ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

Nguyễn Vũ Quang

XÂY DỰNG KHUNG ĐẾ ROBOT CÓ KHẢ NĂNG CÂN BẰNG TRỌNG TÂM

KHÓA LUẬN TỐT NGHIỆP ĐẠI HỌC HỆ CHÍNH QUY

Ngành: Kỹ thuật điều khiển và tự động hóa

HÀ NỘI – 2025

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

Nguyễn Vũ Quang

XÂY DỰNG KHUNG ĐẾ ROBOT CÓ KHẢ NĂNG CÂN BẰNG TRỌNG TÂM

KHÓA LUẬN TỐT NGHIỆP ĐẠI HỌC HỆ CHÍNH QUY
Ngành: Kỹ thuật điều khiển và tự động hóa

Cán bộ hướng dẫn: ThS. Đặng Anh Việt

HÀ NỘI – 2025

LỜI CAM ĐOAN

Tôi xin cam đoan đây là công trình nghiên cứu của riêng tôi dưới sự hướng dẫn của ThS. Đặng Anh Việt. Các số liệu, kết quả nêu trong khóa luận là trung thực và chưa từng được ai công bố trong bất kỳ công trình nào khác.

Tôi xin cam đoan rằng mọi sự giúp đỡ cho việc thực hiện khóa luận này đã được cảm ơn và các thông tin trích dẫn trong khóa luận đã được chỉ rõ nguồn gốc.

Hà Nội, ngày tháng năm 2025

Sinh viên thực hiện

Nguyễn Vũ Quang

LỜI CẢM ƠN

Để hoàn thành khóa luận tốt nghiệp này, tôi đã nhận được rất nhiều sự giúp đỡ và hỗ trợ từ thầy cô, gia đình và bạn bè.

Trước hết, tôi xin gửi lời cảm ơn chân thành và sâu sắc nhất đến ThS. Đặng Anh Việt – người đã trực tiếp hướng dẫn, chỉ bảo tận tình và tạo mọi điều kiện thuận lợi cho tôi trong suốt quá trình thực hiện khóa luận.

Tôi xin chân thành cảm ơn các thầy cô giáo trong Khoa Điện tử - Viễn thông, Trường Đại học Công nghệ, Đại học Quốc gia Hà Nội đã trang bị cho tôi những kiến thức quý báu trong suốt thời gian học tập tại trường.

Cuối cùng, tôi xin gửi lời cảm ơn đến gia đình, bạn bè đã luôn động viên, khích lệ tôi trong suốt quá trình học tập và thực hiện khóa luận.

Hà Nội, tháng 6 năm 2025

Sinh viên

Nguyễn Vũ Quang

TÓM TẮT

Tóm tắt: Khóa luận trình bày quá trình nghiên cứu, thiết kế và chế tạo khung đế robot di động có khả năng tự động cân bằng trọng tâm. Hệ thống sử dụng cảm biến lực (Loadcell) kết hợp với module HX711 để đo chênh lệch trọng lượng giữa hai bên khung. Thuật toán điều khiển PID được áp dụng để điều khiển động cơ bước di chuyển khối đối trọng trên cơ cấu vitme, từ đó bù đắp sự mất cân bằng khi tải thay đổi.

Kết quả thực nghiệm cho thấy hệ thống có khả năng phát hiện và bù đắp độ lệch trọng tâm trong phạm vi ±50g với thời gian đáp ứng nhanh. Chương trình điều khiển được viết theo kiến trúc Non-blocking, đảm bảo động cơ bước hoạt động mượt mà ở tốc độ cao mà không bị ảnh hưởng bởi các tác vụ đọc cảm biến hay giao tiếp Serial.

Khóa luận cũng đề xuất các hướng phát triển mở rộng như điều khiển vị trí, tốc độ chuyển dịch và tích hợp vào robot di động thực tế.

Từ khóa: *Cân bằng trọng tâm, Loadcell, PID, Động cơ bước, Arduino, Non-blocking, Robot di động.*

Mục lục

Danh sách hình vẽ

Danh sách bảng

Danh sách các từ viết tắt

PID: Proportional-Integral-Derivative – Bộ điều khiển vi tích phân tỷ lệ.

PWM: Pulse Width Modulation – Điều chế độ rộng xung.

ADC: Analog to Digital Converter – Bộ chuyển đổi tương tự sang số.

I2C: Inter-Integrated Circuit – Giao thức truyền thông nối tiếp.

SPI: Serial Peripheral Interface – Giao diện ngoại vi nối tiếp.

UART: Universal Asynchronous Receiver-Transmitter – Bộ thu phát không đồng bộ.

GPIO: General Purpose Input/Output – Chân vào/ra đa mục đích.

RPM: Revolutions Per Minute – Vòng quay trên phút.

ISR: Interrupt Service Routine – Trình phục vụ ngắt.

Chương 1.
Tổng quan

1.1. Đặt vấn đề và lý do chọn đề tài

Robot di động ngày càng được ứng dụng rộng rãi trong nhiều lĩnh vực như công nghiệp, y tế, dịch vụ và nghiên cứu khoa học. Một trong những thách thức quan trọng khi thiết kế robot di động là đảm bảo sự ổn định và cân bằng của robot, đặc biệt khi robot mang theo các tải trọng thay đổi hoặc có cánh tay robot gắn trên thân.

