

CHƯƠNG 4: LẬP TRÌNH ĐIỀU KHIỂN

4.1. Quy trình công nghệ

Để hệ thống vận hành chính xác và tránh các va chạm cơ khí không đáng có, quy trình công nghệ được thiết lập theo trình tự bắt buộc: thực hiện thiết lập ở chế độ Thủ công (Manual) trước, sau đó mới kích hoạt chế độ Tự động (Auto).

4.1.1. Chế độ thủ công và thiết lập ban đầu

Đây là bước đệm quan trọng, dùng để kiểm tra thiết bị và đưa robot về vị trí sẵn sàng làm việc.

- Kết nối hệ thống: Đầu tiên, cần kiểm tra và thiết lập kết nối giữa máy tính (PC) và vi điều khiển STM32 qua cổng USB. Khi phần mềm báo đã nhận diện được cổng COM thành công, dữ liệu tọa độ mới có thể truyền nhận thông suốt.
- Thực hiện lấy gốc (Homing): Vì khi mới cấp điện, robot chưa xác định được vị trí hiện tại trong không gian, nên ta phải nhấn lệnh Homing trên giao diện. Lúc này, STM32 điều khiển 3 cánh tay robot đi lên cho đến khi chạm vào các công tắc hành trình gắn ở đỉnh khung nhôm. Đây là bước để máy tự xác lập "tọa độ 0" chuẩn cho mọi hoạt động gấp thả sau đó.
- Kiểm tra thiết bị ngoại vi: Trong chế độ này, người dùng có thể bấm thử các nút chức năng trên màn hình để xem băng tải có quay không, đầu hút chân không có dính vật không và động cơ servo xoay đầu hút có mượt không.
- Xác nhận sẵn sàng: Chỉ khi robot đã về vị trí Home an toàn và các thiết bị chạy ổn định, hệ thống mới cho phép chuyển sang chế độ tự động.

4.1.2. Chế độ tự động (Auto)

Khi đã hoàn tất các bước setup ở chế độ Manual, ta nhấn nút **START** trên mặt tủ điện hoặc trên giao diện điều khiển để máy bắt đầu quy trình làm việc khép kín.

- Nhận diện và phân loại ảnh: Băng tải bắt đầu chạy để đưa sản phẩm đi tới. Khi vật phẩm đi qua tầm mắt của Camera, máy tính sẽ chụp ảnh và sử dụng mô hình học sâu để biết đây là vật nào nằm ở vị khay nào.
- Tính toán tọa độ: Máy tính lập tức tính toán sau đó gửi lệnh điều khiển xuống mạch STM32 qua cổng USB.
- Thực hiện gấp vật: Nhận được lệnh, STM32 phát xung cho 3 động cơ bước đưa cánh tay robot lao xuống đúng chỗ sản phẩm. Nếu sản phẩm nằm nghiêng, động cơ Servo sẽ xoay đầu gấp cho khớp hướng, đồng thời kích hoạt bơm màng để tạo lực hút giữ chặt vật.
- Phân loại sản phẩm: Robot nháy vật lên và đưa về đúng vị trí đã quy định rồi ngắt

lực hút để thả vật xuống.

- Lặp lại chu kỳ: Xong một lượt, robot quay về vị trí chờ sản phẩm tiếp theo. Đèn báo trên tủ điện sẽ sáng liên tục trong suốt quá trình này để thông báo máy vẫn đang vận hành bình thường.

4.2. Tổng quan kiến trúc điều khiển

Hệ thống được thiết kế theo mô hình **Master-Slave**, trong đó máy tính (PC) đóng vai trò Master chịu trách nhiệm xử lý tác vụ nặng (Computer Vision, Kinematics, Nội suy quỹ đạo), và vi điều khiển STM32 đóng vai trò Slave chịu trách nhiệm điều khiển thời gian thực (phát xung điều khiển động cơ bước).

