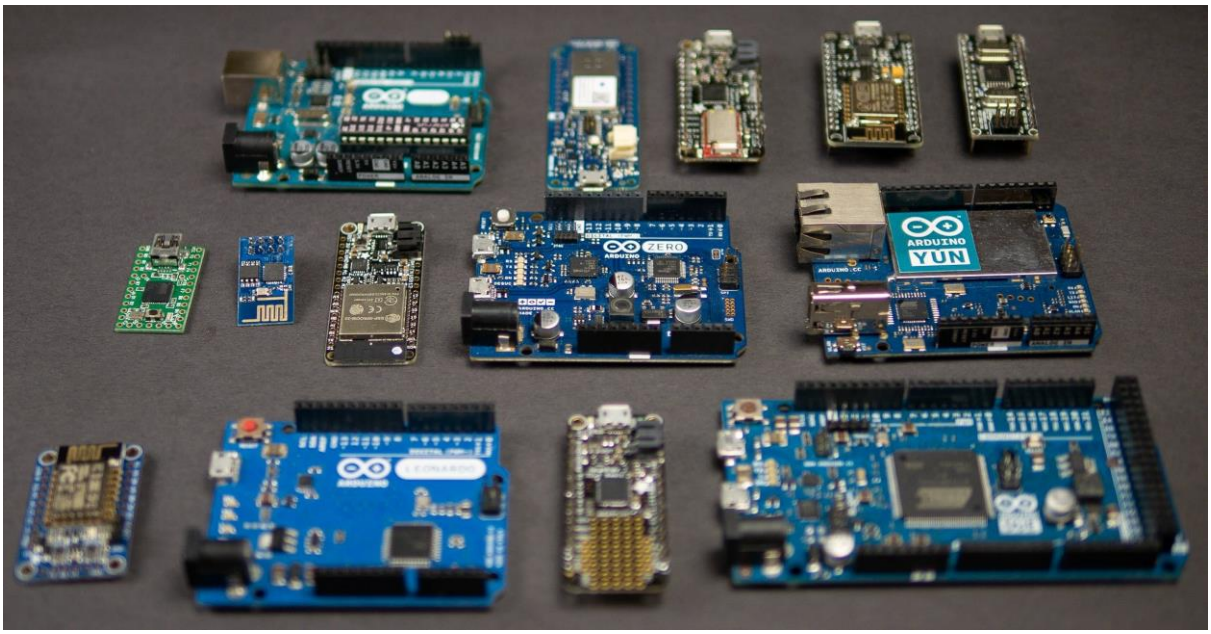
Mode	L_PWM	R_PWM
Điều khiển động cơ quay thuận	HIGH	LOW
Điều khiển động cơ quay nghịch	LOW	HIGH
Điều khiển tốc độ quay thuận	PWM	LOW
Điều khiển tốc độ quay nghịch	LOW	PWM

Bảng 2.1. Các chế độ điều khiển động cơ bằng mạch cầu H

3. Arduino :

Để tính toán giá trị độ rộng xung PWM, xuất tín hiệu PWM điều khiển mạch cầu H, ta sử dụng board mạch Arduino.

Arduino là một nền tảng mã nguồn mở dựa trên ngôn ngữ lập trình C. Arduino bao gồm một phần mềm lập trình trên máy tính và phần cứng là các phiên bản board mạch Arduino phục mục đa dạng các mục đích sử dụng.



Hình 2.10. Một số loại mạch Arduino trên thị trường [6]

3.1. Cấu tạo :

Arduino Uno R3 là một board mạch sử dụng chip Atmega328P.

Hình 2.11. Mạch Arduino Uno R3 [7]

3.2. Chức năng :

Mạch Arduino Uno R3 cung cấp cho ta :

- 14 chân input/output kĩ thuật số (digital) trong đó có 6 chân PWM
- 6 chân input analog : A0 → A5.

- 3 bộ định thời (Timer/Count) : TIMER0, TIMER1, TIMER2 hỗ trợ tạo chu kì cố định thực hiện một số chức năng
- Một chip vi xử lý 8 bit hoạt động ở tần số 16MHZ.

Bằng Arduino, trong khuôn khổ luận văn này sẽ được dùng để tạo chu kì chuẩn để đọc các giá trị độ mở bướm ga mong muốn, độ mở bướm ga thực tế sau đó tính toán ra giá trị PWM điều khiển và xuất giá trị PWM đó ra mạch cầu H để điều khiển động cơ điện trong bộ bướm ga điện tử.

Chương III. THIẾT KẾ BỐ TRÍ CHUNG

Sau khi xác định các thành phần chính của hệ thống, chúng ta phân chia các thành phần chính thành các cụm chức năng đảm nhiệm các nhiệm vụ đặc trưng.

Để giải quyết bài toán ở đề tài, hệ thống điều khiển được xác định gồm 3 cụm chức năng :

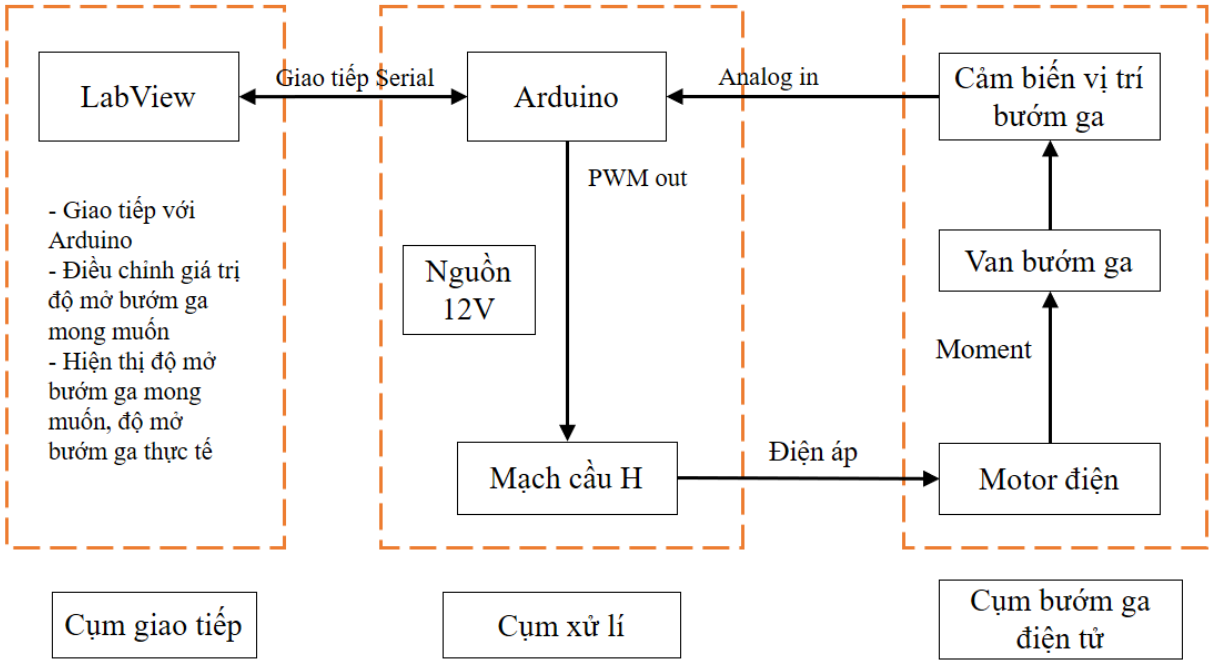
- Cụm giao tiếp
- Cụm xử lí tín hiệu
- Cụm bướm ga điện tử

Với các cụm chức năng đã được bố trí, ta cũng cần phải xác định được đường đi của các tín hiệu, tương tác giữa cụm chức năng.

Trong phần này, ta sẽ giải quyết các vấn đề sau :

1. Sơ đồ khối bố trí chung
2. Tuyến điều khiển PID

1. Sơ đồ khối bố trí chung :



Hình 3.1 Sơ đồ bố trí chung hệ thống điều khiển điện

Hệ thống điều khiển điện tử gồm 3 cụm chính

1.1 Cụm giao tiếp:

Sử dụng phần mềm Labview trên máy tính, cụm giao tiếp có vai trò thực hiện các nhiệm vụ :

- Mô phỏng : Mô phỏng phần tử điều khiển là một biến trở điều khiển độ mở vị trí bướm ga mong muốn (thay cho bàn đạp chân ga)

- Gửi thông tin : Labview sẽ thực hiện giao tiếp Serial với Arduino, gửi giá trị mô phỏng đến cho Arduino qua cổng giao tiếp COM

- Nhận thông tin : Thu thập các giá trị độ mở bướm ga mong muốn, độ mở bướm ga thực tế và giá trị PWM tính toán ở mỗi chu kì giao tiếp. Sau đó thể hiện các giá trị trên một cách trự quan ở dạng các đồ thị.

1.2. Cụm xử lí tín hiệu:

Cụm xử lí tín hiệu gồm một board mạch Arduino và một mạch cầu H.

Vai trò của board mạch Arduino :

- Đọc và tính toán giá trị độ mở bướm ga mong muốn từ Labview

- Đọc và tính toán giá trị độ mở bướm ga thực tế từ cảm biến vị trí bướm ga.

- Tính toán giá trị độ rộng xung PWM điều khiển cần thiết.

- Tạo chu kì đọc để cập nhật các giá trị độ mở bướm ga mong muốn và thực tế.

- Dựa trên tín hiệu PWM đã tính toán để điều khiển mạch cầu H

Mạch cầu H : từ các tín hiệu điều khiển nhận được từ Arduino, mạch cầu H sẽ xuất xung điện dưới dạng điện áp để điều khiển motor điện.

1.3. Cụm bướm ga điện tử:

- Motor điện : từ giá trị điện áp từ cầu chữ H. motor điện sẽ tạo ra moment cơ học làm xoay bướm ga.

- Cảm biến vị trí bướm ga : thu thập giá trị độ mở bướm ga thực tế, cung cấp cho Arduino để sử dụng cho chu kì tính toán tiếp theo.

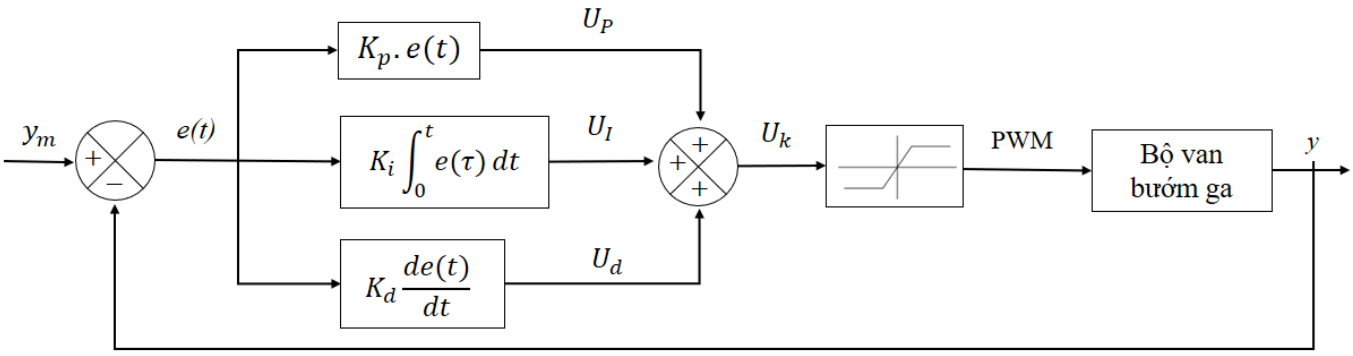
- Van bướm ga : moment quay của motor kết hợp với moment cản từ lò xo hồi vị của bướm ga tạo thành độ mở thực tế của bướm ga. Độ mở này sẽ được đọc bởi cảm biến vị trí bướm ga.

2. Tuyến điều khiển với bộ PID

Để tính toán được độ rộng xung PWM cần thiết để điều khiển động cơ điện, ta sử dụng thuật toán PID.