Khi tải trọng trên robot thay đổi hoặc phân bố không đều, trọng tâm của hệ thống sẽ bị lệch. Điều này gây ra nhiều vấn đề như: robot dễ bị lật, chuyển động không ổn định, giảm khả năng vượt địa hình, và tăng tiêu hao năng lượng. Đặc biệt với các robot có gắn cánh tay thao tác (manipulator), khi cánh tay vươn ra xa hoặc nâng vật nặng, trọng tâm sẽ thay đổi đáng kể.

Để giải quyết vấn đề này, một giải pháp hiệu quả là thiết kế hệ thống cân bằng trọng tâm tự động. Hệ thống này sử dụng cảm biến để phát hiện độ lệch trọng tâm và điều khiển cơ cấu chấp hành để di chuyển khối đối trọng, từ đó bù đắp sự mất cân bằng.

Xuất phát từ nhu cầu thực tiễn trên, đề tài "Xây dựng khung đế robot có khả năng cân bằng trọng tâm" được lựa chọn nhằm nghiên cứu và chế tạo một khung đế có khả năng tự động điều chỉnh trọng tâm, phù hợp để tích hợp vào các robot di động.

1.2. Mục tiêu nghiên cứu

Mục tiêu chính của khóa luận là thiết kế và chế tạo một khung đế robot có khả năng tự động cân bằng trọng tâm. Cụ thể, các mục tiêu bao gồm:

Nghiên cứu nguyên lý cân bằng trọng tâm và phương pháp đo lường độ lệch trọng tâm sử dụng cảm biến lực.

Thiết kế và chế tạo khung cơ khí phù hợp với kích thước robot di động, tích hợp cơ cấu di chuyển đối trọng bằng vitme và động cơ bước.

Thiết kế mạch điện tử điều khiển, bao gồm module đọc cảm biến lực và driver điều khiển động cơ bước.

Xây dựng chương trình điều khiển sử dụng thuật toán PID để tự động điều chỉnh vị trí đối trọng.

Thử nghiệm và đánh giá hiệu quả hoạt động của hệ thống.

1.3. Đối tượng và phạm vi nghiên cứu

1.3.1. Đối tượng nghiên cứu

Đối tượng nghiên cứu của khóa luận bao gồm: cảm biến lực Loadcell và module chuyển đổi HX711, động cơ bước và driver điều khiển TB6600, vi điều khiển Arduino, thuật toán điều khiển PID, và kỹ thuật lập trình thời gian thực cho hệ thống nhúng.

1.3.2. Phạm vi nghiên cứu

Khóa luận tập trung vào việc cân bằng trọng tâm theo một trục (trái-phải). Khung đế có kích thước 35×35cm, phù hợp với các robot di động cỡ nhỏ và vừa. Hệ thống điều khiển vòng hở cho động cơ bước, sử dụng thuật toán PID để điều chỉnh vị trí đối trọng. Phạm vi cân bằng trong khoảng ±500g độ lệch.

1.4. Phương pháp nghiên cứu

Khóa luận sử dụng kết hợp các phương pháp nghiên cứu sau:

Phương pháp nghiên cứu lý thuyết: Nghiên cứu các tài liệu về cơ học, lý thuyết điều khiển, cảm biến và vi điều khiển để xây dựng cơ sở lý thuyết cho đề tài.

Phương pháp thiết kế: Thiết kế cơ khí sử dụng phần mềm CAD, thiết kế mạch điện tử, và thiết kế phần mềm điều khiển.

Phương pháp thực nghiệm: Chế tạo mô hình thực tế, tiến hành thử nghiệm và đo đạc để đánh giá hiệu quả hoạt động của hệ thống.

Phương pháp phân tích: Phân tích kết quả thực nghiệm, so sánh với mục tiêu đề ra và đề xuất các cải tiến.

1.5. Bố cục khóa luận

Khóa luận được trình bày trong 5 chương với nội dung như sau:

Chương 1: Tổng quan – Trình bày lý do chọn đề tài, mục tiêu, đối tượng, phạm vi và phương pháp nghiên cứu.

Chương 2: Cơ sở lý thuyết – Trình bày các kiến thức nền tảng về cân bằng trọng tâm, cảm biến lực, động cơ bước, vi điều khiển và thuật toán PID.

Chương 3: Thiết kế hệ thống – Mô tả chi tiết quá trình thiết kế cơ khí, mạch điện và phần mềm điều khiển.

Chương 4: Thực nghiệm và đánh giá – Trình bày quá trình thử nghiệm, kết quả đo đạc và phân tích đánh giá.

Chương 5: Kết luận và hướng phát triển – Tổng kết kết quả đạt được, nêu hạn chế và đề xuất hướng phát triển.

