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

Trong những năm gần đây, robot di động (Mobile Robot) đã trở thành một trong những lĩnh vực nghiên cứu và ứng dụng phát triển mạnh mẽ nhất trong ngành tự động hóa. Từ các robot vận chuyển hàng hóa trong nhà máy, robot phục vụ trong nhà hàng, đến các robot thám hiểm địa hình phức tạp – tất cả đều đòi hỏi khả năng di chuyển ổn định và thích ứng với các điều kiện tải trọng thay đổi.

Một thách thức kỹ thuật quan trọng trong thiết kế robot di động là vấn đề **cân bằng trọng tâm**. Khi robot mang theo tải trọng hoặc được trang bị cánh tay thao tác (manipulator), trọng tâm của hệ thống sẽ thay đổi theo vị trí và khối lượng của tải. Sự dịch chuyển trọng tâm này gây ra nhiều hệ quả tiêu cực:

- **Giảm ổn định động học:** Robot dễ bị lật hoặc mất cân bằng khi di chuyển trên địa hình không bằng phẳng, đặc biệt khi tải lệch về một phía.
- **Tăng tải không đều lên các bánh xe:** Dẫn đến mài mòn không đồng đều, giảm tuổi thọ cơ cấu truyền động và ảnh hưởng đến độ chính xác điều khiển quỹ đạo.
- **Tăng tiêu hao năng lượng:** Các động cơ dẫn động phải bù đắp momen do tải lệch gây ra, làm giảm hiệu suất và thời gian hoạt động của robot.
- **Ảnh hưởng đến độ chính xác thao tác:** Với robot có cánh tay, sự mất cân bằng của đế ảnh hưởng trực tiếp đến độ chính xác định vị của end-effector.

Giải pháp truyền thống cho vấn đề này thường là thiết kế đế robot với trọng tâm thấp và phân bố tải đối xứng. Tuy nhiên, cách tiếp cận này không linh hoạt khi tải trọng thay đổi trong quá trình vận hành. Một giải pháp tiên tiến hơn là sử dụng **hệ thống cân bằng trọng tâm chủ động** (Active Center of Gravity Balancing System), trong đó một cơ cấu chấp hành sẽ tự động di chuyển khối đối trọng để bù đắp sự thay đổi trọng tâm.

Hệ thống cân bằng trọng tâm chủ động đòi hỏi sự kết hợp của nhiều thành phần: cảm biến đo lường độ lệch, cơ cấu chấp hành di chuyển đối trọng, và thuật toán điều khiển để xác định vị trí đối trọng tối ưu. Đây là một bài toán điều khiển vòng kín điển hình, phù hợp để áp dụng các kiến thức về **lý thuyết điều khiển tự động**, **cảm biến và đo lường**, cũng như **hệ thống nhúng thời gian thực**.

Xuất phát từ nhu cầu thực tiễn và tính ứng dụng cao của vấn đề, đề tài *"Xây dựng khung đế robot có khả năng cân bằng trọng tâm"* được lựa chọn làm khóa luận tốt nghiệp. Đề tài hướng đến việc nghiên cứu, thiết kế và chế tạo một khung đế có khả năng tự động phát hiện và bù đắp độ lệch trọng tâm, tạo nền tảng cho việc phát triển các robot di động ổn định hơn trong tương lai.

1.2. Mục tiêu nghiên cứu

Mục tiêu tổng quát của khóa luận là thiết kế và chế tạo một khung đế robot di động có khả năng tự động cân bằng trọng tâm khi tải trọng thay đổi. Để đạt được mục tiêu này, các mục tiêu cụ thể được xác định như sau:

1.2.1. Mục tiêu về nghiên cứu lý thuyết

Nghiên cứu nguyên lý cân bằng trọng tâm dựa trên lý thuyết momen lực và điều kiện cân bằng tĩnh. Tìm hiểu các phương pháp đo lường độ lệch trọng tâm, trong đó tập trung vào việc sử dụng cảm biến lực (Loadcell) để xác định sự chênh lệch tải trọng giữa các điểm tựa.

Nghiên cứu thuật toán điều khiển PID (Proportional-Integral-Derivative) và phương pháp tinh chỉnh tham số phù hợp với đặc tính của hệ thống cơ điện tử. Đặc biệt, nghiên cứu các kỹ thuật xử lý vùng chết (Deadzone), độ trễ (Hysteresis) để tránh hiện tượng dao động quanh điểm cân bằng.

Nghiên cứu kiến trúc lập trình thời gian thực cho hệ thống nhúng, đảm bảo khả năng đáp ứng nhanh và ổn định của hệ thống điều khiển.

1.2.2. Mục tiêu về thiết kế và chế tạo

Thiết kế khung cơ khí có kích thước phù hợp với robot di động cỡ nhỏ và vừa (khoảng 35×35cm), đảm bảo độ cứng vững và khả năng tích hợp các thành phần điện tử.

Thiết kế cơ cấu di chuyển đối trọng sử dụng động cơ bước kết hợp truyền động vitme, đảm bảo độ chính xác vị trí và khả năng chịu tải.

Thiết kế mạch điện tử điều khiển bao gồm: mạch đọc tín hiệu từ cảm biến lực, mạch điều khiển động cơ bước, và giao tiếp với vi điều khiển.

Chế tạo và lắp ráp hoàn chỉnh mô hình khung đế cân bằng.

1.2.3. Mục tiêu về phần mềm điều khiển

Xây dựng chương trình điều khiển trên nền tảng vi điều khiển Arduino, tích hợp thuật toán PID để tự động điều chỉnh vị trí đối trọng.

Đảm bảo chương trình hoạt động theo kiến trúc Non-blocking (không chặn), cho phép động cơ bước vận hành mượt mà ở tốc độ cao mà không bị ảnh hưởng bởi các tác vụ đọc cảm biến hay giao tiếp.