Sử dụng thuật toán PID, thông số điều khiển sẽ được tính toán dựa trên thông số cố định và thông số phản hồi. Một bộ PID hoàn chỉnh được xác định bởi các thông số :

setpoint (giá trị thiết lập), input (giá trị phản hồi), output (giá trị tính toán), (Kp, Ki, Kd) là các thông số điều chỉnh. Sơ đồ thuật toán PID được mô tả như sau:



Hình 3.2 Sơ đồ thuật toán PID của bộ điều khiển

Giá trị độ rộng xung PWM sẽ được tính gián tiếp thông qua giá trị tín hiệu điều khiển PID là Uk, sau đó qua một khâu quy đổi để được giá trị độ rộng xung PWM.

Gọi :

y_m : giá trị độ mở bướm ga mong muốn

y : giá trị độ mở bướm ga thực tế

$e = y_m - y$: sai số độ mở bướm ga

Giá trị tín hiệu điều khiển PID Uk được tính bằng tổng thành bởi 3 thành phần :

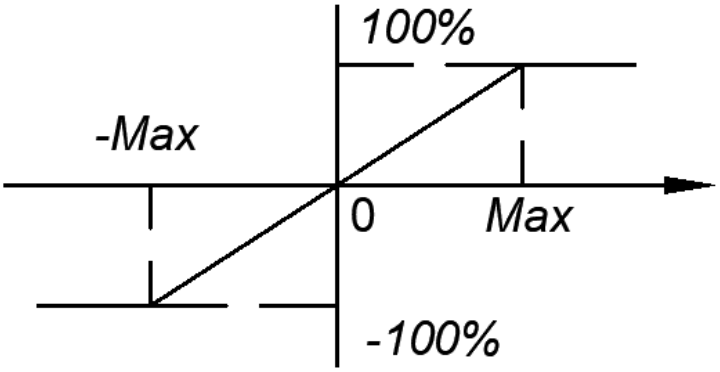
+ $U_p = K_p.e(t)$: thành phần tín hiệu điều khiển tỉ lệ với sai số. Thành phần này gọi là thành phần tỉ lệ.

+ $U_I = K_i \int_0^t e(\tau)\,dt$: thành phần tín hiệu điều khiển tỉ lệ với tổng sai số theo thời gian. Thành phần này gọi là thành phần tích phân.

+ $U_d = K_d \frac{de(t)}{dt}$: thành phần tín hiệu điều khiển tỉ lệ với biến thiên sai số theo thời gian. Thành phần này gọi là thành phần vi phân.

Lấy tổng các thành phần trên ta được thành phần tín hiệu điều khiển PID

$$U_k = U_p + U_i + U_d = K_p.e(t) + K_i \int_0^t e(\tau)\,dt + K_d \frac{de(t)}{dt}$$



Hình 3.3 Khâu quy đổi

Giá trị điện áp tính toán từ PID sẽ được quy đổi thành giá trị PWM qua một khâu quy đổi. Các giá trị U_k nhỏ hơn *–Max* sẽ tương ứng với độ rộng xung -100% còn các giá trị lớn hơn *Max* sẽ tương ứng với độ rộng xung 100%. Còn với các giá trị nằm trong khoảng *(-Max; Max*) thì độ rộng xung PWM tương ứng được tính như sau:

$$PWM = U_k * 100 / Max \ (\%)$$

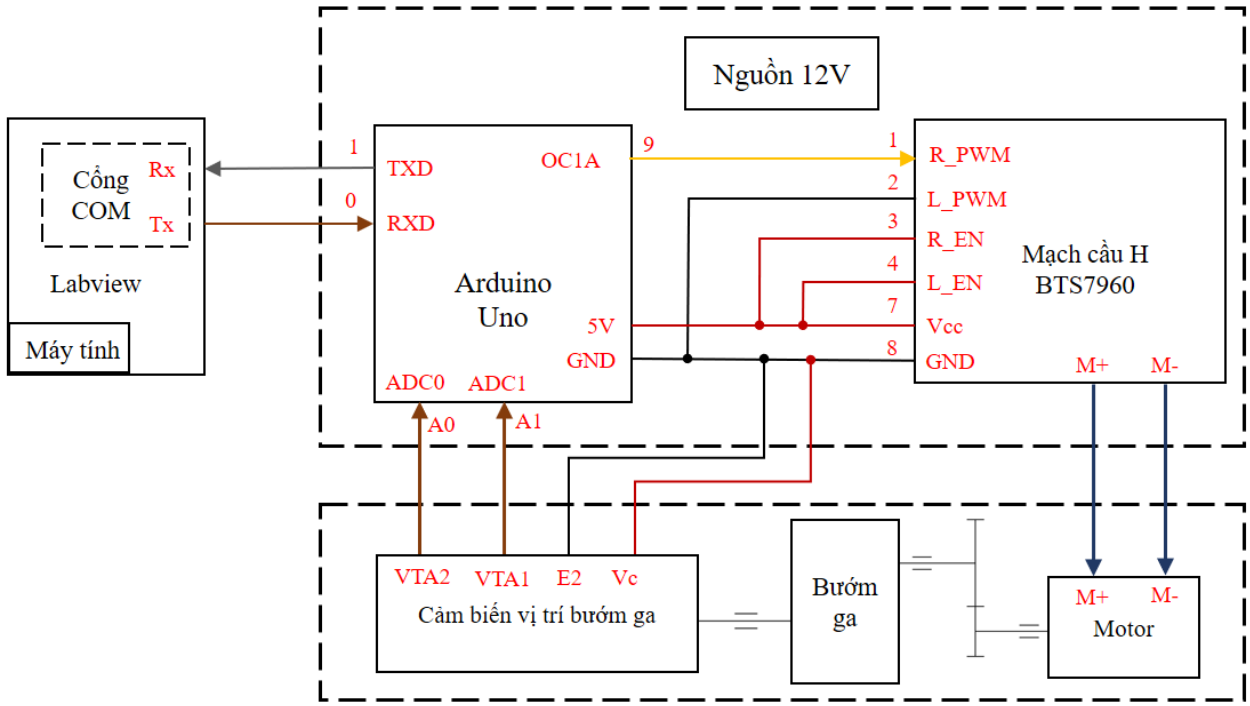
Tùy theo việc lựa chọn các hệ số Kp, Ki, Kd, việc tính toán giá trị PWM sẽ diễn ra với tốc độ và độ ổn định khác nhau.

Bằng phương pháp thực nghiệm ta chọn ra bộ số (Kp, Ki, Kd) tối ưu cho đề bài.

Chương IV. THIẾT KẾ KĨ THUẬT

1. Sơ đồ dấu dây

Sau khi xác định được thiết kế bố trí chung của hệ thống, ta tiến hành đấu nối dây các thành phần của bộ điều khiển theo sơ đồ dưới đây.



Hình 4.1. Sơ đồ đấu dây hệ thống điều khiển điện.

Chân số 0 và 1 (tương ứng với các chân giao tiếp RXD và TXD) của Arduino có nhiệm vụ nhận và gửi thông tin, thực hiện giao tiếp với phần mềm Labview trên máy tính qua cổng COM. Trong đó :

- Tín hiệu nhận gồm có : giá trị Ax để tính độ mở bướm ga mong muốn mm%.

- Giá trị gửi gồm có : mm%, tt% và PWM.

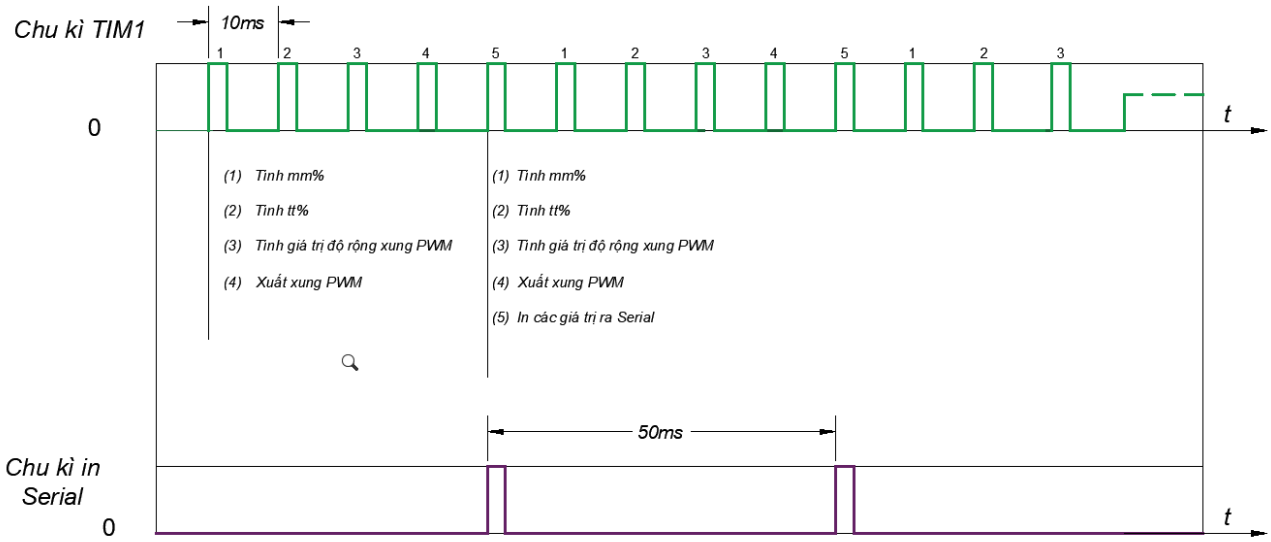
Arduino sẽ đọc tín hiệu vị trí bướm ga từ cảm biến vị trí bướm ga (TPS) qua 2 chân analog A0 và A1, tương ứng được nối với chân VTA2 và VTA1 của cảm biến. Vì giá trị VTA2 có tác dụng kiểm ra nên chỉ sử dụng giá trị VTA 1 trong tính toán và lập trình.

Sau khi tiến hành tính toán, giá trị độ rộng xung PWM sẽ được xuất ra ở chân Digital số 9, tương ứng với chân OC1A là PWM A của Timer 1.

Tín hiệu PWM này sẽ được cấp vào chân số 1 (chân R_PWM) để điều khiển cầu trái của mạch cầu H. Chân số 2 (chân L_PWM của mạch cầu H) sẽ luôn được set tín hiệu xung thấp (LOW) bằng cách nối với chân GND của Arduino.

Các chân M+ và M- trên mạch cầu H sẽ được nối tương ứng với cực dương và âm cảu motor điện trong bộ bướm ga điện tử.

2. Giản đồ công việc theo thời gian



Hình 4.2. Giản đồ công việc theo thời gian

Mỗi chu kì Timer, sẽ lần lượt thực hiện các công việc :