Chương 2.
Cơ sở lý thuyết

2.1. Lý thuyết cân bằng trọng tâm và momen lực

2.1.1. Khái niệm trọng tâm

Trọng tâm của một vật thể là điểm mà tại đó toàn bộ trọng lượng của vật có thể được coi như tập trung. Đối với một hệ thống gồm nhiều vật thể, vị trí trọng tâm được xác định bởi công thức:

$$x_G = \frac{\sum_{i=1}^{n} m_i \cdot x_i}{\sum_{i=1}^{n} m_i} \tag{1}$$

trong đó x_G là tọa độ trọng tâm, m_i là khối lượng của vật thể thứ i, và x_i là tọa độ của vật thể thứ i.

2.1.2. Momen lực và điều kiện cân bằng

Momen lực (hay torque) là đại lượng đặc trưng cho tác dụng làm quay của lực đối với một trục quay. Momen lực được tính theo công thức:

$$M = F \times d \tag{2}$$

trong đó M là momen lực (N.m), F là lực tác dụng (N), và d là cánh tay đòn – khoảng cách từ điểm đặt lực đến trục quay (m).

Điều kiện để một hệ thống cân bằng là tổng các momen lực tác dụng lên hệ bằng không:

$$\sum M = 0 \tag{3}$$

2.1.3. Nguyên lý cân bằng trọng tâm cho robot

Áp dụng vào hệ thống khung đế robot, để cân bằng trọng tâm khi có tải lệch, cần di chuyển một khối đối trọng sao cho momen do đối trọng tạo ra bù đắp momen do tải gây ra:

$$m_{slider} \times d_{slider} = m_{load} \times d_{load} \quad (4)$$

trong đó m_{slider} là khối lượng đối trọng, d_{slider} là khoảng cách từ đối trọng đến tâm, m_{load} là khối lượng tải lệch, và d_{load} là khoảng cách từ tải đến tâm.

2.2. Cảm biến lực Loadcell và module HX711

2.2.1. Nguyên lý hoạt động của Loadcell

Loadcell là cảm biến lực hoạt động dựa trên hiệu ứng điện trở biến dạng (strain gauge). Khi có lực tác dụng, thanh đàn hồi bên trong loadcell bị biến dạng, làm thay đổi điện trở của các strain gauge được dán trên bề mặt.

Các strain gauge được mắc theo cấu hình cầu Wheatstone để chuyển đổi sự thay đổi điện trở thành sự thay đổi điện áp. Điện áp đầu ra rất nhỏ (cỡ mV) và tỷ lệ thuận với lực tác dụng.

2.2.2. Module HX711

HX711 là IC chuyển đổi tương tự sang số (ADC) 24-bit chuyên dụng cho loadcell. Các đặc điểm chính của HX711 bao gồm: độ phân giải 24-bit, tốc độ lấy mẫu 10Hz hoặc 80Hz (tùy cấu hình chân RATE), có bộ khuếch đại tích hợp với độ lợi 32, 64 hoặc 128, và giao tiếp với vi điều khiển qua 2 dây (DOUT và SCK).

2.2.3. Thư viện HX711_ADC

Trong dự án này, thư viện HX711_ADC của tác giả Olav Kallhovd được sử dụng thay vì thư viện HX711 chuẩn. Ưu điểm chính của thư viện này là hỗ trợ chế độ Non-blocking:

Hàm `update()` kiểm tra dữ liệu mới mà không chặn chương trình. Nếu chưa có dữ liệu, hàm trả về ngay lập tức (tốn 0ms). Nếu có dữ liệu mới, hàm lưu vào biến tạm.

Hàm `getData()` lấy giá trị từ biến tạm ra một cách tức thì.

Cơ chế này cho phép vi điều khiển thực hiện các tác vụ khác (như điều khiển động cơ) trong khi chờ dữ liệu từ loadcell, đảm bảo hệ thống hoạt động thời gian thực.

2.3. Động cơ bước và driver TB6600

2.3.1. Nguyên lý động cơ bước

Động cơ bước (stepper motor) là loại động cơ điện chuyển đổi các xung điện rời rạc thành chuyển động quay theo từng bước cố định. Mỗi xung điều khiển làm rotor quay một góc xác định (thường là 1.8° cho động cơ 200 bước/vòng).

Ưu điểm của động cơ bước trong ứng dụng điều khiển vị trí: điều khiển vị trí chính xác mà không cần encoder phản hồi, momen giữ lớn khi đứng yên, dễ dàng điều khiển bằng xung số, và phù hợp với ứng dụng yêu cầu dừng chính xác.

2.3.2. Chế độ vi bước (Microstepping)

Vi bước là kỹ thuật chia nhỏ mỗi bước đầy đủ thành nhiều bước nhỏ hơn bằng cách điều khiển dòng điện qua các cuộn dây theo dạng sóng sin. Ví dụ, chế độ 1/16 microstep chia mỗi bước 1.8° thành 16 bước nhỏ, mỗi bước 0.1125°.

Số bước trên mỗi vòng quay khi sử dụng microstep:

$$N_{micro} = N_{full} \times k \tag{5}$$

trong đó $N_{full} = 200$ bước/vòng và k là hệ số microstep (1, 2, 4, 8, 16...).

2.3.3. Driver TB6600

TB6600 là driver công suất cao cho động cơ bước với các thông số: điện áp làm việc 9-42V DC, dòng điện tối đa 4A/pha, hỗ trợ microstep từ full-step đến 1/32, và có tín hiệu điều khiển: PUL (xung), DIR (chiều), ENA (kích hoạt).