4.2.1. Giao thức giao tiếp

Để đảm bảo việc truyền tải dữ liệu giữa PC và STM32 diễn ra chính xác, tin cậy và đáp ứng được yêu cầu về tốc độ, đồ án sử dụng giao thức giao tiếp qua cổng **USB CDC** (**Communication Device Class**) giả lập cổng Serial.

4.2.1.1. Cấu hình vật lý:

- **Baudrate giả lập:** 115200 bps (để tương thích phần mềm).
- **Tốc độ vật lý:** ~12 Mbps. **Kết nối:** Cáp USB Type-C/Micro-USB nối trực tiếp từ PC xuống cổng USB của STM32.

4.2.1.2. Định dạng gói tin.

Dữ liệu được truyền dưới dạng chuỗi ký tự ASCII để thuận tiện cho việc gỡ lỗi (debug), cấu trúc lệnh được thiết kế theo dạng KEYWORD:DATA.

- Lệnh từ PC gửi xuống (Request):

- ADD_BLOCK:id:{params}: Lệnh di chuyển quan trọng nhất. Trong đó, params là chuỗi định dạng giống JSON chứa thông tin thời gian thực hiện (t), số bước động cơ (s), góc servo (a) và trạng thái bơm (b).
- HOME: Yêu cầu robot về điểm gốc.
- STATUS: Yêu cầu vi điều khiển báo cáo trạng thái hiện tại.
- CONVEYOR:START/STOP: Điều khiển băng tải.

- Phản hồi từ STM32 gửi lên (Response):

- DONE:id: Báo cáo đã thực hiện xong lệnh di chuyển có mã id. Tín hiệu này cực kỳ quan trọng để PC biết khi nào nên gửi tiếp lệnh mới (Flow Control).
- STATUS:state:homed:estop...: Gói tin chứa toàn bộ trạng thái sensor, nút

nhanh, tọa độ hiện tại.

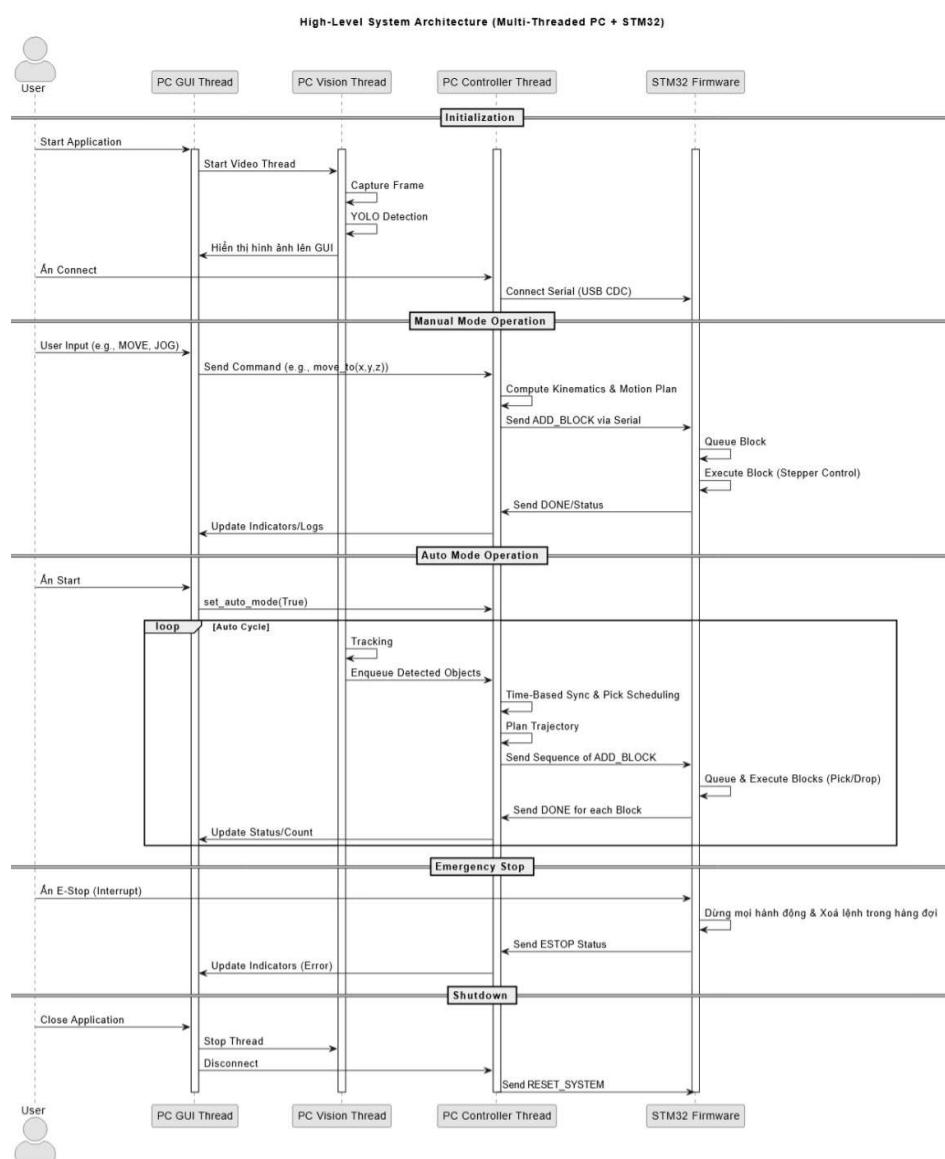
- o ERROR:msg: Báo lỗi hệ thống.