(1) Tính độ mở bướm ga mong muốn

(2) Đọc giá trị độ mở bướm ga thực tế

(3) Tính toán độ rộng xung PWM điều khiển

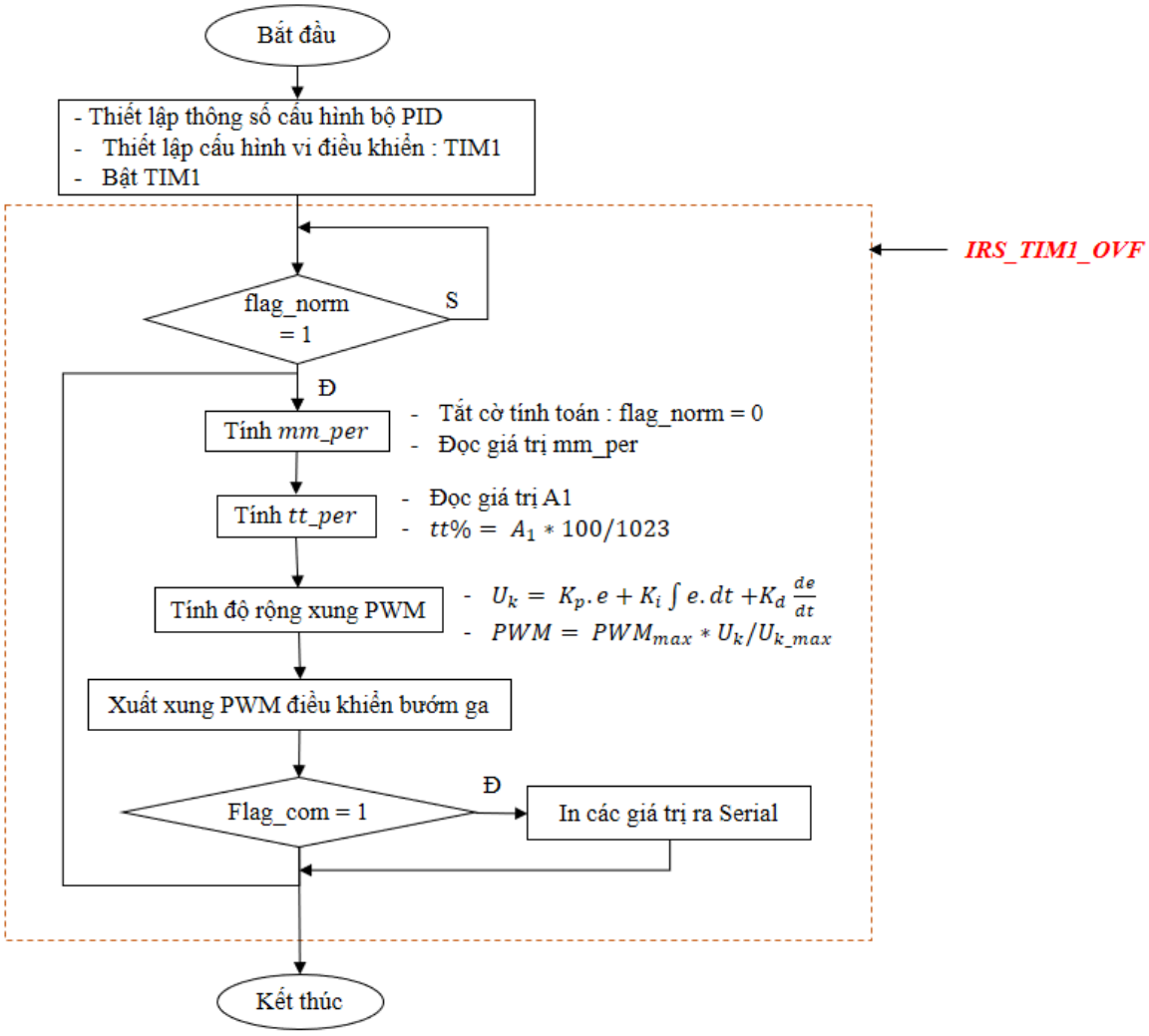
(4) Xuất giá trị PWM ra động cơ bướm ga

Sau một số lượng chu kì tính toán sẽ thực hiện 1 chu kì giao tiếp. Trong một chu kì giao tiếp sau khi thực hiện các công việc từ (1) đến (4) sẽ thực hiện thêm công việc (5):

(5) In các giá trị setpoint, input, ouput ra cổng Serial.

Để dễ quan sát, chọn tần số của một chu kì giao tiếp là 20Hz tương ứng với chu kì 50ms. Chọn với 5 chu kì tính toán sẽ thực hiện 1 chu kì giao tiếp. Khi đó chu kì tính toán sẽ là 10ms.

3. Lưu đồ giải thuật



Hình 4.3. Lưu đồ giải thuật chương trình chính

3.1. Bắt đầu chương trình :

- Khai báo các biến :

  + Biến liên quan để đọc các giá trị đầu vào

  + Biến liên quan để tính toán giá trị xung điều khiển

  + Biến cấu hình bộ tính toán PID

- Thiết lập clock và cấp clock cho timer TIM1

- Thiết lập ngõ vào và ngõ ra của vi điều khiển :

+ Chân Input A0 và A1 đọc giá trị từ cảm biến

+ Chân Output digital số 9 (OCRA1) để điều khiển xung PWM

- Bật TIM1

3.2. Tính độ mở bướm ga mong muốn :

- Đọc giá trị Ax từ Labview. Đây là biến trung gian để tính mm%, có giá trị từ từ 0 đến 1023.

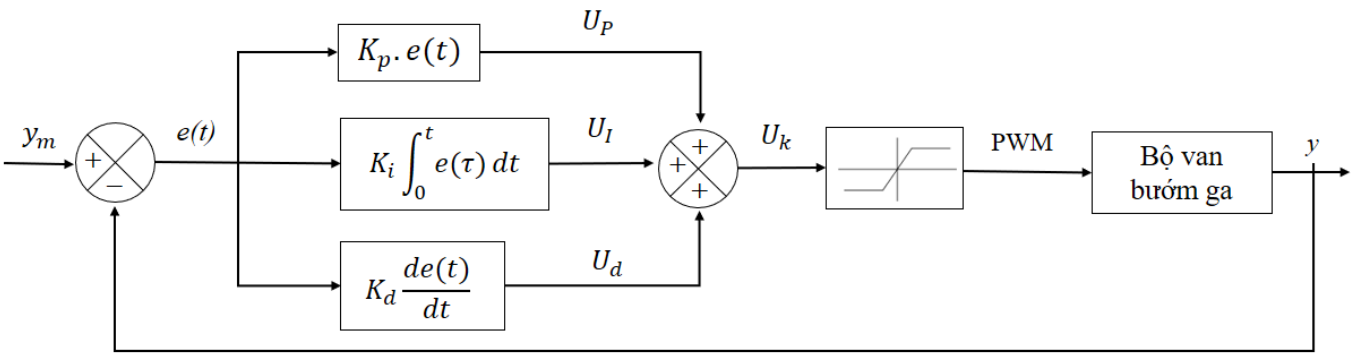
-Tính $mm\% = A_x * 100/1023$. Đây là giá trị độ mở bướm ga mong muốn, có giá trị từ 0 đến 100 (%).

3.2. Tính độ mở bướm ga thực tế :

- Đọc giá trị A1. Đây là giá trị VTA1 dùng để tính tt%.

- Tính $tt\% = A_1 * 100/1023$. Đây là giá trị độ mở bướm ga thực tế, có giá trị từ 0 đến 100 (%).

3.3. Tính độ rộng xung PWM



Hình 4.4 Sơ đồ thuật toán PID của bộ điều khiển

- Tính giá trị điều khiển U_k bằng thuật toán PID :

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

với :

- K_p , K_i , K_d là các hệ số PID được nêu ở phần cơ sở lý thuyết

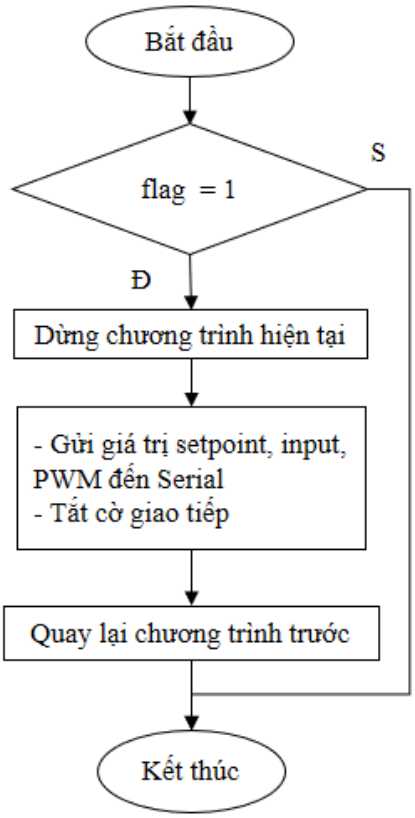
- $e = mm\% - tt\%$: là sai số giữa độ mở bướm ga mong muốn và thực tế

- Tính giá trị độ rộng xung PWM : $PWM = PWM_{max} * U_k/U_{k\_max}$

3.4. Sự kiện in giá trị ra Serial

Sự kiện được thực hiện khi cờ giao tiếp flag_com = 1. Sự kiện sẽ thực hiện các thao tác:

- In giá trị mm% ra Serial
- In giá trị tt% ra Serial
- In giá trị PWM ra Serial
- Tắt cờ giao tiếp : flag_com = 0

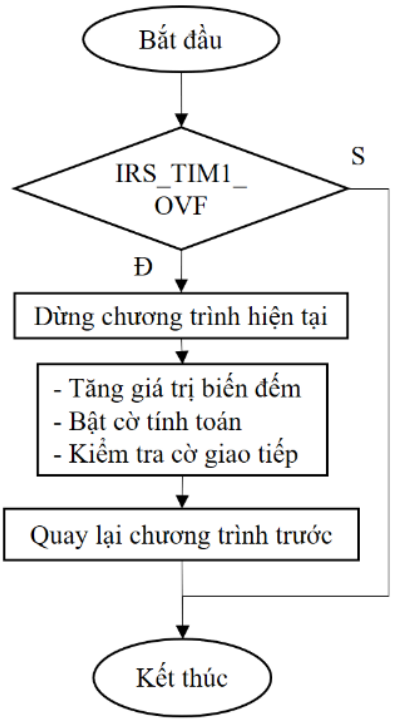


Hình 4.4. Sự kiện chu kì in Serial.

3.5. Sự kiện chu kì đếm 10ms :

Ở mỗi chu kì đếm 10ms tạo ra bởi timer TIM1, các công việc sau sẽ được thực hiện :

- Tăng giá trị biến đếm : count += 1
- Bật cờ tính toán : flag_norm = 1
- Kiểm tra chu kì giao tiếp gửi, nếu giá trị biến đếm bằng 5 : flag_com = 1



Hình 4.5. Sự kiện chu kì tính toán

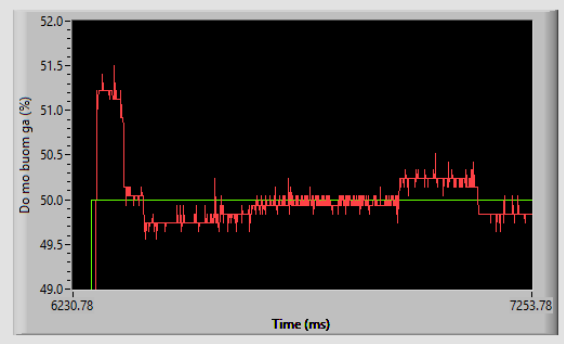
Chương V. KẾT QUẢ & ĐÁNH GIÁ

1. Xác định các thông số của cảm biến vị trí bướm ga

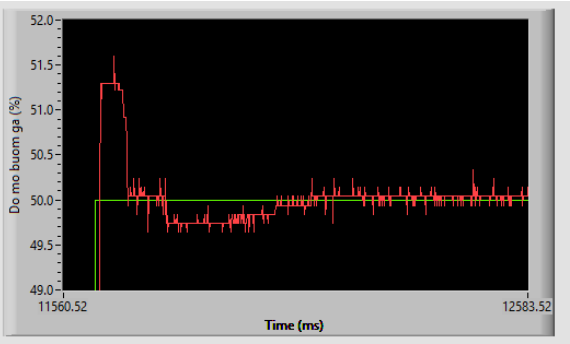
- Khi bướm ga đóng hoàn toàn : điện áp đầu ra cảm biến cực tiểu = 0.8V
- Khi bướm ga mở hoàn toàn : điện áp đầu ra cảm biến cực đại = 3.98V

2. Khảo sát ảnh hưởng của các hệ số Kp, Ki, Kd đến đáp ứng điều khiển

2.1 Ảnh hưởng của Kp đến quá trình điều khiển :

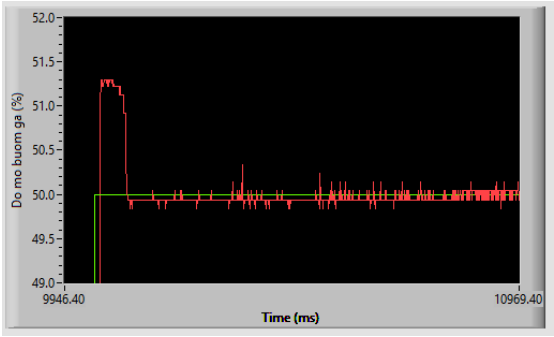


a) Kp = 2

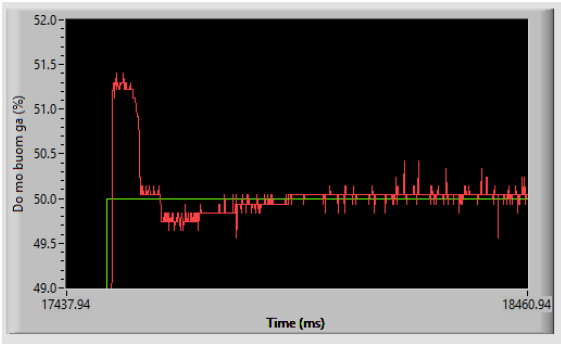


b) Kp = 3

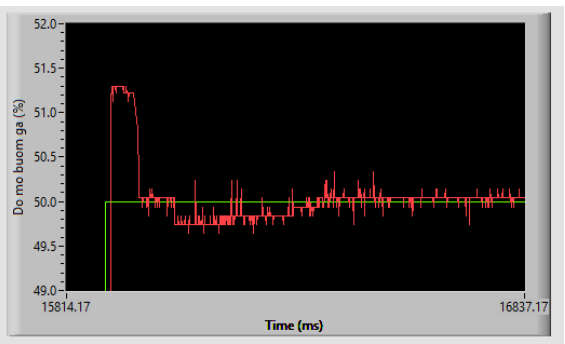
Nếu Hệ số của khâu tỉ lệ quá thấp, bộ điều khiển sẽ kém nhạy hoặc đáp ứng chậm.



c) Kp = 6 :



d) Kp=10



e) Kp = 12:

Nếu hệ số của khâu tỉ lệ **quá cao**, hệ thống sẽ **không ổn định**.