Phát triển giao diện giám sát thông qua Serial Monitor để theo dõi các thông số hoạt động và hỗ trợ quá trình tinh chỉnh hệ thống.

1.2.4. Mục tiêu về thử nghiệm và đánh giá

Tiến hành thử nghiệm hệ thống với các kịch bản tải trọng khác nhau. Đánh giá các chỉ tiêu: thời gian đáp ứng, độ chính xác cân bằng, độ ổn định, và phạm vi tải trọng có thể bù đắp.

So sánh kết quả thực nghiệm với mục tiêu thiết kế, phân tích các hạn chế và đề xuất hướng cải tiến.

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 các thành phần chính của hệ thống cân bằng trọng tâm:

Cảm biến lực Loadcell và module HX711: Loadcell là cảm biến đo lực dựa trên nguyên lý điện trở biến dạng. Module HX711 là bộ chuyển đổi ADC 24-bit chuyên dụng cho loadcell, có khả năng đọc tín hiệu với độ phân giải cao. Trong dự án này, thư viện HX711_ADC được sử dụng với ưu điểm hỗ trợ chế độ đọc Non-blocking, cho phép vi điều khiển thực hiện các tác vụ khác trong khi chờ dữ liệu từ cảm biến.

Động cơ bước và driver TB6600: Động cơ bước Nema 17 được sử dụng làm cơ cấu chấp hành, với đặc điểm điều khiển vị trí chính xác theo vòng hở. Driver TB6600 hỗ trợ điều khiển vi bước (microstepping) với các mức 1/2, 1/4, 1/8, 1/16, cho phép tăng độ phân giải và giảm rung động.

Vi điều khiển Arduino: Nền tảng Arduino (chip ATmega328P, tần số 16MHz) được chọn làm bộ xử lý trung tâm do tính phổ biến, dễ lập trình, và có nhiều thư viện hỗ trợ. Thư viện AccelStepper được sử dụng để điều khiển động cơ bước với khả năng điều chỉnh tốc độ và gia tốc mượt mà.

Thuật toán điều khiển PID: Bộ điều khiển PID là thuật toán điều khiển vòng kín phổ biến trong công nghiệp, phù hợp cho các hệ thống yêu cầu độ chính xác và ổn định cao. Thư viện PID_v1 cho Arduino được sử dụng để triển khai thuật toán.

Kỹ thuật lập trình thời gian thực: Nghiên cứu các vấn đề về độ trễ (latency), blocking I/O, và các giải pháp Non-blocking để đảm bảo hệ thống đáp ứng các ràng buộc thời gian thực.

1.3.2. Phạm vi nghiên cứu

Khóa luận tập trung vào các giới hạn sau:

Về không gian: Hệ thống cân bằng trọng tâm theo **một trục** (trái-phải). Việc mở rộng sang hai trục được đề cập như hướng phát triển.

Về kích thước: Khung đế có kích thước 35×35cm, phù hợp với các robot di động cỡ nhỏ và vừa. Hành trình di chuyển đối trọng: 120mm về bên trái, 110mm về bên phải (tính từ vị trí tâm).

Về tải trọng: Cảm biến loadcell có tải trọng định mức 50kg mỗi cụm. Khối lượng đối trọng ban đầu là 200g, có thể điều chỉnh tùy theo yêu cầu cân bằng.

Về điều khiển: Sử dụng điều khiển vòng hở cho động cơ bước (không có encoder phản hồi vị trí). Thuật toán PID với vùng chết ±50g – tức là hệ thống chấp nhận độ lệch trong phạm vi này mà không cần điều chỉnh.

Về hiệu năng: Tốc độ tối đa của động cơ bước bị giới hạn bởi khả năng xử lý của vi điều khiển và đặc tính của thư viện AccelStepper (khoảng 4000 bước/giây trong điều kiện tối ưu).

1.4. Phương pháp nghiên cứu

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

1.4.1. Phương pháp nghiên cứu lý thuyết

Thu thập và phân tích các tài liệu về: cơ học lý thuyết (momen lực, điều kiện cân bằng), lý thuyết điều khiển tự động (bộ điều khiển PID, ổn định hệ thống), kỹ thuật cảm biến (nguyên lý loadcell, xử lý tín hiệu), và lập trình hệ thống nhúng (kiến trúc real-time, xử lý ngắt).

Tổng hợp kiến thức từ các nghiên cứu trước đó về điều khiển động cơ bước, đặc biệt là các vấn đề về hiệu năng thời gian thực khi kết hợp nhiều tác vụ (đọc cảm biến, điều khiển động cơ, giao tiếp Serial).

1.4.2. Phương pháp mô hình hóa và thiết kế

Xây dựng mô hình toán học của hệ thống cân bằng dựa trên nguyên lý momen lực. Từ đó xác định mối quan hệ giữa độ lệch tải trọng và vị trí đối trọng cần thiết để cân bằng.

Thiết kế cơ khí sử dụng phương pháp thiết kế mô-đun, cho phép dễ dàng điều chỉnh và thay thế các thành phần. Thiết kế mạch điện theo sơ đồ khối, xác định rõ chức năng và giao tiếp của từng module.

Thiết kế phần mềm theo kiến trúc phân tầng: tầng phần cứng (Hardware Abstraction Layer), tầng điều khiển (Control Layer), và tầng ứng dụng (Application Layer).

1.4.3. Phương pháp thực nghiệm

Chế tạo mô hình thực tế dựa trên thiết kế đã xây dựng. Tiến hành hiệu chuẩn (calibration) các cảm biến loadcell để đảm bảo độ chính xác đo lường.

Thực hiện các bài test với nhiều kịch bản tải trọng khác nhau: tải tĩnh (đặt vật nặng cố định), tải động (di chuyển vật nặng trong quá trình hoạt động), và tải biến thiên (thay đổi khối lượng tải).