2.4. Vi điều khiển Arduino

Arduino là nền tảng vi điều khiển mã nguồn mở phổ biến trong các dự án điện tử. Trong khóa luận này, Arduino Uno/Nano (chip ATmega328P) được sử dụng với các thông số: tần số xung nhịp 16MHz, bộ nhớ Flash 32KB, bộ nhớ SRAM 2KB, 14 chân I/O số và 6 chân analog.

Giới hạn quan trọng cần lưu ý khi sử dụng Arduino trong điều khiển thời gian thực: với tần số 16MHz, mỗi chu kỳ lệnh mất khoảng 62.5ns. Tuy nhiên, các thao tác phức tạp như đọc ADC, giao tiếp Serial có thể tiêu tốn hàng nghìn chu kỳ.

2.5. Thuật toán điều khiển PID

2.5.1. Nguyên lý PID

PID (Proportional-Integral-Derivative) là thuật toán điều khiển vòng kín phổ biến nhất trong công nghiệp. Bộ điều khiển PID tính toán tín hiệu điều khiển dựa trên sai số giữa giá trị mong muốn (setpoint) và giá trị thực tế (process variable):

$$u(t) = K_p \cdot e(t) + K_i \cdot \int_0^t e(\tau)d\tau + K_d \cdot \frac{de(t)}{dt} \tag{6}$$

trong đó $u(t)$ là tín hiệu điều khiển, $e(t)$ là sai số, K_p là hệ số tỷ lệ, K_i là hệ số tích phân, và K_d là hệ số vi phân.

2.5.2. Vai trò của từng thành phần

Thành phần tỷ lệ (P) tạo ra tín hiệu điều khiển tỷ lệ thuận với sai số hiện tại. Kp càng lớn, hệ thống phản ứng càng nhanh nhưng có thể gây dao động.

Thành phần tích phân (I) tích lũy sai số theo thời gian, giúp triệt tiêu sai số xác lập (steady-state error). Tuy nhiên, Ki quá lớn có thể gây hiện tượng vọt lố (overshoot).

Thành phần vi phân (D) dự đoán xu hướng thay đổi của sai số, giúp giảm dao động và tăng tốc độ đáp ứng.

2.5.3. Điều chỉnh tham số PID

Phương pháp thử-sai (Trial and Error) thường được sử dụng để tinh chỉnh PID:

Bước 1: Đặt Ki = Kd = 0, tăng Kp cho đến khi hệ thống bắt đầu dao động.

Bước 2: Giảm Kp xuống khoảng 60-70% giá trị gây dao động.

Bước 3: Tăng Kd để giảm dao động và vọt lố.

Bước 4: Thêm Ki nhỏ để triệt tiêu sai số xác lập (nếu cần).

2.6. Lập trình thời gian thực cho hệ thống nhúng

2.6.1. Vấn đề Blocking trong điều khiển động cơ bước

Khi sử dụng thư viện AccelStepper để điều khiển động cơ bước, hàm `runSpeed()` cần được gọi liên tục với tần số cao. Ở chế độ 1/16 microstep với tốc độ 300 RPM, khoảng cách giữa các xung chỉ khoảng 62.5μs.

Nếu trong vòng lặp có các hàm blocking như `delay()`, `Serial.print()` (khi buffer đầy), hoặc `scale.get_units()` (đọc loadcell blocking), vi điều khiển sẽ bị "đứng hình" trong thời gian đó, khiến động cơ bị mất xung và chạy giật cục.

2.6.2. Giải pháp Non-blocking

Giải pháp là sử dụng cơ chế Time-Slicing với hàm `millis()`:

Nhiệm vụ nền (chạy mỗi vòng lặp): Gọi `LoadCell.update()` và `stepper.runSpeed()`.

Nhiệm vụ điều khiển (20ms/lần): Đọc dữ liệu loadcell, tính toán PID, cập nhật tốc độ động cơ.

Nhiệm vụ giám sát (100ms/lần): In thông tin ra Serial để debug.

Cấu trúc này đảm bảo hàm `runSpeed()` được gọi đủ thường xuyên để duy trì chuyển động mượt mà của động cơ.

Chương 3.
Thiết kế hệ thống

3.1. Yêu cầu thiết kế

Dựa trên mục tiêu đề tài và điều kiện thực tế, hệ thống cần đáp ứng các yêu cầu sau:

Về cơ khí: Khung đế có kích thước phù hợp để gắn lên robot di động (khoảng 35×35cm). Cơ cấu di chuyển đối trọng mượt mà, độ chính xác cao. Hành trình di chuyển đủ lớn để bù đắp độ lệch trọng tâm trong phạm vi yêu cầu.

Về điện tử: Đo được chênh lệch trọng lượng giữa hai bên với độ phân giải cao. Điều khiển động cơ bước chính xác về vị trí và tốc độ. Xử lý tín hiệu và tính toán PID trong thời gian thực.