Hình 5.1. Đáp ứng hệ thống với các hệ số Kp

Nhận xét:

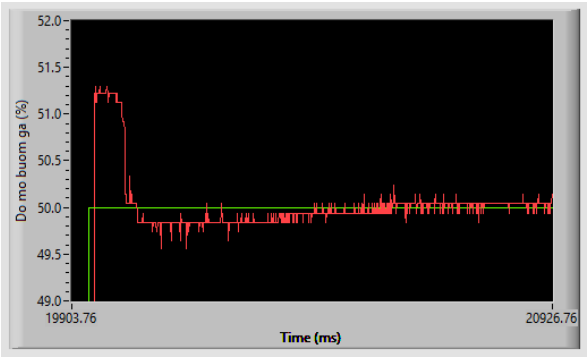
Khâu tỉ lệ (đôi khi còn được gọi là *độ lợi*) làm thay đổi giá trị đầu ra, tỉ lệ với giá trị sai số hiện tại. Đáp ứng tỉ lệ có thể được điều chỉnh bằng cách nhân sai số đó với một hằng số K_p, được gọi là hệ số tỉ lệ.

Hệ số của khâu tỉ lệ lớn là do thay đổi lớn ở đầu ra mà sai số thay đổi nhỏ. Nếu hệ số của khâu tỉ lệ **quá cao**, hệ thống sẽ **không ổn định**.

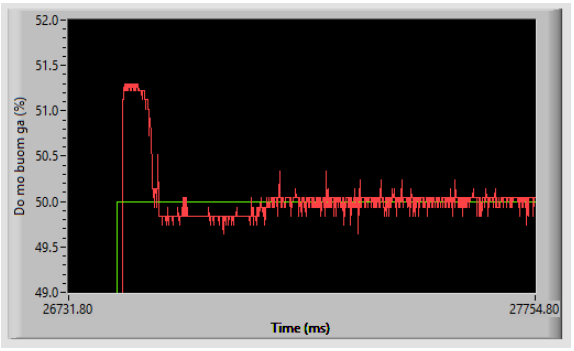
Ngược lại, hệ số nhỏ là do đáp ứng đầu ra nhỏ trong khi sai số đầu vào lớn, và làm cho bộ điều khiển kém nhạy, hoặc đáp ứng chậm. Nếu Hệ số của khâu tỉ lệ quá thấp, tác động điều khiển có thể sẽ quá bé khi đáp ứng với các nhiễu của hệ thống.

2.2. Ảnh hưởng của Ki đến quá trình điều khiển :

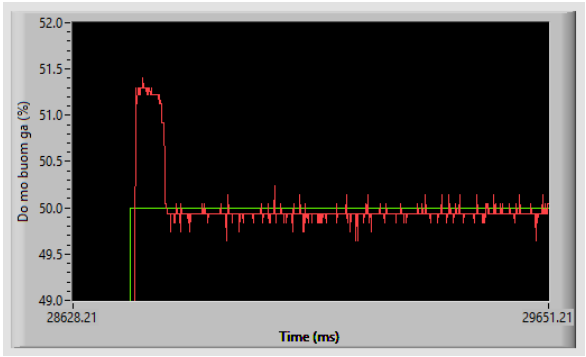
Phân phối của khâu tích phân (đôi khi còn gọi là *reset*) tỉ lệ thuận với cả biên độ sai số lẫn quảng thời gian xảy ra sai số. Tổng sai số tức thời theo thời gian (tích phân sai số) cho ta tích lũy bù đã được hiệu chỉnh trước đó. Tích lũy sai số sau đó được nhân với độ lợi tích phân và cộng với tín hiệu đầu ra của bộ điều khiển. Biên độ phân phối của khâu tích phân trên tất cả tác động điều chỉnh được xác định bởi độ lợi tích phân, Ki.



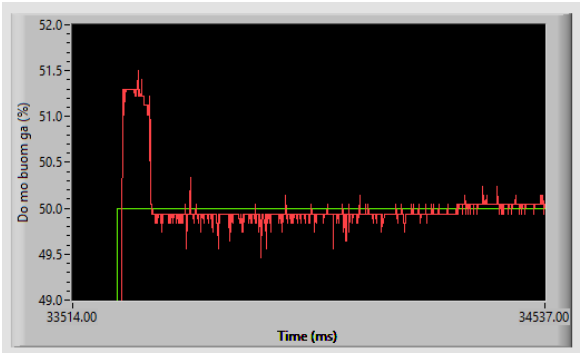
Ki = 1



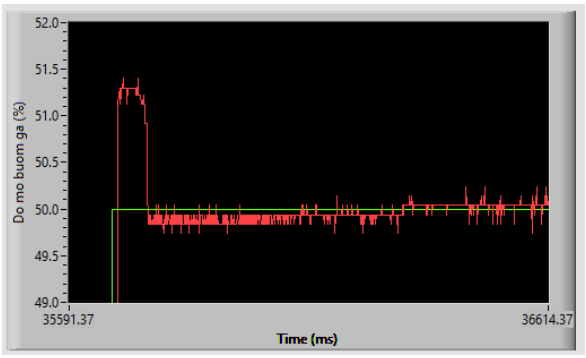
Ki = 4,5 :



Ki = 6,5



Ki = 8,5



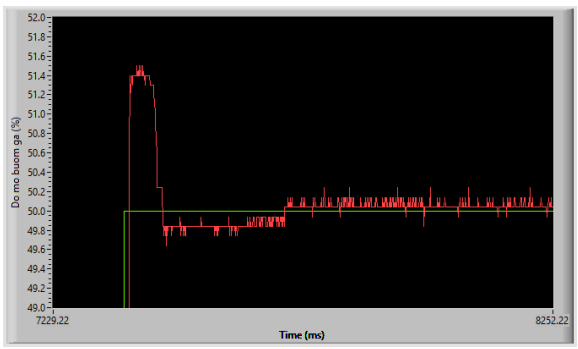
Ki = 10:

Hình 5.2 Đáp ứng của hệ thống với các hệ số Ki

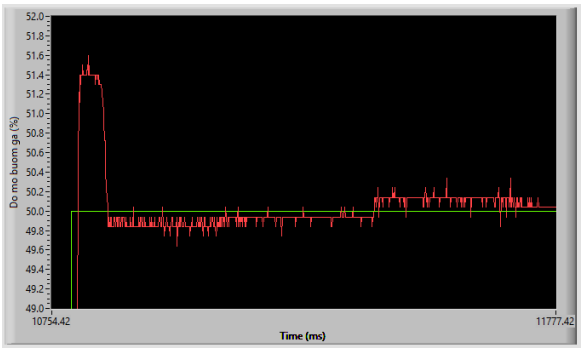
Nhận xét:

Khâu tích phân (khi cộng thêm khâu tỉ lệ) sẽ tăng tốc chuyển động của quá trình tới điểm đặt và khử số dư sai số ổn định với một tỉ lệ chỉ phụ thuộc vào bộ điều khiển. Tuy nhiên, vì khâu tích phân là đáp ứng của sai số tích lũy trong quá khứ, nó có thể khiến giá trị hiện tại vọt lố qua giá trị đặt (ngang qua điểm đặt và tạo ra một độ lệch với các hướng khác).

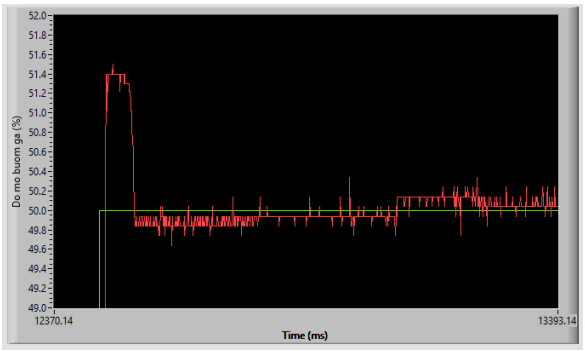
2.3. Ảnh hưởng của Kd đến quá trình :



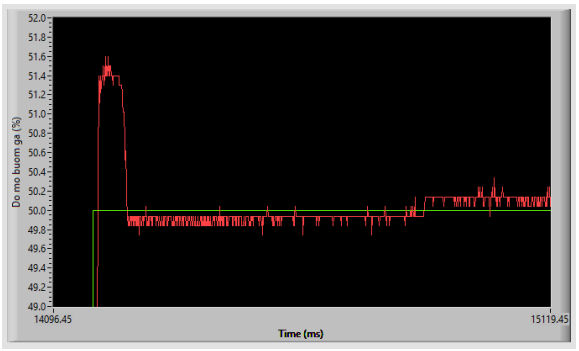
Kd = 0,2



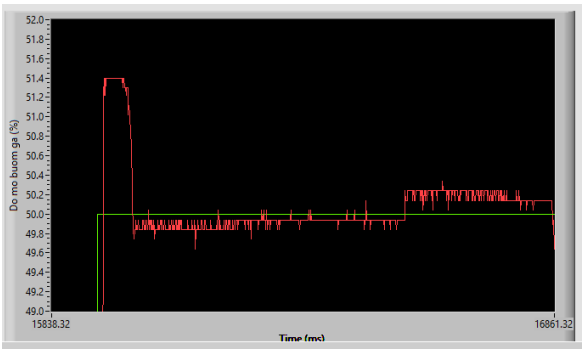
Kd = 0,5



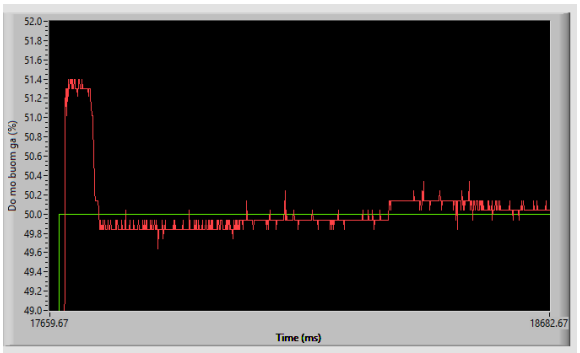
Kd = 0,55



Kd = 0,6 :

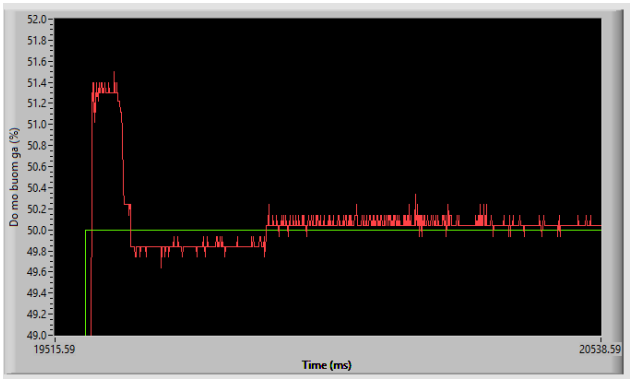


Kd = 0,6



Kd = 0,7

Kd = 1 :



Hình 5.3. Đáp ứng của hệ thống với các hệ số Kd

Nhận xét

Tốc độ thay đổi của sai số của 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 tốc độ này với độ lợi tỉ lệ Kd. Biên độ của phân phối khâu vi phân (đôi khi được gọi là *tốc độ*) trên tất cả các hành vi điều khiển được giới hạn bởi độ lợi vi phân, Kd.