Thu thập dữ liệu thông qua Serial Monitor, bao gồm: giá trị đọc từ loadcell, sai số (Input), tín hiệu điều khiển (Output), và tốc độ động cơ. Phân tích dữ liệu để đánh giá hiệu quả của thuật toán điều khiển.

1.4.4. Phương pháp phân tích và đánh giá

So sánh kết quả thực nghiệm với các chỉ tiêu thiết kế ban đầu. Phân tích nguyên nhân của các sai lệch (nếu có) và đề xuất giải pháp khắc phục.

Đánh giá định lượng các thông số: thời gian đáp ứng (từ khi có tải lệch đến khi đạt cân bằng), độ chính xác cân bằng (sai số còn lại sau khi ổn định), và phạm vi hoạt động (độ lệch tối đa có thể bù đắp).

Phân tích các yếu tố ảnh hưởng đến hiệu năng hệ thống, bao gồm: tham số PID, tốc độ lấy mẫu, độ phân giải microstep, và các vấn đề về lập trình thời gian thực.

1.5. Ý nghĩa khoa học và thực tiễn

1.5.1. Ý nghĩa khoa học

Khóa luận đóng góp vào việc nghiên cứu ứng dụng lý thuyết điều khiển tự động trong bài toán cân bằng trọng tâm cho robot di động. Các kết quả nghiên cứu về ảnh

hưởng của kiến trúc phần mềm (blocking vs non-blocking) đến hiệu năng điều khiển động cơ bước có giá trị tham khảo cho các nghiên cứu liên quan.

Việc phân tích chi tiết các vấn đề thời gian thực trong hệ thống nhúng – từ độ trễ của giao tiếp Serial đến ảnh hưởng của tần số lấy mẫu – cung cấp cơ sở lý thuyết và thực nghiệm cho việc thiết kế các hệ thống điều khiển tương tự.

1.5.2. Ý nghĩa thực tiễn

Sản phẩm của khóa luận là một khung đế robot có khả năng tự cân bằng, có thể được tích hợp vào các robot di động thực tế để cải thiện độ ổn định khi vận hành.

Các kinh nghiệm thiết kế và chế tạo (lựa chọn linh kiện, giải quyết các vấn đề kỹ thuật, tinh chỉnh hệ thống) được tài liệu hóa trong khóa luận, có thể làm tài liệu tham khảo cho các dự án tương tự.

Mã nguồn chương trình điều khiển được cung cấp đầy đủ trong phụ lục, có thể được sử dụng lại hoặc phát triển thêm cho các ứng dụng khác.

1.6. Bố cục khóa luận

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

Chương 1: Tổng quan – Trình bày bối cảnh và lý do chọn đề tài, xác định mục tiêu nghiên cứu, đối tượng và phạm vi nghiên cứu, các phương pháp nghiên cứu được sử dụng, cũng như ý nghĩa khoa học và thực tiễn của đề tài.

Chương 2: Cơ sở lý thuyết – Trình bày các kiến thức nền tảng về nguyên lý cân bằng trọng tâm và momen lực, cảm biến lực Loadcell và module HX711, động cơ bước và kỹ thuật vi bước, vi điều khiển Arduino, thuật toán điều khiển PID, và các vấn đề lập trình thời gian thực cho hệ thống nhúng.

Chương 3: Thiết kế hệ thống – Mô tả chi tiết quá trình thiết kế gồm: phân tích yêu cầu, thiết kế cơ khí (khung, cơ cấu vitme-thanh trượt, bố trí loadcell), thiết kế mạch điện (sơ đồ khối, kết nối phần cứng), và thiết kế phần mềm (cấu trúc chương trình, quy trình Homing, logic điều khiển PID với Deadzone và Hysteresis).

Chương 4: Thực nghiệm và đánh giá – Trình bày môi trường và điều kiện thử nghiệm, quá trình tinh chỉnh tham số PID, kết quả thực nghiệm với các kịch bản tải trọng khác nhau, phân tích hiệu năng hệ thống, và đánh giá so với mục tiêu đề ra.

Chương 5: Kết luận và hướng phát triển – Tổng kết các kết quả đạt được của khóa luận, nêu rõ các hạn chế còn tồn tại, và đề xuất các hướng phát triển tiếp theo bao gồm: tăng tốc độ phản hồi, mở rộng cân bằng hai trục, và tích hợp điều khiển vị trí/tốc độ.

Chương 2.
Cơ sở lý thuyết

Chương này trình bày các kiến thức nền tảng cần thiết cho việc thiết kế và xây dựng hệ thống cân bằng trọng tâm tự động. Các nội dung được trình bày bao gồm: lý thuyết cơ học về cân bằng và momen lực, nguyên lý hoạt động của cảm biến lực và module chuyển đổi tín hiệu, đặc tính của động cơ bước và kỹ thuật điều khiển vi bước, nền tảng vi điều khiển Arduino, thuật toán điều khiển PID, và đặc biệt là các vấn đề về lập trình thời gian thực trong hệ thống nhúng.

2.1. Lý thuyết cân bằng trọng tâm và momen lực

2.1.1. Khái niệm trọng tâm

Trọng tâm (Center of Gravity - CoG) của một vật thể hoặc hệ vật là điểm mà tại đó toàn bộ trọng lượng của hệ có thể được coi như tập trung. Đối với một hệ thống gồm n vật thể rời rạc, vị trí trọng tâm theo trục x được xác định bởi công thức:

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

trong đó x_G là tọa độ trọng tâm của hệ, m_i là khối lượng của vật thể thứ i, x_i là tọa độ của vật thể thứ i theo trục x, và $M = \sum m_i$ là tổng khối lượng của hệ.

Công thức tương tự được áp dụng cho các trục y và z trong không gian ba chiều. Trong phạm vi khóa luận này, bài toán được đơn giản hóa thành cân bằng theo một trục (trái-phải), tương ứng với việc xét trọng tâm trên trục x.