Về phần mềm: Chương trình điều khiển ổn định, không bị giật lag. Có khả năng tinh chỉnh tham số PID. Hiển thị thông tin giám sát qua Serial.

3.2. Thiết kế cơ khí

3.2.1. Khung nhôm định hình

Khung đế được xây dựng từ nhôm định hình 20×20mm với kích thước tổng thể 35×35cm. Nhôm định hình được chọn vì: dễ gia công và lắp ráp, độ cứng vững cao, có rãnh để gắn các linh kiện, và nhẹ nhưng chịu lực tốt.

3.2.2. Cơ cấu vitme - thanh trượt

Vitme bi (ball screw) với bước ren 2mm được sử dụng để chuyển đổi chuyển động quay của động cơ thành chuyển động tịnh tiến của khối đối trọng. Tính toán tốc độ di chuyển: với 1/16 microstep (3200 bước/vòng) và tốc độ tối đa 4000 bước/giây:

$$v_{max} = \frac{4000}{3200} \times 2 = 2.5 \text{ mm/s} \tag{7}$$

Thanh trượt tròn đường kính 8mm được lắp song song với vitme để đảm bảo chuyển động thẳng và chịu tải trọng.

Bảng 3.1. Bảng kết nối phần cứng

Thiết bị	Chân thiết bị	Chân Arduino
HX711 Trái	DT SCK	Pin 4 Pin 5
HX711 Phải	DT SCK	Pin 6 Pin 7
Driver TB6600	PUL+ (STEP) DIR+ ENA+ Các chân (-)	Pin 8 Pin 9 Không kết nối GND
Công tắc hành trình	NO COM	Pin 10 GND

3.2.3. Bố trí Loadcell

Hệ thống sử dụng 2 cụm loadcell 50kg (mỗi cụm 2 loadcell ghép thành cầu Wheatstone hoàn chỉnh) đặt ở hai bên trái và phải của khung. Độ chênh lệch giữa giá trị đọc từ hai cụm loadcell phản ánh độ lệch trọng tâm.

3.2.4. Giới hạn hành trình

Hành trình di chuyển của đối trọng: 120mm về bên trái và 110mm về bên phải (tính từ vị trí tâm). Công tắc hành trình được lắp ở vị trí biên phải để xác định điểm gốc (Home) khi khởi động.

3.3. Thiết kế mạch điện

3.3.1. Sơ đồ khối hệ thống

Hệ thống gồm các khối chức năng: Khối cảm biến (2 cụm Loadcell + 2 module HX711), Khối xử lý trung tâm (Arduino), Khối điều khiển động cơ (Driver TB6600 + Động cơ bước Nema 17), Khối nguồn (12V cho động cơ, 5V cho mạch điều khiển), và Khối giới hạn hành trình (Công tắc hành trình).

3.3.2. Kết nối phần cứng

Bảng kết nối các chân Arduino:

3.4. Thiết kế phần mềm

3.4.1. Cấu trúc chương trình

Chương trình được tổ chức theo mô hình Superloop với cơ chế Time-Slicing:

```
void loop() {
    // Nhiem vu nen - chay moi vong lap
    LoadCellLeft.update();
    LoadCellRight.update();
    stepper.runSpeed();

    // Nhiem vu dieu khien - 20ms/lan
    if (millis() > t_pid + 20) {
        // Doc cam bien, tinh PID, cap nhat toc do
        t_pid = millis();
    }

    // Nhiem vu giam sat - 100ms/lan
    if (millis() > t_print + 100) {
        // In thong tin ra Serial
        t_print = millis();
    }
}
```

3.4.2. Quy trình Homing

Khi khởi động, hệ thống thực hiện quy trình Homing 2 giai đoạn để xác định điểm gốc chính xác:

Giai đoạn 1 (Tìm nhanh): Di chuyển với tốc độ cao về phía công tắc hành trình. Khi chạm công tắc, dừng lại và đánh dấu vị trí tạm thời.

Giai đoạn 2 (Tìm chậm): Lùi ra 5mm, sau đó di chuyển chậm trở lại cho đến khi chạm công tắc lần thứ hai. Vị trí này được ghi nhận là điểm gốc chính xác.

Sau đó, con trượt di chuyển về vị trí tâm (0) và chờ 5 giây để hệ thống ổn định trước khi bắt đầu cân bằng.

3.4.3. Logic điều khiển PID với Deadzone và Hysteresis

Để tránh hiện tượng động cơ rung lắc khi độ lệch nhỏ, hệ thống áp dụng vùng chết (Deadzone) ± 50g và độ trễ (Hysteresis) ± 10g:

Nếu đang dừng và độ lệch > 60g: Bắt đầu cân bằng.

Nếu đang chạy và độ lệch < 40g: Dừng cân bằng.

Logic Hysteresis này ngăn chặn việc bật/tắt liên tục khi độ lệch dao động quanh ngưỡng 50g.

3.4.4. Xử lý tốc độ tối thiểu

Động cơ bước có vùng tốc độ thấp gây cộng hưởng và rung lắc. Để tránh điều này, khi PID tính ra tốc độ > 0 nhưng < 2000 bước/s, hệ thống ép tốc độ lên mức tối thiểu 2000 bước/s để đảm bảo động cơ hoạt động mượt mà.