Khâu vi phân làm chậm tốc độ thay đổi của đầu ra bộ điều khiển và đặc tính này là đáng chú ý nhất để đạt tới điểm đặt của bộ điều khiển. Từ đó, điều khiển vi phân được sử dụng để làm **giảm biên độ vọt lố được tạo ra bởi thành phần tích phân** và tăng cường độ ổn định của bộ điều khiển hỗn hợp. Tuy nhiên, **phép vi phân của một tín hiệu sẽ khuếch đại nhiễu** và do đó khâu này sẽ nhạy hơn đối với nhiễu trong sai số, và có thể khiến quá trình trở nên không ổn định nếu nhiễu và độ lợi vi phân đủ lớn. Do đó một xấp xỉ của bộ vi sai với băng thông giới hạn thường được sử dụng hơn. Chẳng hạn như mạch bù sớm pha.

Tóm tắt :

Khâu tỉ lệ, tích phân, vi phân được cộng lại với nhau để tính toán đầu ra của bộ điều khiển PID. Định nghĩa rằng u(t) là đầu ra của bộ điều khiển, biểu thức cuối cùng của giải thuật PID là:

trong đó các thông số điều chỉnh là:

- Độ lợi tỉ lệ, Kp : giá trị càng lớn thì đáp ứng càng nhanh do đó sai số càng lớn, bù khâu tỉ lệ càng lớn. Một giá trị độ lợi tỉ lệ quá lớn sẽ dẫn đến quá trình mất ổn định và dao động.

- Độ lợi tích phân, Ki : giá trị càng lớn kéo theo sai số ổn định bị khử càng nhanh. Đổi lại là độ vọt lố càng lớn: bất kỳ sai số âm nào được tích phân trong suốt đáp ứng quá độ phải được triệt tiêu tích phân bằng sai số dương trước khi tiến tới trạng thái ổn định.

- Độ lợi vi phân, Kd : giá trị càng lớn càng giảm độ vọt lố, nhưng lại làm chậm đáp ứng quá độ và có thể dẫn đến mất ổn định do khuếch đại nhiễu tín hiệu trong phép vi phân sai số.

* Điều chỉnh vòng lặp :

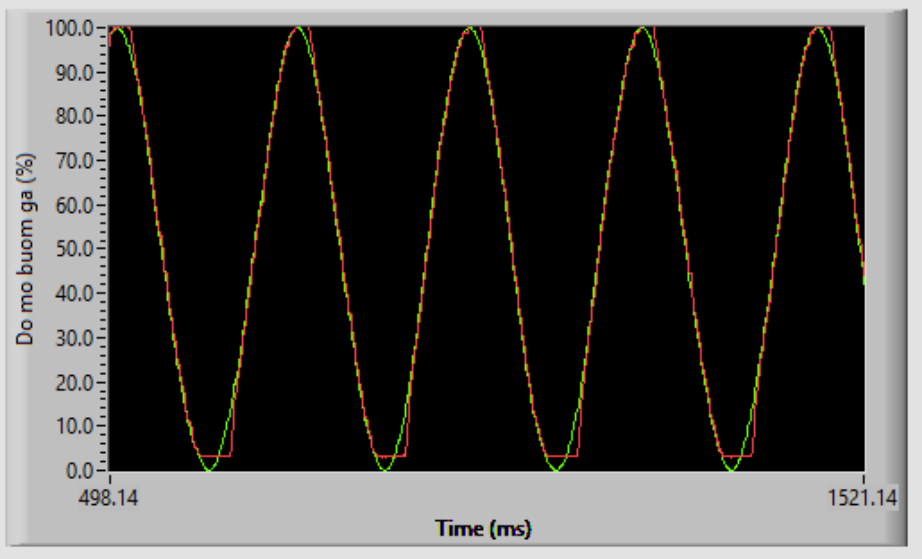
Điều chỉnh một vòng điều khiển là điều chỉnh các thông số điều khiển của nó (độ lợi/dải tỉ lệ, độ lợi tích phân/reset, độ lợi vi phân/tốc độ) tới giá trị đáp ứng điều khiển tối ưu. Độ ổn định (dao động biên) là một yêu cầu căn bản, nhưng ngoài ra, các hệ thống khác nhau, có những hành vi khác nhau, những ứng dụng khác nhau có những yêu cầu khác nhau, và vài yêu cầu lại mâu thuẫn với nhau. Hơn nữa, vài quá trình có một mức độ phi tuyến nào đấy khiến các thông số làm việc tốt ở điều kiện đầy tải sẽ không làm việc khi quá trình khởi động từ không tải; điều này có thể khắc phục bằng chương trình độ lợi (sử dụng các thông số khác nhau cho những khu vực hoạt động khác nhau). Các bộ điều khiển PID thường cung cấp các điều khiển có thể chấp nhận được thậm chí không cần điều chỉnh, nhưng kết quả nói chung có thể được cải thiện bằng cách điều chỉnh kỹ lưỡng, và kết quả có thể không chấp nhận được nếu điều chỉnh kém.

Điều chỉnh PID là một bài toán khó, ngay cả khi chỉ có 3 thông số và về nguyên tắc là dễ miêu tả, bởi vì nó phải thỏa mãn các tiêu chuẩn phức tạp nằm trong Những hạn chế của điều khiển PID [3]. Vì vậy có nhiều phương pháp khác nhau để điều chỉnh vòng lặp, và các kỹ thuật phức tạp hơn là đề tài cho nhiều phát minh sáng chế; phần này miêu tả vài phương pháp thủ công truyền thống để điều chỉnh vòng lặp.

3. Một số trường hợp làm việc của bộ điều khiển điện tử :

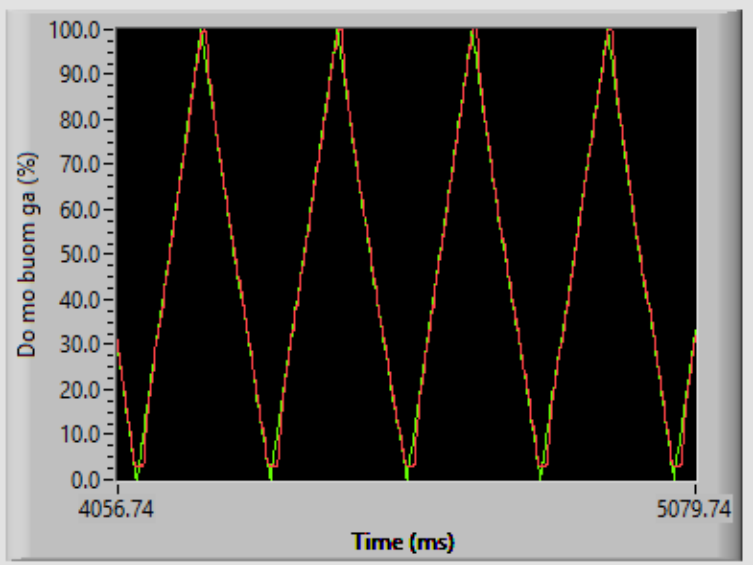
Sau khi chọn một bộ thông số điều khiển PID tương đối, ta tiến hành xét khả năng đáp ứng của hệ thống điều khiển điện tử đã thiết lập đối với một số trường hợp độ mở bướm ga mong muốn.

3.1. Đáp ứng xung hình sine : Tần số xung : 20Hz



Hình 5.4. Đáp ứng hệ thống với sung hình sine

3.2. Đáp ứng xung tam giác :



Hình 5.5. Đáp ứng xung tam giác của bộ điều khiển

Chương VI. KẾT LUẬN

1. Đánh giá tiến độ luận văn :

Sau thời gian thực hiện, luận văn đã đạt đúng tiến độ và giải quyết được các bài toán đặt ra.

Qua quá trình thực nghiệm, hoạt động của bộ điều khiển đã đáp ứng các tiêu chí :

- Điều chỉnh được độ mở bướm ga mong muốn
- Tính toán được giá trị điện áp điều khiển bướm ga
- Điều khiển được bướm ga theo giá trị điện áp tính toán
- Hiển thị và theo dõi được các giá trị độ mở bướm ga mong muốn, độ mở bướm ga thực tế và độ rộng xung PWM ở dạng đồ thị và dạng số.
- Sai số

Tuy nhiên qua thực nghiệm, hệ thống còn một số điểm hạn chế :

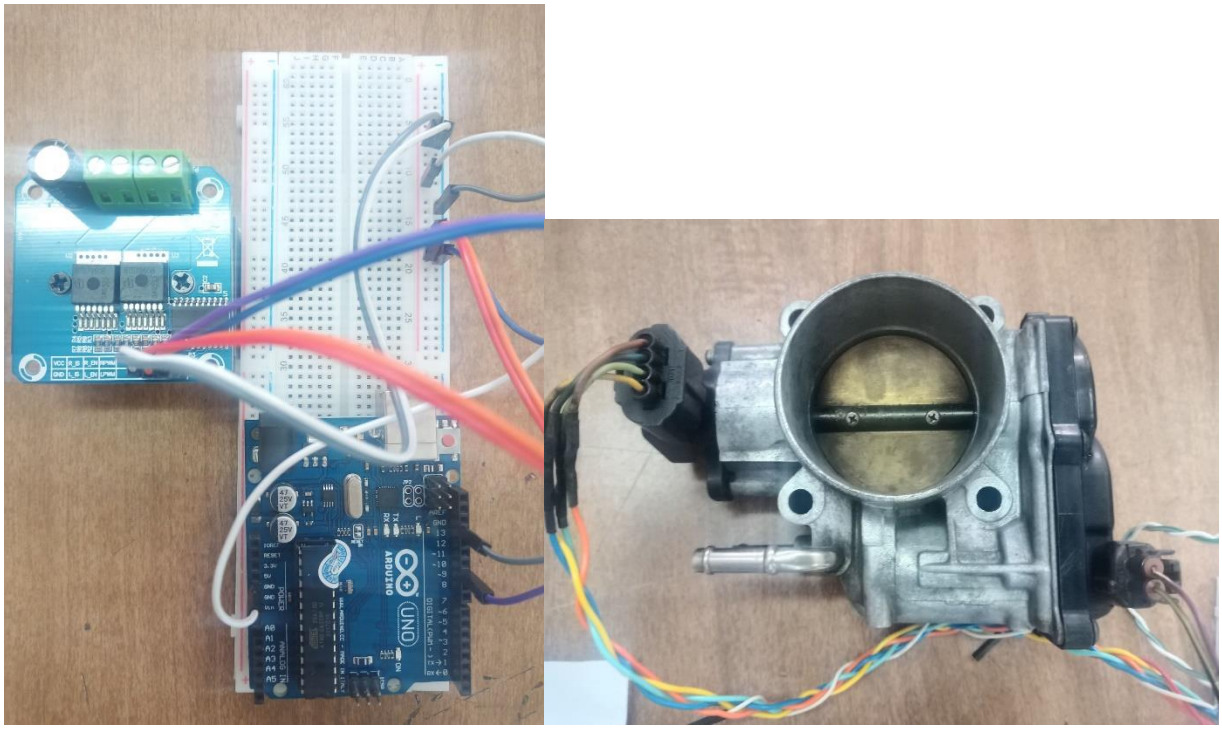
- Bộ tính toán PID chưa tối ưu

- Đáp ứng xung chưa tốt

- Còn độ trễ khi hiển thị các giá trị trên Labview so với thực tế

- Chỉ điều khiển dựa trên 1 tham số (độ mở bướm ga mong muốn), chưa liên kết với bàn đạp chân ga.