2.1.2. Momen lực và điều kiện cân bằng tĩnh

Momen lực (Torque) là đại lượng vật lý đặc trưng cho tác dụng làm quay của một lực đối với một trục quay. Độ lớn của momen lực được tính theo công thức:

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

trong đó M là momen lực (đơn vị N.m hoặc N.mm), F là lực tác dụng (N), và d là cánh tay đòn – khoảng cách vuông góc từ đường tác dụng của lực đến trục quay (m hoặc mm).

Quy ước dấu: Momen làm vật quay theo chiều kim đồng hồ thường được quy ước là âm, ngược chiều kim đồng hồ là dương (hoặc ngược lại, tùy theo hệ quy chiếu được chọn).

Điều kiện cân bằng tĩnh của một vật rắn yêu cầu tổng các momen lực tác dụng lên vật đối với một trục quay bất kỳ phải bằng không:

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

Điều kiện này có nghĩa là tổng các momen theo chiều dương phải cân bằng với tổng các momen theo chiều âm.

2.1.3. Áp dụng nguyên lý momen vào bài toán cân bằng khung robot

Xét mô hình khung đế robot như một thanh cứng được đỡ tại điểm tựa ở giữa (tâm khung). Khi có tải trọng đặt lệch về một phía, momen do tải gây ra sẽ làm khung mất cân bằng. Để khôi phục cân bằng, cần di chuyển một khối đối trọng về phía đối diện sao cho momen do đối trọng tạo ra bù đắp momen do tải.

Điều kiện cân bằng được viết như sau:

$$m_{slider} \times d_{slider} = m_{load} \times d_{load} \tag{4}$$

trong đó m_{slider} là khối lượng của khối đối trọng (kg hoặc g), d_{slider} là khoảng cách từ đối trọng đến tâm khung (mm), m_{load} là độ chênh lệch khối lượng tải giữa hai bên (kg hoặc g), và d_{load} là khoảng cách từ điểm đặt tải đến tâm khung (mm).

Từ phương trình (4), có thể suy ra vị trí cần thiết của đối trọng:

$$d_{slider} = \frac{m_{load} \times d_{load}}{m_{slider}} \tag{5}$$

Phương trình này cho thấy một hạn chế quan trọng: với khối lượng đối trọng cố định m_{slider} và hành trình giới hạn $d_{slider,max}$, độ lệch tải tối đa có thể bù đắp là:

$$m_{load,max} = \frac{m_{slider} \times d_{slider,max}}{d_{load}} \tag{6}$$

Ví dụ cụ thể từ hệ thống được thiết kế: với $m_{slider} = 200\text{g}$, $d_{slider,max} = 120\text{mm}$, và $d_{load} = 175\text{mm}$ (một nửa chiều rộng khung 35cm), độ lệch tải tối đa có thể bù đắp là:

$$m_{load,max} = \frac{200 \times 120}{175} \approx 137 \text{ g} \tag{7}$$

Kết quả này cho thấy để tăng khả năng cân bằng, có thể: (1) tăng khối lượng đối trọng, (2) tăng hành trình di chuyển, hoặc (3) bố trí tải gần tâm hơn.

2.2. Cảm biến lực Loadcell và module HX711

2.2.1. Cấu tạo và nguyên lý hoạt động của Loadcell

Loadcell là cảm biến chuyển đổi lực cơ học thành tín hiệu điện, hoạt động dựa trên hiệu ứng điện trở biến dạng (Piezoresistive effect). Cấu tạo cơ bản của loadcell gồm:

Thân đàn hồi (Elastic element): Thường làm từ thép hợp kim hoặc nhôm, được gia công với hình dạng đặc biệt để biến dạng theo hướng xác định khi chịu lực.

Điện trở biến dạng (Strain gauge): Là các dải điện trở mỏng được dán trực tiếp lên bề mặt thân đàn hồi. Khi thân đàn hồi biến dạng, điện trở của strain gauge thay đổi theo quan hệ:

$$\frac{\Delta R}{R} = GF \times \varepsilon \tag{8}$$

trong đó $\Delta R/R$ là tỷ lệ thay đổi điện trở, GF là hệ số gauge (Gauge Factor, thường từ 2 đến 4 cho strain gauge kim loại), và ε là biến dạng tương đối của vật liệu.

Cầu Wheatstone: Bốn strain gauge được mắc theo cấu hình cầu Wheatstone để chuyển đổi sự thay đổi điện trở thành sự thay đổi điện áp. Cấu hình cầu đầy đủ (Full-bridge) cho độ nhạy cao nhất và khả năng bù nhiệt tốt.

Điện áp đầu ra của cầu Wheatstone:

$$V_{out} = V_{exc} \times \frac{\Delta R}{R} \tag{9}$$

trong đó V_{exc} là điện áp kích thích (thường 5V hoặc 10V).

2.2.2. Loadcell 3 dây và cách ghép thành cầu đầy đủ

Trong dự án này, loadcell 3 dây (half-bridge) được sử dụng. Loại loadcell này chỉ chứa hai strain gauge, cần ghép hai loadcell lại để tạo thành cầu Wheatstone đầy đủ.

Cách đấu nối hai loadcell 3 dây thành một cầu cho module HX711:

– Loadcell 1: Dây trắng vào E-, dây đen vào E+

– Loadcell 2: Dây trắng vào A-, dây đen vào E+

– Kết nối chung: Nối hai dây đỏ của cả hai loadcell lại với nhau và đưa vào chân A+

Với cách ghép này, hệ thống sử dụng 4 loadcell 50kg (2 cặp), tạo thành 2 cụm cảm biến độc lập cho bên trái và bên phải, mỗi cụm có khả năng chịu tải tổng cộng 100kg.