Chương 4.
Thực nghiệm và đánh giá

4.1. Môi trường thử nghiệm

Hệ thống được thử nghiệm trong môi trường phòng thí nghiệm với các điều kiện: nguồn điện ổn định 12V/5A cho động cơ và 5V từ USB cho Arduino, khung đế đặt trên mặt phẳng ngang, và các vật nặng chuẩn (100g, 200g, 500g) để tạo độ lệch.

Phần mềm giám sát: Arduino Serial Monitor và Serial Plotter ở tốc độ 115200 baud.

4.2. Tinh chỉnh tham số PID

4.2.1. Quá trình tinh chỉnh

Áp dụng phương pháp thử-sai với các bước:

Bước 1: Bắt đầu với Kp = 10, Ki = 0, Kd = 0. Đặt vật 200g lên một bên, quan sát phản ứng. Hệ thống phản ứng chậm, tăng Kp.

Bước 2: Tăng Kp lên 25. Hệ thống phản ứng nhanh hơn nhưng có dao động nhẹ quanh điểm cân bằng.

Bước 3: Thêm Ki = 0.05 để triệt tiêu sai số xác lập. Hệ thống đạt cân bằng ổn định.

Bước 4: Với Kd = 0, hệ thống hoạt động tốt cho các thay đổi tải từ từ. Có thể thêm Kd nhỏ nếu cần đáp ứng nhanh hơn với thay đổi đột ngột.

4.2.2. Tham số PID cuối cùng

Sau quá trình tinh chỉnh, các tham số PID được chọn: Kp = 25, Ki = 0.05, Kd = 0.

4.3. Kết quả thực nghiệm

4.3.1. Thử nghiệm với tải tĩnh

Đặt vật 200g lên bên phải khung, hệ thống phát hiện độ lệch và di chuyển đối trọng sang trái. Sau khoảng 3-5 giây, độ lệch giảm về dưới ngưỡng 50g và hệ thống dừng lại ở trạng thái cân bằng.

4.3.2. Thử nghiệm với tải thay đổi

Di chuyển vật nặng từ bên này sang bên kia, hệ thống theo dõi và điều chỉnh liên tục. Đối trượt di chuyển mượt mà, không có hiện tượng giật cục nhờ kiến trúc Non-blocking.

4.3.3. Giới hạn khả năng cân bằng

Với đối trọng 200g và hành trình 120mm, khả năng cân bằng tối đa:

$$m_{load,max} = \frac{200 \times 120}{175} \approx 137 \text{ g} \quad (8)$$

trong đó 175mm là khoảng cách từ tâm khung đến vị trí đặt tải (một nửa chiều rộng khung 35cm).

Để cân bằng độ lệch lớn hơn, cần tăng khối lượng đối trọng hoặc tăng hành trình.

4.4. Phân tích và đánh giá

4.4.1. Ưu điểm

Hệ thống hoạt động ổn định, động cơ chạy mượt mà ở tốc độ cao nhờ kiến trúc Non-blocking. Logic Deadzone và Hysteresis hiệu quả trong việc ngăn chặn rung lắc. Quy trình Homing 2 giai đoạn đảm bảo xác định điểm gốc chính xác.

4.4.2. Hạn chế

Tốc độ di chuyển chậm (2.5mm/s) do sử dụng vitme bước nhỏ (2mm) và microstep cao (1/16). Khả năng cân bằng hạn chế bởi khối lượng đối trọng nhỏ (200g). Chỉ cân bằng theo một trục (trái-phải).

4.4.3. So sánh với mục tiêu

Mục tiêu phát hiện và bù đắp độ lệch trọng tâm: Đạt được trong phạm vi ±137g. Mục tiêu động cơ hoạt động mượt mà: Đạt được nhờ kiến trúc Non-blocking. Mục tiêu tinh chỉnh PID: Đạt được với bộ tham số Kp=25, Ki=0.05, Kd=0.

Chương 5.
Kết luận và hướng phát triển

5.1. Kết quả đạt được

Khóa luận đã hoàn thành các mục tiêu đề ra:

Về lý thuyết: Nghiên cứu và trình bày đầy đủ cơ sở lý thuyết về cân bằng trọng tâm, cảm biến lực, động cơ bước, thuật toán PID và lập trình thời gian thực cho hệ thống nhúng.

Về thiết kế: Thiết kế hoàn chỉnh hệ thống khung đế cân bằng bao gồm cơ khí (khung nhôm, cơ cấu vitme-thanh trượt), mạch điện (kết nối Arduino, HX711, TB6600), và phần mềm điều khiển.

Về chế tạo: Chế tạo thành công mô hình khung đế hoạt động ổn định. Hệ thống có khả năng phát hiện và bù đắp độ lệch trọng tâm trong phạm vi thiết kế.

Về phần mềm: Xây dựng chương trình điều khiển theo kiến trúc Non-blocking, đảm bảo động cơ bước hoạt động mượt mà. Tích hợp thuật toán PID với các kỹ thuật Deadzone, Hysteresis và Min Speed Cutoff.