2. Tóm tắt các đặc trưng kỹ thuật của thiết kế/sản phẩm và kết quả đạt được



Hình 6.1. Các thành phần chính của bộ điều khiển

Hệ thống điện điện tử được thiết kế với các đặc tính kĩ thuật như sau :

- Độ mở bướm ga mong muốn điều khiển được từ 0 đến 100%
- Chu kì điều khiển 10ms (100Hz)
- Chu kì hiển thị kết quả 50ms (20Hz)
- Điện áp điều khiển cực đại : 12V

3. Hướng phát triển

3.1. Các vấn đề cần khắc phục:

Để hoàn thiện hơn hệ thống điều khiển, các vấn đề cần giải quyết để phát triển đồ án trong tương lai :

- Tối ưu hóa bộ tính toán PID để hệ thống đáp ứng tốt hơn
- Tối ưu hóa quá trình giao tiếp
- Khử nhiễu khi đọc cảm biến để quá trình tính toán chính xác hơn.

3.2. Tính ứng dụng và phát triển đề tài

3.2.1. Tính ứng dụng :

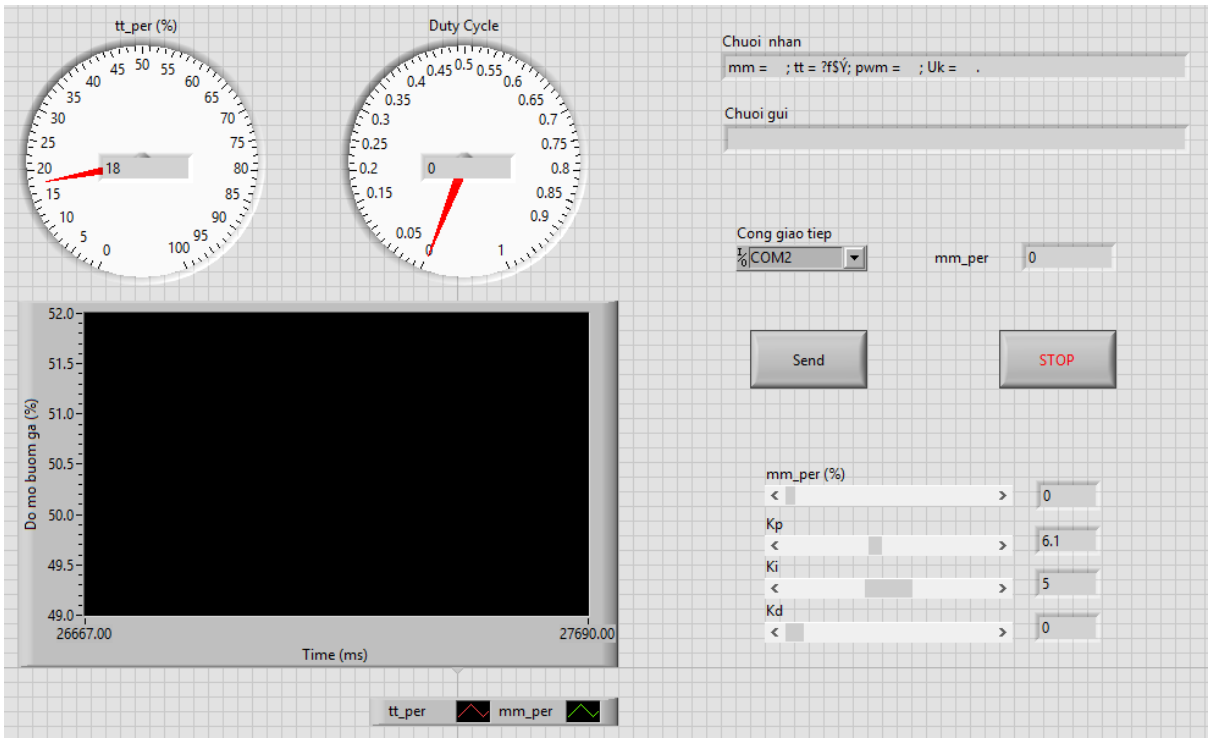
- Có thể áp dụng trên các loại ô tô, xe máy sử dụng bướm ga điện tử

3.2.2. Mở rộng đề tài :

Mục tiêu của đề là tìm hiểu việc điều khiển bướm ga theo một độ mở mong muốn. Để phát triển, phải bổ sung các tham số điều khiển như mức độ nhấn ga ở bàn đạp chân ga, tốc độ động cơ, áp suất động cơ, cảm biến oxy, … nâng bộ điều khiển từ đơn biến trở thành đa biến.

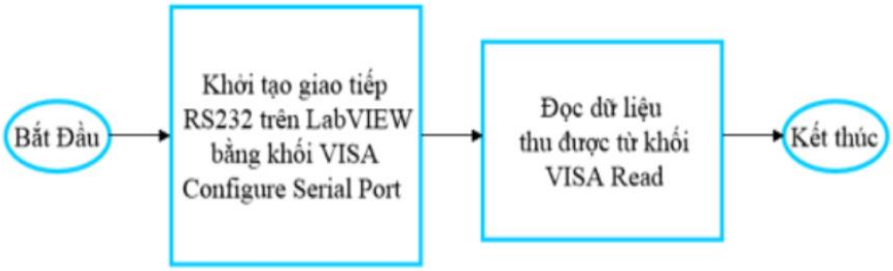
Cũng từ đó, bộ điều khiển có thể phân biệt và điều khiển độ mở bướm ga thích hợp hơn dựa trên các chế độ làm việc (cầm chừng, nửa tải, toàn tải, tăng tốc đột ngột, khởi động).

Phụ lục 1: Giao diện điều khiển sử dụng phần mềm Labview



Hình 7.1. Giao diện điều khiển trên Labview

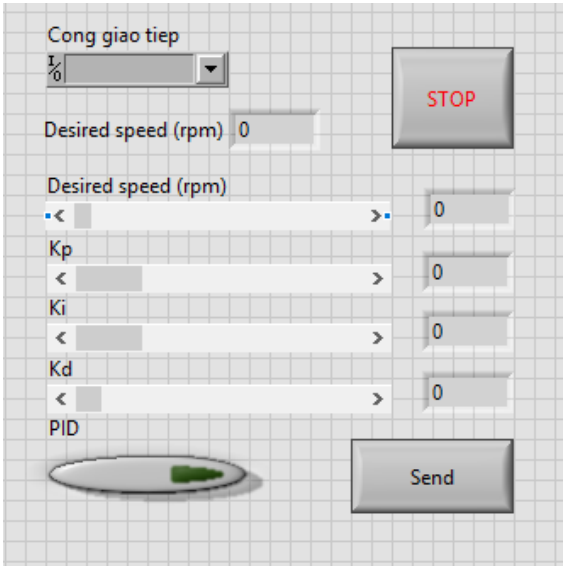
Để thực hiện gửi/nhận tín hiệu giữa Labview và Arduino, ta thiết lập giao tiếp RS232 trên Labview.



Hình 7.2. Sơ đồ khối giao tiếp giữa Labview và Arduino

Sau khi thiết lập giao tiếp giữa Labview và Arduino, ta tiến hành thiết lập các khối lệnh thực hiện các công việc gửi/nhận tín hiệu qua Serial.

1. Nhiệm vụ điều khiển (gửi):



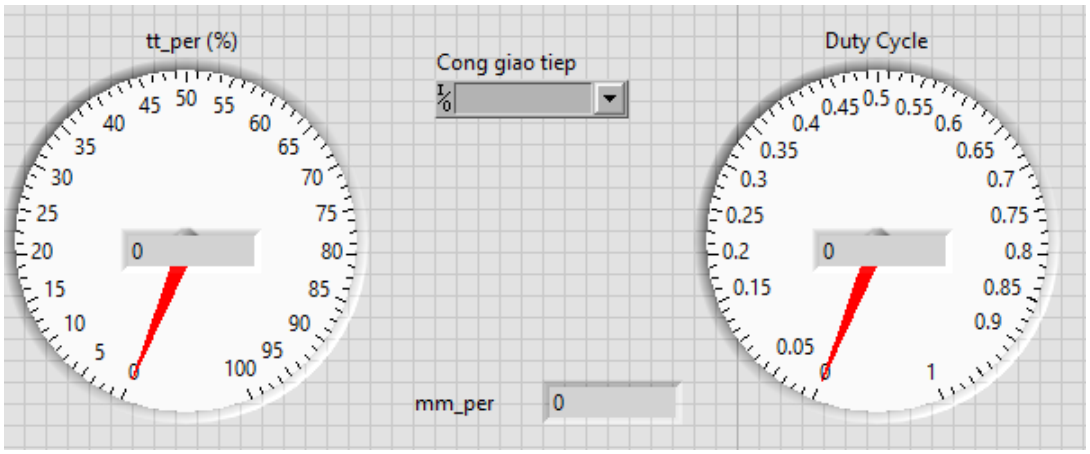
Hình 7.3. Nhóm các khối điều khiển trên giao diện Labview

Độ mở bướm ga mong muốn sẽ được điều khiển bằng một núm xoay

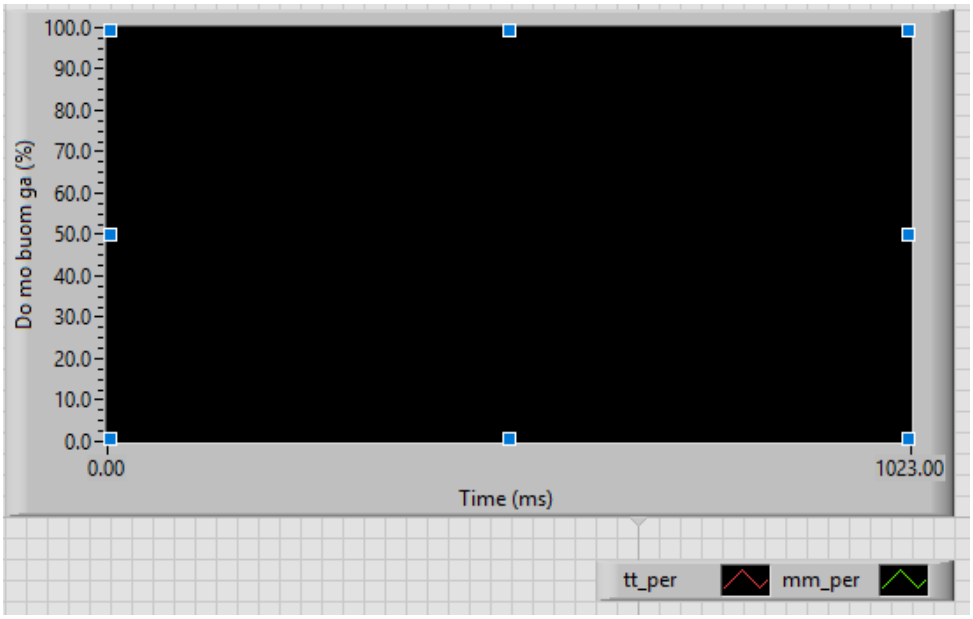
- Giá trị trên núm xoay : sẽ có giá trị từ 0 đến 1023 (10 bit)

- Giá trị trên núm xoay sẽ được chuyển thành chuỗi và xuất ra ở cổng Serial và được đọc bởi Arduino

2. Nhiệm vụ hiển thị (đọc):



a) Khối hiển thị dạng số



b) Khối hiển thị dạng đồ thị

Hình 7.4. Nhóm các khối hiển thị trên giao diện điều khiển

Kết luận :

Với giao diện Labview được thiết kế trên, ta đã có thể thực hiện các công việc

- Điều chỉnh độ mở bướm ga mong muốn (qua núm xoay)
- Hiển thị được các giá trị đọc được từ Serial để theo dõi
- Vẽ được đồ thị từ các giá trị đó

Phụ lục : TIMER/COUNTER TRONG ARDUINO

Timer/Counter là module hoạt động độc lập và không thể thiếu của bất kỳ Microcontroller nào. Chức năng của Timer/Counter gồm: định thời, đếm sự kiện, tạo xung PWM,....

Trên chip Atmega328p của Arduino có 3 bộ Timer/Counter là: **Timer/Counter0** (8bit), **Timer/Counter1** (16 bit), **Timer/Counter2** (8 bit).