2.2.3. Module HX711 và giao thức truyền thông

HX711 là IC chuyển đổi tương tự sang số (ADC) 24-bit được thiết kế chuyên dụng cho các ứng dụng cân điện tử. Các đặc tính kỹ thuật chính:

- Độ phân giải: 24-bit (tương đương khoảng 16.7 triệu mức lượng tử hóa)
- Tốc độ lấy mẫu: 10 SPS (Samples Per Second) hoặc 80 SPS, tùy thuộc vào mức logic của chân RATE
- Bộ khuếch đại tích hợp: Độ lợi (Gain) có thể chọn 32, 64, hoặc 128
- Giao tiếp: Giao thức nối tiếp đồng bộ 2 dây (DOUT và SCK)
- Nguồn cấp: 2.6V đến 5.5V

Giao thức truyền thông của HX711 hoạt động như sau: Vi điều khiển tạo xung clock trên chân SCK và đọc dữ liệu từ chân DOUT. Mỗi lần đọc, HX711 truyền 24 bit dữ liệu ADC, sau đó 1-3 xung clock bổ sung để chọn kênh và độ lợi cho lần đọc tiếp theo.

2.2.4. Thư viện HX711_ADC và cơ chế Non-blocking

Một trong những quyết định kỹ thuật quan trọng của dự án là sử dụng thư viện `HX711_ADC` của tác giả Olav Kallhovd thay vì thư viện `HX711` chuẩn (của Bogde). Sự khác biệt cốt lõi nằm ở cơ chế đọc dữ liệu:

Thư viện HX711 chuẩn (Blocking):

- Hàm chính: `scale.get_units(n)`
- Cơ chế: Khi gọi hàm, vi điều khiển **dừng lại và chờ** cho đến khi HX711 hoàn thành chuyển đổi
- Thời gian chờ: Ở tốc độ 10Hz, mỗi lần đọc mất 100ms. Nếu lấy trung bình 5 mẫu (`get_units(5)`), thời gian chờ lên đến 500ms
- Hậu quả: Trong thời gian chờ, các lệnh điều khiển động cơ không được thực thi, gây hiện tượng động cơ chạy giật cục hoặc dừng hẳn

Thư viện HX711_ADC (Non-blocking):

- Hàm chính: `LoadCell.update()` và `LoadCell.getData()`
- Cơ chế: Hàm `update()` chỉ kiểm tra xem có dữ liệu mới hay không. Nếu chưa có, hàm trả về ngay lập tức (tốn vài micro-giây). Nếu có dữ liệu mới, hàm đọc và lưu vào biến nội bộ

– Hàm `getData()` lấy giá trị từ biến nội bộ ra một cách tức thì, không cần chờ đợi

– Lợi ích: Vi điều khiển có thể thực hiện các tác vụ khác (như điều khiển động cơ) trong khi chờ dữ liệu từ cảm biến

Sự khác biệt này có ý nghĩa quyết định đối với hiệu năng hệ thống. Với kiến trúc Non-blocking, hàm `stepper.runSpeed()` có thể được gọi liên tục trong vòng lặp chính mà không bị gián đoạn bởi việc đọc cảm biến, đảm bảo động cơ bước hoạt động mượt mà ở tốc độ cao.

2.3. Động cơ bước và driver TB6600

2.3.1. Nguyên lý hoạt động của 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 góc cố định. Khác với động cơ DC thông thường (có tốc độ phụ thuộc vào điện áp), vị trí và tốc độ của động cơ bước được xác định hoàn toàn bởi số lượng và tần số của các xung điều khiển.

Động cơ bước lai (Hybrid Stepper Motor) – loại phổ biến nhất trong các ứng dụng công nghiệp và được sử dụng trong dự án này (Nema 17) – kết hợp ưu điểm của động cơ bước nam châm vĩnh cửu và động cơ bước biến từ trở. Cấu tạo gồm:

Stator: Chứa các cuộn dây được quấn theo cặp (Phase A và Phase B cho động cơ lưỡng cực). Khi có dòng điện chạy qua, các cuộn dây tạo ra từ trường.

Rotor: Gồm lõi nam châm vĩnh cửu được bọc bởi hai đĩa răng lệch pha nhau. Kết hợp với răng trên stator, cấu trúc này cho phép góc bước nhỏ (thường 1.8°, tương đương 200 bước/vòng).

Nguyên lý hoạt động: Khi dòng điện trong các cuộn dây thay đổi theo trình tự xác định, từ trường stator quay theo, kéo rotor quay theo từng bước. Mỗi bước tương ứng với một góc cố định:

$$\theta_{step} = \frac{360}{N_{steps}} \tag{10}$$

trong đó N_{steps} là số bước trên một vòng quay (200 cho động cơ 1.8°).

2.3.2. Chế độ vi bước (Microstepping) và ảnh hưởng đến hiệu năng

Vi bước là kỹ thuật điều khiển dòng điện trong các cuộn dây theo dạng sóng sin/cosin thay vì dạng bước vuông. Điều này cho phép chia nhỏ mỗi bước đầy đủ thành nhiều bước nhỏ hơn.

Số bước trên mỗi vòng quay khi sử dụng vi bước:

$$N_{micro} = N_{full} \times k \tag{11}$$

trong đó $N_{full} = 200$ bước/vòng (cho động cơ 1.8°) và k là hệ số vi bước (1, 2, 4, 8, 16, 32...).