5.2. Hạn chế của đề tài

Tốc độ phản hồi: Do sử dụng vitme bước nhỏ (2mm) kết hợp microstep cao (1/16), tốc độ di chuyển đối trọng chỉ đạt 2.5mm/s, chưa đáp ứng được các tình huống thay đổi tải đột ngột.

Khả năng cân bằng: Với đối trọng 200g, hệ thống chỉ bù đắp được độ lệch tối đa khoảng 137g. Để cân bằng tải lớn hơn cần tăng khối lượng đối trọng.

Phạm vi cân bằng: Hệ thống chỉ cân bằng theo một trục. Robot thực tế có thể cần cân bằng theo cả hai trục.

5.3. Hướng phát triển

Tăng tốc độ phản hồi: Sử dụng vitme bước lớn hơn (4mm hoặc 8mm) hoặc giảm microstep xuống 1/4 để tăng tốc độ di chuyển.

Tăng khả năng cân bằng: Sử dụng đối trọng nặng hơn (có thể tận dụng pin/acquy của robot) hoặc thiết kế cơ cấu có cánh tay đòn dài hơn.

Mở rộng sang hai trục: Thiết kế thêm cơ cấu di chuyển đối trọng theo trục trước-sau để cân bằng hoàn toàn.

Điều khiển vị trí và tốc độ: Như yêu cầu mở rộng của đề tài, có thể phát triển thêm chế độ điều khiển vị trí tuyệt đối và giới hạn tốc độ chuyển dịch của đối trọng.

Tích hợp vào robot thực tế: Thiết kế phiên bản nhỏ gọn hơn, tối ưu nguồn điện và giao tiếp với hệ thống điều khiển chính của robot.

Tài liệu tham khảo

Tiếng Việt

[1] Phan Xuân Minh, Nguyễn Doãn Phước, *Lý thuyết điều khiển tự động*, Nhà xuất bản Khoa học và Kỹ thuật, 2012.

[2] Nguyễn Phùng Quang, *Matlab & Simulink dành cho kỹ sư điều khiển tự động*, Nhà xuất bản Khoa học và Kỹ thuật, 2015.

Tiếng Anh

[3] K. Ogata, *Modern Control Engineering*, 5th Edition, Prentice Hall, 2010.

[4] N. S. Nise, *Control Systems Engineering*, 7th Edition, Wiley, 2015.

[5] Mike McCauley, "AccelStepper Library for Arduino", *AirSpayce*, 2023, https://www.airspayce.com/mikem/arduino/AccelStepper/.

[6] Olav Kallhovd, "HX711_ADC Library", *GitHub Repository*, 2023, https://github.com/olkal/HX711_ADC.

[7] Arduino, "Arduino Reference", *Arduino Documentation*, 2023, https://www.arduino.cc/reference/en/.