- Timer/Counter1: là 1 bộ Timer/Counter đa năng 16 bit, gồm 5 chế độ hoạt động.
- Timer/Counter2: là 1 bộ Timer/Counter 8 bit, gồm 4 chế độ.

Một số định nghĩa quan trong chúng ta cần rõ:

- **BOTTOM**: là giá trị thấp nhất mà 1 T/C đạt được, tất nhiên BOTTOM luôn bằng 0.
- **MAX**: là giá trị lớn nhất mà 1 T/C có thể đạt được, ở thanh ghi 8 bit giá trị MAX = 2^8 -1 = 255, ở thanh ghi 16 bit giá trị MAX = 2^16 - 1 = 65535. Và tất nhiên giá trị MAX cũng là cố định với từng T/C.
- **TOP**: là giá trị đỉnh mà tại có T/C thay đổi trạng thái, giá trị TOP không nhất thiết phải bằng MAX mà có thể thay đổi bằng các thanh ghi.
- **Interrupt**: (còn gọi là Ngắt) là 1 chương trình có độ ưu tiên cao nhất, được thực hiện ngay lập tức khi có tín hiệu Interrupt.

8	0x000E	TIMER2 COMPA	Timer/Counter2 Compare Match A
9	0x0010	TIMER2 COMPB	Timer/Counter2 Compare Match B
10	0x0012	TIMER2 OVF	Timer/Counter2 Overflow
11	0x0014	TIMER1 CAPT	Timer/Counter1 Capture Event
12	0x0016	TIMER1 COMPA	Timer/Counter1 Compare Match A
13	0x0018	TIMER1 COMPB	Timer/Coutner1 Compare Match B
14	0x001A	TIMER1 OVF	Timer/Counter1 Overflow
15	0x001C	TIMER0 COMPA	Timer/Counter0 Compare Match A
16	0x001E	TIMER0 COMPB	Timer/Counter0 Compare Match B
17	0x0020	TIMER0 OVF	Timer/Counter0 Overflow

Bảng 8.1: Intterupt Vector của Timer/Counter trên Atmega328

1. Timer/Counter 1

1.2 Giới thiệu các thanh ghi

1.1.1 Thanh ghi TCNT1 (Timer/Counter 1 Register)

Là thanh ghi 16 bit, lưu giữ giá trị của Timer/Counter1, cho phép đọc-ghi trực tiếp, do đó, chúng ta có thể thực hiện các phép gán hoặc thay đổi giá trị của **TCNT1**.

1.1.2 Thanh ghi TCCR1B (Timer/Counter 1 Control Register B)

Là 1 trong 2 thanh ghi điều khiển hoạt đông của Timer/Counter1 (cùng với **TCCR1A**, nhưng với những mục đích đơn giản, chúng ta chỉ cần thanh ghi **TCCR1B**).

7	6	5	4	3	2	1	0	
ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
0	0	0	0	0	0	0	0	

Bảng 8.2. Thanh ghi TCCR1B

Trong thanh ghi **TCCR1B** chúng ta chỉ cần sử dụng 3 bit **CS10**, **CS11**, **CS12** để lựa chọn xung nhịp cho T/C1. Chúng ta sẽ tham khảo bảng này:

Table 16-5. Clock Select Bit Description

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/1 (No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

Bảng 8.3. Mô tả Clock Select Bit trên thanh ghi TCCR1B

Theo mặc định, chip Atmega328p trên Arduino chạy ở 16MHz, prescaler = 64. Điều này có nghĩa là: theo mặc định, các bộ T/C trên Arduino sẽ có tần số hoạt động là 16MHz/64 = 250kHz.

1.1.3 Thanh ghi TIMSK1 (Timer/Counter1 Interrupt Mask Register)

Là thanh ghi lưu giữ các Interrupt Mask của T/C1. Đây là thanh ghi giúp chúng ta thực hiện các Timer Interrupt. Trên thanh ghi TIMSK1 chúng ta cần chú ý các bit sau:

7	6	5	4	3	2	1	0	
–	–	ICIE1	–	–	OCIE1B	OCIE1A	TOIE1	TIMSK1
R	R	R/W	R	R	R/W	R/W	R/W	
0	0	0	0	0	0	0	0	

Bảng 8.4: Thanh ghi TIMSK1 (Timer/Counter1)

- bit 5 - **ICIE1**: Input Capture Interrupt Enable - Cho phép ngắt khi dùng Input Capture.
- bit 2 - **OCIE1B**: Output Compare Interrupt Enable 1 channel B - Cho phép ngắt khi dùng Output Compare ở channel B.
- bit 1 - **OCIE1A**: Output Compare Interrupt Enable 1 channel A - Cho phép ngắt khi dùng Output Compare ở channel A.
- bit 0 - **TOIE1**: Overflow Interrupt Enable 1 - Cho phép ngắt khi xảy ra tràn trên T/C.

1.1.4 Thanh ghi OCR1A và OCR1B (Output Compare Register channel A và channel B)

Lưu giữ giá trị so sánh ở kênh A và kênh B: khi T/C1 hoạt động, giá trị TCNT1 được tăng dần, giá trị này liên tục được so sánh với các giá trị trong thanh ghi OCR1A và OCR1B, việc so sánh này chính là "Output Compare", khi giá trị của TCNT1 bằng giá trị của OCR1A (hoặc OCR1B) thì "Match" xảy ra, lúc này sẽ có 1 Interrupt được thực hiện (nếu đã được Enable ở thanh ghi TIMSK1).

1.1.5 Thanh ghi ICR1 (Input Capture Register 1)

Giá trị của thanh ghi ICR1 sẽ được cập nhật theo thanh ghi TCNT1 mỗi lần có sự kiện xảy ra trên chân ICP1 (tương ứng là chân digital 8 của Arduino). Chức năng này mình sẽ giới thiệu trong 1 bài viết khác.

1.2 Các chế độ của Timer/Counter 1

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1x at	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX

Bảng 8.5. Waveform Generation Mode Bit (Timer/Counter1)

Hai mode cơ bản nhất của T/C1 là: **Normal Mode** và **CTC Mode**.

1.2.1 Normal Mode

Đây là chế độ hoạt động đơn giản nhất của T/C1 (mode 0), giá trị của thanh ghi TCNT1 sẽ tăng từ 0 (BOTTOM) đến 65535 (MAX) và quay về 0. Nếu chúng ta gán trước cho TCNT1 một giá trị nào đó thì TCNT1 sẽ bắt đầu đếm từ giá trị này.

Ví dụ: Ta muốn viết 1 chương trình để đọc dữ liệu từ cảm biến nhiệt mỗi 0.1s, nhưng trong thân chương trình lại có vài hàm delay(), do đó sẽ không đảm bảo là bạn cập nhật được giá trị nhiệt độ mỗi 0.1s nếu chỉ dùng hàm if và hàm millis(). Phương án ở đây là chúng ta sẽ dùng Interrupt của Timer/Counter.

Theo mặc định, chip Atmega328p trên Arduino chạy ở 16MHz, prescaler = 64, vì vậy thời gian để TCNT1 tăng lên 1 đơn vị là 64/16MHz = 4us, thời gian để T/C1 đếm từ 0 đến 65535 là 4us*65536 = 0.262144s, mà thời gian chúng ta cần tạo là 0.1s (thỏa mãn vì 0.1 < 0.262144), do đó ta cần 0.1s/4us = 25000 lần đếm. Giá trị ban đầu của TCNT1 = 65536 - 25000 = 40536.

1.2.2 Clear Timer on Compare Match (CTC) mode

Có 2 CTC mode trên T/C1 là **mode 4** và **mode 12**

Để chọn mode 4, chúng ta cần set các bit như sau: WGM13 = 0, WGM12 = 1, WGM11 = 0, WGM10 = 0.

CTC mode hoạt động như sau: thanh ghi OCR1A lưu giữ giá trị TOP, thanh ghi TCNT1 bắt đầu đếm từ 0, khi giá trị TCNT1 = OCR1A thì "Compare Match", lúc này ngắt Compare Match có thể xảy ra nếu bit OCIE1A đã được set ở thanh ghi TIMSK1.

Để chọn mode 12, chúng ta cần set các bit như sau: WGM13 =1, WGM12 = 1, WGM11 = 0, WGM10 = 0.

2. Timer/Counter 2

2.1 Các thanh ghi

Trên T/C2 cũng có những thanh ghi tương tự T/C1

2.1.1 Thanh ghi TCNT2 (Timer/Counter 2 Register)

Là thanh ghi 8 bit, lưu giữ giá trị của Timer/Counter2.

2.1.2 Thanh ghi TCCR2A và TCCR2B (Timer/Counter 2 Control Register A và B)

Là 2 thanh ghi điều khiển hoạt động của Timer/Counter2.

TCCR2A – Timer/Counter Control Register A

Bit	7	6	5	4	3	2	1	0	
(0xB0)	COM2A1	COM2A0	COM2B1	COM2B0	–	–	WGM21	WGM20	TCCR2A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TCCR2B – Timer/Counter Control Register B

Bit	7	6	5	4	3	2	1	0	
(0xB1)	FOC2A	FOC2B	–	–	WGM22	CS22	CS21	CS20	TCCR2B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bảng 8.6: Thanh ghi TCCR2A và TCCR2B (Timer/Counter 2)

CS22	CS21	CS20	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	clk_{T2S}/(No prescaling)
0	1	0	clk_{T2S}/8 (From prescaler)
0	1	1	clk_{T2S}/32 (From prescaler)
1	0	0	clk_{T2S}/64 (From prescaler)
1	0	1	clk_{T2S}/128 (From prescaler)
1	1	0	clk_{T2S}/256 (From prescaler)
1	1	1	clk_{T2S}/1024 (From prescaler)

Bảng 8.7: Mô tả Clock Select Bit

2.1.3 Thanh ghi TIMSK2 (Timer/Counter2 Interrupt Mask Register)

Là thanh ghi lưu giữ các Interrupt Mask của T/C2.

TIMSK2 – Timer/Counter2 Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
(0x70)	–	–	–	–	–	OCIE2B	OCIE2A	TOIE2	TIMSK2
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bảng 8.8: Thanh ghi TIMSK2 (Timer/Counter 2)

2.1.4 Thanh ghi OCR2A và OCR2B (Output Compare Register channel A và channel B)

Lưu giữ giá trị so sánh ở kênh A và kênh B khi T/C2 hoạt động.

OCR2A – Output Compare Register A

Bit	7	6	5	4	3	2	1	0	
(0xB3)	OCR2A[7:0]								OCR2A
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

OCR2B – Output Compare Register B

Bit	7	6	5	4	3	2	1	0	
(0xB4)	OCR2B[7:0]								OCR2B
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bảng 8.9: Lưu giữ giá trị so sánh ở kênh A và kênh B khi T/C2 hoạt động.

2.2 Các chế độ hoạt động

Theo bảng Waveform Generation Mode bit, T/C2 có **Normal Mode 0** và **CTC mode 2**.

Để không trùng lặp nội dụng với T/C1, mình chỉ giới thiệu cách set thanh ghi trong T/C2.

- **Ở Normal Mode**, các bạn chỉ cần set các bit CS20, CS21, CS22 trong thanh ghi TCCR2B để chọn prescaler.
- **Ở CTC Mode**: ngoài set các bit trong thanh ghi TCCR2B để chọn prescaler, các bạn cần set bit WGM21 lên 1 bằng dòng: TCCR2A |= (1 << WGM21);
- Cách set thanh ghi TIMSK2 tương tự như TIMSK1: OCIE2A (Output Compare Interrupt Enable 2 Channel A), OCIE2B, TOIE2 (Timer Overflow Interrupt 2 Enable).