Với chế độ 1/16 microstep được sử dụng trong dự án:

$$N_{micro} = 200 \times 16 = 3200 \text{ bước/vòng} \tag{12}$$

Ưu điểm của vi bước:

– Chuyển động mượt mà hơn, giảm rung động và tiếng ồn

– Độ phân giải vị trí cao hơn

– Giảm hiện tượng cộng hưởng ở một số dải tốc độ

Nhược điểm và thách thức:

– Yêu cầu tần số xung điều khiển cao hơn để đạt cùng tốc độ quay

– Tăng tải xử lý cho vi điều khiển

– Momen xoắn giảm nhẹ so với chế độ full-step

Phân tích ràng buộc thời gian thực:

Xét yêu cầu tốc độ 300 RPM (5 vòng/giây):

– Ở chế độ Full-step: Tần số xung = $200 \times 5 = 1000$ Hz. Khoảng cách giữa các bước = 1000 μs

– Ở chế độ 1/16 Microstep: Tần số xung = $3200 \times 5 = 16000$ Hz. Khoảng cách giữa các bước = 62.5 μs

Con số 62.5 μs là "ngân sách thời gian" (time budget) tối đa mà vi điều khiển có để hoàn thành mọi tác vụ giữa hai lần tạo xung. Nếu bất kỳ tác vụ nào (đọc cảm biến, tính toán PID, giao tiếp Serial) chiếm thời gian lớn hơn ngưỡng này, hệ thống sẽ vi phạm ràng buộc thời gian thực và động cơ sẽ mất bước.

2.3.3. Driver TB6600 và cấu hình điều khiển

TB6600 là driver công suất cao cho động cơ bước, được thiết kế để điều khiển các động cơ Nema 17 và Nema 23. Thông số kỹ thuật:

– Điện áp làm việc: 9-42V DC

– Dòng điện tối đa: 4A/pha (có thể điều chỉnh bằng DIP switch)

– Hỗ trợ vi bước: Full, 1/2, 1/4, 1/8, 1/16, 1/32

– Tín hiệu điều khiển: PUL+ (xung bước), DIR+ (chiều quay), ENA+ (kích hoạt)

– Cách ly quang học giữa tín hiệu điều khiển và mạch công suất

Giao thức điều khiển đơn giản: Mỗi xung trên chân PUL làm động cơ quay một vi bước. Mức logic trên chân DIR quyết định chiều quay (HIGH = thuận, LOW = nghịch). Chân ENA thường được bỏ trống hoặc nối với mức logic phù hợp để luôn kích hoạt driver.

2.3.4. Thư viện AccelStepper và cơ chế Polling

Thư viện AccelStepper của Mike McCauley là thư viện phổ biến nhất cho điều khiển động cơ bước trên Arduino. Thư viện hoạt động theo cơ chế **Polling** (hỏi vòng), không phải Interrupt-driven (ngắt).

Hàm cốt lõi `runSpeed()` hoạt động như sau (mã giả):

```
boolean AccelStepper::runSpeed() {
    unsigned long time = micros();
    if (time >= _nextStepTime) {
        step(_direction);  // Thuc hien buoc
        _nextStepTime = time + _stepInterval;
        return true;
    }
    return false;  // Chua den luc, khong lam gi
}
```

Thiết kế này đặt ra yêu cầu quan trọng: **Hàm `runSpeed()` phải được gọi với tần số cao hơn nhiều so với tần số bước của động cơ**. Nếu động cơ cần bước mỗi 62.5 μs (ở 1/16 microstep, 300 RPM), thì `runSpeed()` cần được gọi ít nhất mỗi 30 μs để đảm bảo độ chính xác thời gian.

Thư viện cũng cung cấp các hàm khác như `run()` (có gia tốc), `runToPosition()` và `runToNewPosition()` (chạy đến vị trí đích). Tuy nhiên, hai hàm sau là **blocking** – chúng chứa vòng lặp nội bộ và không trả về cho đến khi động cơ đạt vị trí đích. Trong dự án này, `runSpeed()` được sử dụng trong vòng lặp chính để đảm bảo tính non-blocking.

2.4. Vi điều khiển Arduino

2.4.1. Kiến trúc phần cứng ATmega328P

Arduino Uno/Nano sử dụng vi điều khiển ATmega328P của Atmel (nay thuộc Microchip). Đây là vi điều khiển 8-bit kiến trúc AVR với các thông số:

– Tần số xung nhịp: 16 MHz (chu kỳ lệnh 62.5 ns)

– Bộ nhớ Flash (chương trình): 32 KB

– Bộ nhớ SRAM (dữ liệu): 2 KB

– Bộ nhớ EEPROM: 1 KB

– Chân I/O số: 14 (trong đó 6 chân hỗ trợ PWM)

– Chân đầu vào analog: 6 (ADC 10-bit)

– Giao tiếp: UART, SPI, I2C

– Timer: 2 timer 8-bit, 1 timer 16-bit

Với tần số 16 MHz, vi điều khiển có thể thực thi khoảng 16 triệu lệnh đơn giản mỗi giây. Tuy nhiên, các thao tác phức tạp như phép tính dấu phẩy động, giao tiếp Serial, hay đọc ADC tiêu tốn nhiều chu kỳ xung nhịp hơn.

2.4.2. Mô hình lập trình Arduino

Arduino sử dụng mô hình lập trình đơn giản với hai hàm chính:

`setup()`: Được gọi một lần khi khởi động, dùng để khởi tạo các thành phần phần cứng và phần mềm.

`loop()`: Được gọi lặp đi lặp lại vô hạn sau khi `setup()` hoàn thành. Đây là nơi chứa logic chính của chương trình.

Mô hình này tương đương với kiến trúc **Superloop** trong lập trình hệ thống nhúng – một vòng lặp vô hạn thực thi tuần tự các tác vụ. Ưu điểm là đơn giản, dễ hiểu. Nhược điểm là khó đảm bảo thời gian đáp ứng cho các tác vụ quan trọng nếu có tác vụ blocking trong vòng lặp.

2.4.3. Vấn đề Blocking I/O trong giao tiếp Serial

Giao tiếp Serial (UART) là nguồn gây blocking phổ biến nhất trong các chương trình Arduino. Phân tích chi tiết cơ chế hoạt động:

Bộ đệm truyền (TX Buffer):

Arduino sử dụng bộ đệm vòng (ring buffer) với kích thước mặc định 64 byte để chứa dữ liệu chờ gửi. Khi gọi `Serial.print()`, dữ liệu được sao chép vào buffer. Nếu buffer còn trống, hàm trả về ngay lập tức (non-blocking). Một trình phục vụ ngắt (ISR) chạy ngầm sẽ lấy từng byte từ buffer và gửi ra chân TX.