Phụ lục A.
Mã nguồn chương trình điều khiển

Dưới đây là mã nguồn hoàn chỉnh của chương trình điều khiển khung cân bằng trọng tâm:

```
/*
 * DU AN KHUNG CAN BANG - PHIEN BAN HOAN CHINH
 * Tinh nang:
 * 1. Non-blocking: Dong co chay muot, khong bi khung khi doc cam bien.
 * 2. PID Control: Dieu khien vi tri con truot theo sai so trong luong.
 * 3. Deadzone: Vung chet +/- 50g giup dong co nghi ngoi khi da can bang.
 * 4. Quantization: Lam tron gia tri doc ve boi so cua 10g de giam nhieu.
 */

#include <HX711_ADC.h>
#include <AccelStepper.h>
#include <PID_v1.h>

// === CAU HINH HE THONG ===
const int DOUT_PIN_LEFT = 4; const int SCK_PIN_LEFT = 5;
const int DOUT_PIN_PHAI = 6; const int SCK_PIN_PHAI = 7;
const int STEP_PIN = 8;
const int DIR_PIN = 9;
const int HOME_SWITCH_PIN = 10;

// --- THONG SO CO KHI ---
const int MICROSTEP = 16;
const int MOTOR_STEP = 200;
const int PITCH = 2;
const float STEPS_PER_MM = (float)(MOTOR_STEP * MICROSTEP) / PITCH;

// --- GIOI HAN HANH TRINH ---
const long MAX_POS_RIGHT = 110.0 * STEPS_PER_MM;
const long MAX_POS_LEFT = -120.0 * STEPS_PER_MM;

// --- TOC DO & GIA TOC ---
const float MAX_SPEED_PID = 8000.0;
const float MOTOR_ACCEL = 16000.0;
const float HOMING_SPEED_FAST = 8000.0;
const float HOMING_SPEED_SLOW = 4000.0;

// --- CAU HINH CHONG RUNG ---
const float MIN_SPEED_THRESHOLD = 2000.0;
const float HYSTERESIS_GAP = 10.0;
```

```
40
41 // --- CAU HINH PID ---
42 double Kp = 25.0;
43 double Ki = 0.05;
44 double Kd = 0;
45 double Setpoint = 0, Input, Output;
46 PID myPID(&Input, &Output, &Setpoint, Kp, Ki, Kd, DIRECT);
47
48 // --- HIEU CHUAN ---
49 float CALIB_LEFT = -53.13;
50 float CALIB_RIGHT = -55.36;
51 const float ACCEPTABLE_RANGE = 50.0;
52 const float STEP_SIZE = 10.0;
53
54 // === KHOI TAO DOI TUONG ===
55 HX711_ADC LoadCellLeft(DOUT_PIN_LEFT, SCK_PIN_LEFT);
56 HX711_ADC LoadCellRight(DOUT_PIN_PHAI, SCK_PIN_PHAI);
57 AccelStepper stepper(AccelStepper::DRIVER, STEP_PIN, DIR_PIN);
58
59 unsigned long t_pid = 0;
60 unsigned long t_print = 0;
61 float fL = 0, fR = 0, targetSpeed = 0, currentSpeed = 0;
62 boolean isBalancing = false;
63
64 void setup() {
65     Serial.begin(115200);
66     pinMode(HOME_SWITCH_PIN, INPUT_PULLUP);
67
68     // Khoi tao Loadcell
69     LoadCellLeft.begin(); LoadCellRight.begin();
70     LoadCellLeft.start(1000, true);
71     LoadCellRight.start(1000, true);
72     LoadCellLeft.setCalFactor(CALIB_LEFT);
73     LoadCellRight.setCalFactor(CALIB_RIGHT);
74
75     // Khoi tao PID
76     myPID.SetMode(AUTOMATIC);
77     myPID.SetOutputLimits(-MAX_SPEED_PID, MAX_SPEED_PID);
78     myPID.SetSampleTime(20);
79
80     // Khoi tao Dong co
81     stepper.setMaxSpeed(MAX_SPEED_PID);
82     stepper.setAcceleration(MOTOR_ACCEL);
83
84     runHomingSequence();
85 }
86
87 void loop() {
88     // Nhiem vu nen
89     LoadCellLeft.update();
```

```
    LoadCellRight.update();
    stepper.runSpeed();

    // Nhiem vu dieu khien (20ms)
    if (millis() > t_pid + 20) {
        float rawL = LoadCellLeft.getData();
        float rawR = LoadCellRight.getData();
        fL = ceil(rawL / STEP_SIZE) * STEP_SIZE;
        fR = ceil(rawR / STEP_SIZE) * STEP_SIZE;
        Input = fR - fL;

        float startThreshold = ACCEPTABLE_RANGE + HYSTERESIS_GAP;
        float stopThreshold = ACCEPTABLE_RANGE - HYSTERESIS_GAP;

        if (!isBalancing && abs(Input) > startThreshold) {
            isBalancing = true;
            myPID.SetMode(AUTOMATIC);
        }
        else if (isBalancing && abs(Input) < stopThreshold) {
            isBalancing = false;
            Output = 0;
            myPID.SetMode(MANUAL);
        }

        if (isBalancing) {
            myPID.Compute();
            float rawSpeed = Output;
            if (abs(rawSpeed) > 0 && abs(rawSpeed) < MIN_SPEED_THRESHOLD
) {
                targetSpeed = (rawSpeed > 0) ? MIN_SPEED_THRESHOLD : -
MIN_SPEED_THRESHOLD;
            } else {
                targetSpeed = rawSpeed;
            }
        } else {
            targetSpeed = 0;
        }

        long currentPos = stepper.currentPosition();
        if (currentPos <= MAX_POS_LEFT && targetSpeed < 0) targetSpeed =
 0;
        if (currentPos >= MAX_POS_RIGHT && targetSpeed > 0) targetSpeed
= 0;

        stepper.setSpeed(targetSpeed);
        currentSpeed = targetSpeed;
        t_pid = millis();
    }

    // Nhiem vu giam sat (100ms)
```

```
    if (millis() > t_print + 100) {
        Serial.print("L: "); Serial.print(fL, 0);
        Serial.print(" | R: "); Serial.print(fR, 0);
        Serial.print(" | Lech: "); Serial.print(Input, 0);
        Serial.print(" | Spd: "); Serial.print(currentSpeed, 0);
        Serial.println(isBalancing ? " [RUN]" : " [OK]");
        t_print = millis();
    }
}

void runHomingSequence() {
    Serial.println("[HOMING] Bat dau ve Home...");
    stepper.setSpeed(HOMING_SPEED_FAST);
    while (digitalRead(HOME_SWITCH_PIN) == HIGH) { stepper.runSpeed(); }
    stepper.stop();
    stepper.setCurrentPosition(MAX_POS_RIGHT);

    stepper.moveTo(MAX_POS_RIGHT - (5.0 * STEPS_PER_MM));
    while (stepper.distanceToGo() != 0) { stepper.run(); }

    stepper.setSpeed(HOMING_SPEED_SLOW);
    while (digitalRead(HOME_SWITCH_PIN) == HIGH) { stepper.runSpeed(); }
    stepper.stop();
    stepper.setCurrentPosition(MAX_POS_RIGHT);

    stepper.moveTo(0);
    while (stepper.distanceToGo() != 0) { stepper.run(); }

    delay(5000); // Cho on dinh
    Serial.println("[HOMING] Hoan tat.");
}
```

Listing A.1: Chương trình điều khiển khung cân bằng trọng tâm