Phụ lục : CHƯƠNG TRÌNH ĐIỀU KHIỂN ARDUINO

1. Code :

```
#include <avr/interrupt.h>
#include <PID_v1.h>
#include <string.h>
#define motor 9
bool flag_com = 0, flag_norm = 0;
char chuoi_giaotiep;
unsigned int count_norm = 0, count_com=0, tmp =999;
double voltage0, voltage1, temp;
double pwm, mm_per = 0, tt_per, Uk=0;
double Kp = 0.6, Ki = 3, Kd = 0.001;
PID myPID(&tt_per, &Uk, &mm_per, Kp, Ki, Kd, DIRECT);
byte* p_Kp = (byte*)&Kp;
byte* p_Ki = (byte*)&Ki;
byte* p_Kd = (byte*)&Kd;
byte* p_ne_req = (byte*)&mm_per;
uint8_t start_pid_count = 48;
char read_buffer[6];

void send_num_float(double num_float)
{
  byte* p_num_float = (byte*)&num_float;
  Serial.print((char)*(p_num_float + 3));
  Serial.print((char)*(p_num_float + 2));
  Serial.print((char)*(p_num_float + 1));
  Serial.print((char)*p_num_float);
}
```

```
void setup()
{
  Serial.begin(9600);

  pinMode(motor, OUTPUT);
  pinMode(A0, INPUT);

  myPID.SetMode(AUTOMATIC);
  myPID.SetTunings(Kp, Ki, Kd);
  myPID.SetOutputLimits(0, 3.18);
  myPID.SetSampleTime(10);

  cli();
  TCCR1A = 0;   TCCR1B = 0;   TIMSK1 = 0;
  TCCR1B |= (1 << WGM13) | (1 << WGM12);    // Mode 12 : TOP = ICR1
, đếm từ 0 --> ICR1.
  TCCR1B |= (1 << CS10);      // prescale = 1
  TCCR1A |= (1 << WGM11) | (1 << COM1A1);   //

  ICR1 = tmp;    // F = 20KHz
  OCR1A = 0;

  TIMSK1 = (1 << TOIE1);  // ngắt tràn
  sei();
}
```

```
void loop()
{
  if (Serial.available() >= 8)
  {
    switch (Serial.read())
    {
      case 'n':
      {
        if (Serial.read() == 'e')
          Serial.readBytes(read_buffer, 5);
        if (Serial.read() == ';')
        {
          *p_ne_req = (byte) read_buffer[3];
          *(p_ne_req + 1) = (byte) read_buffer[2];
          *(p_ne_req + 2) = (byte) read_buffer[1];
          *(p_ne_req + 3) = (byte) read_buffer[0];
          start_pid_count = (uint8_t)(read_buffer[4]);
        }
        break;
      }
      case 'k':
      {
        switch (Serial.read())
        {
          case 'p':
          {
            Serial.readBytes(read_buffer, 5);
            if (Serial.read() == ';')
            {
```

```
    *p_Kp = (byte) read_buffer[3];
    *(p_Kp + 1) = (byte) read_buffer[2];
    *(p_Kp + 2) = (byte) read_buffer[1];
    *(p_Kp + 3) = (byte) read_buffer[0];
    start_pid_count = (uint8_t)(read_buffer[4]);
  }
  break;
}
case 'i':
{
  Serial.readBytes(read_buffer, 5);
  if (Serial.read() == ';')
  {
    *p_Ki = (byte) read_buffer[3];
    *(p_Ki + 1) = (byte) read_buffer[2];
    *(p_Ki + 2) = (byte) read_buffer[1];
    *(p_Ki + 3) = (byte) read_buffer[0];
    start_pid_count = (uint8_t)(read_buffer[4]);
  }
  break;
}
case 'd':
{
  Serial.readBytes(read_buffer, 5);
  if (Serial.read() == ';')
  {
    *p_Kd = (byte) read_buffer[3];
    *(p_Kd + 1) = (byte) read_buffer[2];
    *(p_Kd + 2) = (byte) read_buffer[1];
```

```
            *(p_Kd + 3) = (byte) read_buffer[0];
            start_pid_count = (uint8_t)(read_buffer[4]);
          }
          break;
        }
        myPID.SetTunings(Kp, Ki, Kd);
      }
    }
    break;
  }
}

if (flag_norm)
{
  flag_norm=0;

  voltage1 = map(analogRead(A1), 0, 1023, 0, 5000);
  tt_per = voltage1/1000.0;
  tt_per = (tt_per-0.8)/3.18*5.0;
  myPID.Compute();
  if (Uk < 0.0)        { pwm = 0.0;  }
  else  if (Uk > 3.18)  { pwm = 0.99; }
  else                 { pwm = Uk/3.18; }

  temp = (unsigned short int) (pwm * tmp);
  OCR1A = temp;
}

if (flag_com)
```

```
  {
    flag_com = 0;
    Serial.print("mm = ");
    send_num_float(mm_per);
    Serial.print("; tt = ");
    send_num_float(tt_per);
    Serial.print("; pwm = ");
    send_num_float(pwm);
    Serial.print("; Uk = ");
    send_num_float(Uk);
    Serial.println(".");
  }
}


ISR (TIMER1_OVF_vect)
{
  if (++count_norm == 200)   // Tần số tính toán là 100Hz <=> Chu kì
0,01s
  {
    flag_norm = 1;
    count_norm = 0;
    count_com += 1;
  }
  if (count_com == 5){    // Tần số gửi Serial là 20Hz <-> Chu kì gửi 0,05s
    flag_com = 1;
    count_com = 0;
  }


}
```

2. Giải thích code :

2.1. Bắt đầu chương trình :

- Khởi tạo các biến lưu giá trị chuyển đổi tín hiệu từ cảm biến TPS, giá trị tín hiệu từ Labview, các giá trị trung gian tính toán các thông số đầu vào và các biến xác định trạng thái, thời điểm

- Thiết lập clock và cấp clock cho ngoại vi TIM1

- Thiết lập TIM1 tạo chu kì lấy mẫu đọc ADC từ Labview và cảm biến

- Thiết lập ngõ vào vi điều khiển :

    + Chân Input ADC A0, A1 đọc tín hiệu từ TPS và Labview

    + Chân Output 10 điều khiển PWM cầu H

- Bật TIM1

Code thực hiện :

```
#include <avr/interrupt.h>
#include <PID_v1.h>

#define motor 10
#define resistor A0
#define sensor A1

bool flag = 0;
unsigned int count;

double pwm, setpoint, input, output;
double kp = 30, ki = 20, kd = 0.01;
PID myPID(&input, &output, &setpoint, kp, ki, kd, DIRECT);
```

2.1.1 Khai báo thư viện

```
#include <avr/interrupt.h>
#include <PID_v1.h>
```

Giải thích :

- Thư viện *<avr/interupt.h>* hỗ trợ cho việc thiết lập và sử dụng trình ngắt trong Arduino.

- Thư viện *<PID_v1.h>* hỗ trợ cho việc tính toán PID.

2.1.2 Khai báo các biến đếm và đánh dấu :

```
bool flag = 0;
unsigned int count;
```

Trong đó :

- *flag* : biến boolean dùng để xác định cờ giao tiếp giữa Arduino và Labview

- *count* : biến đếm số lần lặp timer.

2.1.3. Khai báo các thông số cấu hình bộ tính toán PID :

```
double pwm, setpoint, input, output;
double kp = 30, ki = 20, kd = 0.01;
PID myPID(&input, &output, &setpoint, kp, ki, kd, DIRECT);
```

2.1.4. Định nghĩa các chân Arduino :

```
#define motor 10
#define resistor A0
#define sensor A1
```

Giải thích :

- Chân số 10 : chân PWM điều khiển cầu H.

- Chân A0 : chân analog đọc giá trị từ Labview.

- Chân A1 : chân analog đọc giá trị từ cảm biến.

2.2. Thiết lập bộ định thời Timer – Counter :

Code thực hiện :

```
cli();
TCCR1A = 0;
TCCR1B = 0;
TIMSK1 = 0;
TCCR1B |= (1 << WGM13) | (1 << WGM12) | (1 << CS11) | (1 << CS10);
TCCR1A |= (1 << WGM11) | (1 << COM1A1);
TCNT1 = 0;
ICR1 = ((unsigned short int) (16000000.0/64.0/1000.0)) - 1;
OCR1A = 0;
TIMSK1 = (1 << TOIE1);
sei();
```

Giải thích :

- cli() : dùng để tắt ngắt toàn cục.

- Reset các thanh ghi :

```
TCCR1A = 0;   TCCR1B = 0;   TIMSK1 = 0;
```

Trong thanh ghi **TCCR1B :**

- Sử dụng 3 bit **CS10**, **CS11**, **CS12** để lựa chọn xung nhịp cho T/C1.

- Các bit WGM10, WGM11, WGM12, WGM13 dùng để chọn chế độ cho Timer 1.

- **TOIE1**: Overflow Interrupt Enable 1 - Cho phép ngắt khi xảy ra tràn trên T/C.

- sei() : dùng để bật ngắt toàn cục.

2.3. Đọc giá trị ADC và tính toán giá trị output :

Code thực hiện :

```
if (flag_norm)
  {
    flag_norm=0;

    voltage1 = map(analogRead(A1), 0, 1023, 0, 5000);
    tt_per = voltage1/1000.0;
    tt_per = (tt_per-0.8)/3.18*5.0;
    myPID.Compute();
    if (Uk < 0.0)        { pwm = 0.0;  }
    else  if (Uk > 3.18)  { pwm = 0.99; }
    else                 { pwm = Uk/3.18; }

    temp = (unsigned short int) (pwm * tmp);
    OCR1A = temp;
  }
```

- voltage0, voltage1 : các biến tượng trưng cho điện áp đọc từ các chân analog
- Đọc giá trị A0 và tính toán giá trị độ mở bướm ga mong muốn (setpoint)
- Đọc giá trị A1 và tính toán giá trị độ mở bướm ga thực tế (input)
- Tính toán giá trị điện áp điều khiển : myPID.Compute();
- Tính giá trị PWM theo giá trị của output :
  + output < 0.0 : bướm ga đang cân bằng, không cần tăng điện áp → pwm = 0
  + output > 50.0 : bướm ga mở hết mức → pwm = 1.0 (100%)
  + còn lại : pwm = output/50.0

2. 4. Mỗi chu kì timer :

Code thực hiện :

```
ISR (TIMER1_OVF_vect)
```

```
{
 if (++count_norm == 200)   // Tần số tính toán là 100Hz <=> Chu kì 0,01s
 {
  flag_norm = 1;
  count_norm = 0;
  count_com += 1;
 }
 if (count_com == 5){    // Tần số gửi Serial là 20Hz <-> Chu kì gửi 0,05s
  flag_com = 1;
  count_com = 0;
 }

}
```

Giải thích code :

- Tăng biến đếm lên 1
- Nếu biến đếm chu kì bằng 10 thì
    + Bật cờ giao tiếp
    + Đặt giá trị biến đếm về 0.

- **ISR** (*Vector_name*) là các trình phục vụ ngắt, trong đó ISR là keyword, Vector_name ở chương trình này là *TIMER1_OVF_vect*, có nghĩa là "Ngắt tràn trên Timer/Counter1".

2. 5. Sự kiện giao tiếp Serial :

Code thực hiện :

```
 if (flag_com)
 {
  flag_com = 0;
  Serial.print("mm = ");
  send_num_float(mm_per);
  Serial.print("; tt = ");
```

```
            send_num_float(tt_per);
            Serial.print("; pwm = ");
            send_num_float(pwm);
            Serial.print("; Uk = ");
            send_num_float(Uk);
            Serial.println(".");
          }
```

Giải thích code :

- Tắt cờ giao tiếp
- Xuất các giá trị setpoint, input, PWM ra Serial.

TRÍCH DẪN

[1] https://oto.edu.vn/tim-hieu-ve-cam-bien-vi-tri-buom-ga-tps-sensor/

[2] http://www.ttundra.com/829/throttle_pedal_position_sensor_switch_a_circuit_p0120_p0123_p0220_p0222_p0223_p2135_.html

[3] https://vi.wikipedia.org/wiki/B%E1%BB%99_%C4%91i%E1%BB%81u_khi%E1%BB%83n_PID

[4] https://www.handsontec.com/dataspecs/module/BTS7960%20Motor%20Driver.pdf

[5] https://electronics.stackexchange.com/questions/430642/how-to-control-bts7960-43a-motor-driver-directions

[6] https://www.hackster.io/news/picking-the-right-arduino-341a0a9550c7

[7] https://www.elektor.com/arduino-uno-r3