Hiện tượng Buffer Overflow:

Vấn đề xảy ra khi tốc độ ghi dữ liệu vào buffer cao hơn tốc độ gửi ra ngoài. Tốc độ gửi bị giới hạn bởi Baud Rate. Ở 9600 baud, thời gian gửi 1 byte:

$$t_{byte} = \frac{10 \text{ bits}}{9600 \text{ bps}} \approx 1.04 \text{ ms} \tag{13}$$

(10 bits = 1 start bit + 8 data bits + 1 stop bit)

Khi buffer đầy (64 byte), hàm `Serial.write()` chuyển sang chế độ blocking – chờ đợi cho đến khi có chỗ trống trong buffer. Trong thời gian chờ, CPU không thể thực hiện các tác vụ khác.

Tính toán thiệt hại thời gian:

Giả sử chương trình in dòng lệnh 25 ký tự mỗi vòng lặp, và buffer đã đầy. Thời gian blocking:

$$T_{delay} \approx 25 \times 1.04 \text{ ms} = 26 \text{ ms} \tag{14}$$

Trong 26 ms này:

– Hàm `runSpeed()` không được gọi

– Động cơ (yêu cầu xung mỗi 62.5 μs) bỏ lỡ: $\frac{26000}{62.5} \approx 416$ xung

– Tốc độ thực tế giảm từ 16000 bước/s xuống còn khoảng 38 bước/s

Đây là lý do tại sao việc in Serial không kiểm soát có thể làm động cơ bước "chậm như bị mắc kẹt" – hiện tượng được ghi nhận trong quá trình phát triển hệ thống.

2.5. Thuật toán điều khiển PID

2.5.1. Mô hình toán học của bộ điều khiển 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 $u(t)$ dựa trên sai số $e(t)$ giữa giá trị đặt (Setpoint) và giá trị thực tế (Process Variable):

$$u(t) = K_p \cdot e(t) + K_i \cdot \int_0^t e(\tau)\, d\tau + K_d \cdot \frac{de(t)}{dt} \tag{15}$$

trong đó:

– $u(t)$: Tín hiệu điều khiển (output)

– $e(t) = SP - PV$: Sai số (error)

– K_p: Hệ số khuếch đại tỷ lệ (Proportional gain)

– K_i: Hệ số khuếch đại tích phân (Integral gain)

– K_d: Hệ số khuếch đại vi phân (Derivative gain)

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:

$$u_P = K_p \cdot e(t) \tag{16}$$

K_p càng lớn, hệ thống phản ứng càng nhanh với sai số. Tuy nhiên, nếu K_p quá lớn, hệ thống có thể dao động hoặc mất ổn định. Thành phần P đơn thuần không thể triệt tiêu hoàn toàn sai số xác lập (steady-state error).

Thành phần tích phân (I):

Tích lũy sai số theo thời gian:

$$u_I = K_i \cdot \int_0^t e(\tau)\, d\tau \tag{17}$$

Thành phần này giúp triệt tiêu sai số xác lập bằng cách tích lũy các sai số nhỏ theo thời gian. Tuy nhiên, K_i quá lớn có thể gây hiện tượng vọt lố (overshoot) và làm chậm đáp ứng của hệ thống. Ngoài ra, cần chú ý hiện tượng "windup" khi tích phân tích lũy quá lớn trong các tình huống bão hòa.

Thành phần vi phân (D):

Dự đoán xu hướng thay đổi của sai số:

$$u_D = K_d \cdot \frac{de(t)}{dt} \tag{18}$$

Thành phần này phản ứng với tốc độ thay đổi của sai số, giúp "phanh" hệ thống khi sai số đang giảm nhanh, từ đó giảm vọt lố và dao động. Nhược điểm là nhạy cảm với nhiễu tần số cao trong tín hiệu đo lường.

2.5.3. Dạng rời rạc của PID cho hệ thống số

Trong thực tế, vi điều khiển làm việc với tín hiệu rời rạc. Thuật toán PID được rời rạc hóa với chu kỳ lấy mẫu T_s:

$$u[k] = K_p \cdot e[k] + K_i \cdot T_s \sum_{j=0}^{k} e[j] + K_d \cdot \frac{e[k] - e[k-1]}{T_s} \tag{19}$$

Thư viện PID_v1 cho Arduino triển khai dạng rời rạc này với các tối ưu như: anti-windup (chống tích lũy tích phân khi bão hòa), derivative kick prevention (tránh đột biến vi phân khi thay đổi setpoint), và on-the-fly tuning (cho phép thay đổi tham số trong khi chạy).

2.5.4. Phương pháp tinh chỉnh tham số PID

Có nhiều phương pháp tinh chỉnh (tuning) tham số PID, từ các phương pháp giải tích (Ziegler-Nichols, Cohen-Coon) đến các phương pháp thử nghiệm. Trong dự án này, phương pháp thử-sai (Trial and Error) được áp dụng:

Bước 1: Đặt $K_i = K_d = 0$. Tăng dần K_p cho đến khi hệ thống bắt đầu dao động liên tục quanh điểm cân bằng.

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

Bước 3: Tăng dần K_d để giảm dao động và vọt lố. Thành phần D đặc biệt hiệu quả cho hệ thống có quán tính (như khối lượng đối trọng).

Bước 4: Nếu còn sai số xác lập (hệ thống không về đúng điểm cân bằng), thêm K_i với giá trị nhỏ.

Bước 5: Lặp lại các bước trên để tinh chỉnh.

2.6. Lập trình thời gian thực cho hệ thống nhúng

2.6.1. Khái niệm hệ thống thời gian thực

Hệ thống thời gian thực (Real-time System) là hệ thống mà tính đúng đắn của kết quả không chỉ phụ thuộc vào giá trị logic của phép tính, mà còn phụ thuộc vào thời điểm kết quả được tạo ra. Trong ngữ cảnh điều khiển động cơ bước:

- **Deadline:** Mỗi xung điều khiển phải được tạo ra đúng thời điểm (trong phạm vi jitter cho phép)
- **Hậu quả vi phạm deadline:** Động cơ mất bước, chạy giật cục, hoặc dừng hẳn
- **Loại hệ thống:** Soft real-time (vi phạm deadline gây suy giảm chất lượng, không gây hỏng hóc nghiêm trọng)

2.6.2. Blocking vs Non-blocking I/O

Blocking I/O: Khi thực hiện một thao tác I/O (đọc cảm biến, gửi dữ liệu Serial), CPU chờ đợi cho đến khi thao tác hoàn thành. Trong thời gian chờ, không có tác vụ nào khác được thực hiện.

Non-blocking I/O: Thao tác I/O được khởi tạo và trả về ngay lập tức. CPU có thể thực hiện các tác vụ khác trong khi chờ I/O hoàn thành. Khi I/O sẵn sàng, CPU được thông báo (qua polling hoặc interrupt).

Trong dự án này, các giải pháp non-blocking được áp dụng:

- Thư viện HX711_ADC với hàm `update()` non-blocking

– Giới hạn tần suất in Serial (mỗi 100ms thay vì mỗi vòng lặp)

– Sử dụng Baud Rate cao (115200) để giảm thời gian blocking khi buffer đầy

– Kiểm tra `Serial.availableForWrite()` trước khi gửi dữ liệu lớn

2.6.3. Cơ chế Time-Slicing với millis()

Time-Slicing là kỹ thuật phân chia thời gian CPU cho các tác vụ khác nhau dựa trên đồng hồ hệ thống. Trong Arduino, hàm `millis()` trả về số mili-giây kể từ khi khởi động.

Cấu trúc Time-Slicing điển hình:

```
unsigned long t_task1 = 0;
unsigned long t_task2 = 0;

void loop() {
    // Tac vu nen - chay moi vong lap
    stepper.runSpeed();
    LoadCell.update();

    // Tac vu 1 - chay moi 20ms
    if (millis() - t_task1 >= 20) {
        // Doc cam bien, tinh PID
        t_task1 = millis();
    }

    // Tac vu 2 - chay moi 100ms
    if (millis() - t_task2 >= 100) {
        // In thong tin Serial
        t_task2 = millis();
    }
}
```

Cơ chế này cho phép:

– Tác vụ ưu tiên cao (`runSpeed()`) được gọi liên tục

– Tác vụ điều khiển (PID) chạy với tần số cố định (50Hz)

– Tác vụ giám sát (Serial) chạy với tần số thấp hơn (10Hz) để không ảnh hưởng đến các tác vụ quan trọng

2.6.4. Vấn đề Pulse Starvation trong điều khiển động cơ bước

Pulse Starvation (đói xung) là hiện tượng động cơ bước không nhận đủ xung điều khiển do CPU bận thực hiện các tác vụ khác. Hậu quả:

– **Giảm tốc độ thực tế:** Thay vì 16000 bước/s, động cơ có thể chỉ đạt vài trăm bước/s

– **Mất đồng bộ từ trường:** Rotor không theo kịp từ trường stator, gây mất momen hoặc quay ngược

– **Rung lắc và tiếng ồn:** Xung không đều gây kích thích cộng hưởng cơ học

Để tránh Pulse Starvation:

– Loại bỏ hoặc giảm thiểu các hàm blocking trong vòng lặp chính

– Đảm bảo thời gian thực thi của mỗi vòng lặp nhỏ hơn khoảng cách giữa các xung yêu cầu

– Sử dụng thư viện điều khiển động cơ dựa trên ngắt Timer (như FastAccelStepper) nếu cần tốc độ rất cao

2.6.5. Kỹ thuật Deadzone và Hysteresis trong điều khiển

Deadzone (Vùng chết):

Là vùng giá trị sai số mà hệ thống không phản ứng. Trong dự án này, Deadzone được đặt ± 50g:

– Nếu $|e| \leq 50$g: Động cơ dừng, PID tắt

– Nếu $|e| > 50$g: PID hoạt động, động cơ di chuyển

Mục đích: Tránh động cơ chạy liên tục để bù các sai số nhỏ do nhiễu cảm biến hoặc rung động cơ học.

Hysteresis (Độ trễ):

Là kỹ thuật sử dụng hai ngưỡng khác nhau cho việc bật và tắt một chế độ:

– Ngưỡng bật (Start threshold): 60g (= 50g + 10g hysteresis)

– Ngưỡng tắt (Stop threshold): 40g (= 50g - 10g hysteresis)

Logic hoạt động:

– Nếu đang dừng và $|e| > 60$g: Bật chế độ cân bằng

– Nếu đang chạy và $|e| < 40$g: Tắt chế độ cân bằng

Mục đích: Tránh hiện tượng bật/tắt liên tục (chattering) khi sai số dao động quanh ngưỡng Deadzone.

Minimum Speed Threshold (Ngưỡng tốc độ tối thiểu):

Động cơ bước có vùng tốc độ thấp gây cộng hưởng và rung lắc. Giải pháp:

– Nếu PID tính ra tốc độ > 0 nhưng < 2000 bước/s: Ép tốc độ lên 2000 bước/s

– Nếu PID tính ra tốc độ $= 0$ hoặc ≥ 2000: Giữ nguyên

Điều này đảm bảo khi động cơ cần chạy, nó chạy ở tốc độ đủ cao để vượt qua vùng cộng hưởng.

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:

$$v_{max} = \frac{4000}{3200} \times 2 = 2.5 \text{ mm/s} \tag{20}$$

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:

$$m_{load,max} = \frac{200 \times 120}{175} \approx 137 \text{ g} \tag{21}$$

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