4.2.1.3. Cơ chế điều khiển luồng (Flow Control)

Do bộ nhớ RAM của STM32 có hạn, hệ thống không gửi toàn bộ quỹ đạo di chuyển một lúc. Thay vào đó, một cơ chế "**Sliding Window**" được áp dụng: PC duy trì một hàng đợi lệnh trên STM32 luôn đầy ở mức an toàn. Khi STM32 thực hiện xong một lệnh, PC sẽ lập tức gửi lệnh tiếp theo để điền vào chỗ trống.

4.2.2. Tổng quát về kiến trúc hệ thống

Dựa trên sơ đồ kiến trúc hệ thống, hoạt động của robot được chia thành các giai đoạn chính sau đây, tận dụng kiến trúc đa luồng trên máy tính để đảm bảo xử lý song song các tác vụ.



Hình 4-1: Kiến trúc của hệ thống robot Delta

a. Khởi tạo (Initialization)

- Khi người dùng khởi động ứng dụng, ba luồng chính được kích hoạt: Luồng giao diện (GUI), Luồng xử lý ảnh và Luồng điều khiển.
- **Vision Thread:** Ngay lập tức kết nối với Camera, bắt đầu chụp hình và nạp mô hình YOLOv8 để sẵn sàng nhận diện [7].
- **Controller Thread:** Người dùng ấn nút "Connect", PC thiết lập kết nối Serial với STM32. STM32 khởi tạo các ngoại vi (GPIO, Timer, USB) và đi vào trạng thái chờ lệnh.

b. Chế độ Thủ công

Đây là chế độ dùng để kiểm tra, tinh chỉnh hoặc xử lý sự cố:

- Khi người dùng thao tác trên GUI (ví dụ: nhấn nút JOG X+), lệnh move được gửi đến Controller Thread.
- PC tính toán động học nghịch để ra số bước, sau đó đóng gói thành lệnh ADD_BLOCK gửi xuống STM32.
- STM32 nhận lệnh, đưa vào hàng đợi và sử dụng ngắt Timer để điều khiển động cơ bước chạy đúng quỹ đạo. Sau khi hoàn thành, STM32 gửi DONE để PC cập nhật giao diện.

c. Chế độ Tự động

Đây là chế độ hoạt động chính của hệ thống, thể hiện tính liên tục và đồng bộ thời gian thực:

1. **Nhận diện:** Vision Thread liên tục xử lý hình ảnh từ camera. Khi phát hiện vật thể, nó gửi thông tin (tọa độ, loại vật, góc xoay) vào một hàng đợi an toàn (Thread-safe Queue).
2. **Lập lịch:** Controller Thread lấy dữ liệu từ hàng đợi. Dựa trên tốc độ băng tải và vị trí hiện tại của robot, thuật toán sẽ tính toán điểm đón (Intercept Point) sao cho robot và vật gặp nhau tại đúng một thời điểm trong tương lai.
3. **Quy hoạch & Gửi lệnh:** Quỹ đạo di chuyển được chia nhỏ thành nhiều đoạn ngắn và gửi liên tục xuống STM32 thông qua cơ chế Sliding Window đã mô tả ở trên.
4. **Thực thi:** STM32 thực thi các lệnh trong bộ đệm vòng (Ring Buffer), điều khiển đầu hút và thả vật mà không cần dừng lại chờ PC, giúp chuyển động mượt mà.

d. Xử lý An toàn (Emergency Stop)

- Nút E-Stop được đấu nối trực tiếp vào phần cứng STM32 và được xử lý bằng ngắt ngoài (External Interrupt).
 - Khi nút E-Stop bị nhấn, STM32 lập tức ngắt xung cấp cho động cơ, xóa toàn bộ hàng đợi lệnh và gửi trạng thái ESTOP lên PC.
 - PC nhận tín hiệu sẽ khóa giao diện điều khiển và hiển thị cảnh báo lỗi, ngăn chặn mọi hành động không kiểm soát thao tác cho đến khi xử lý được sự cố
- e. Đóng hệ thống (Shutdown)
- Khi người dùng tắt ứng dụng, PC gửi lệnh ngắt kết nối và giải phóng tài nguyên Camera, Serial để tránh treo cổng COM cho lần sử dụng sau. STM32 nhận lệnh RESET_SYSTEM để đưa các cơ cấu chấp hành về trạng thái an toàn.

4.3. Xây dựng chương trình điều khiển STM32

4.3.1. Cấu trúc tổ chức mã nguồn

Chương trình điều khiển nhúng được viết bằng ngôn ngữ C, sử dụng thư viện HAL (Hardware Abstraction Layer) của STM32 để đảm bảo tính ổn định và dễ dàng bảo trì. Mã nguồn được chia thành các module chức năng riêng biệt nhằm đảm bảo tính đóng gói (encapsulation).

Danh sách các tệp tin mã nguồn chính trong dự án:

1. Tầng Ứng dụng (Application Layer):

- main.c: Tệp chính chứa hàm main(), khởi tạo hệ thống và vòng lặp vô tận (while(1)). Đóng vai trò bộ điều phối trung tâm, gọi các hàm xử lý từ các module khác.

2. Tầng Logic Điều khiển (Control Logic Layer):

- robot_control.c: Chứa thuật toán cốt lõi điều khiển động cơ bước, quy trình về gốc (Homing), và xử lý ngắt Timer để tạo xung.
- command_queue.c: Cài đặt cấu trúc dữ liệu hàng đợi vòng (Ring Buffer) để lưu trữ các lệnh chuyển động, giúp tách biệt quá trình nhận lệnh và thực thi lệnh.
- command_parser.c: Phân tích chuỗi ký tự nhận được từ máy tính (JSON text) thành các tham số số học (thời gian, bước, trạng thái I/O).

3. Tầng Giao tiếp & Ngoại vi (Driver/Peripheral Layer):

- cdc_handler.c: Xử lý giao tiếp USB CDC ở cấp độ thấp, nhận dữ liệu thô

tù bộ đệm phần cứng.

- button_handler.c: Xử lý ngắn nút nhấn, khử rung (debounce) và logic dừng khẩn cấp (E-Stop).
- conveyor.c: Các hàm điều khiển động cơ băng tải (Start/Stop/Set Speed).
- status_led.c: Điều khiển đèn báo trạng thái hệ thống.

4.3.2. Phân công đầu ra đầu vào và cấu hình STM32 trên Cube MX

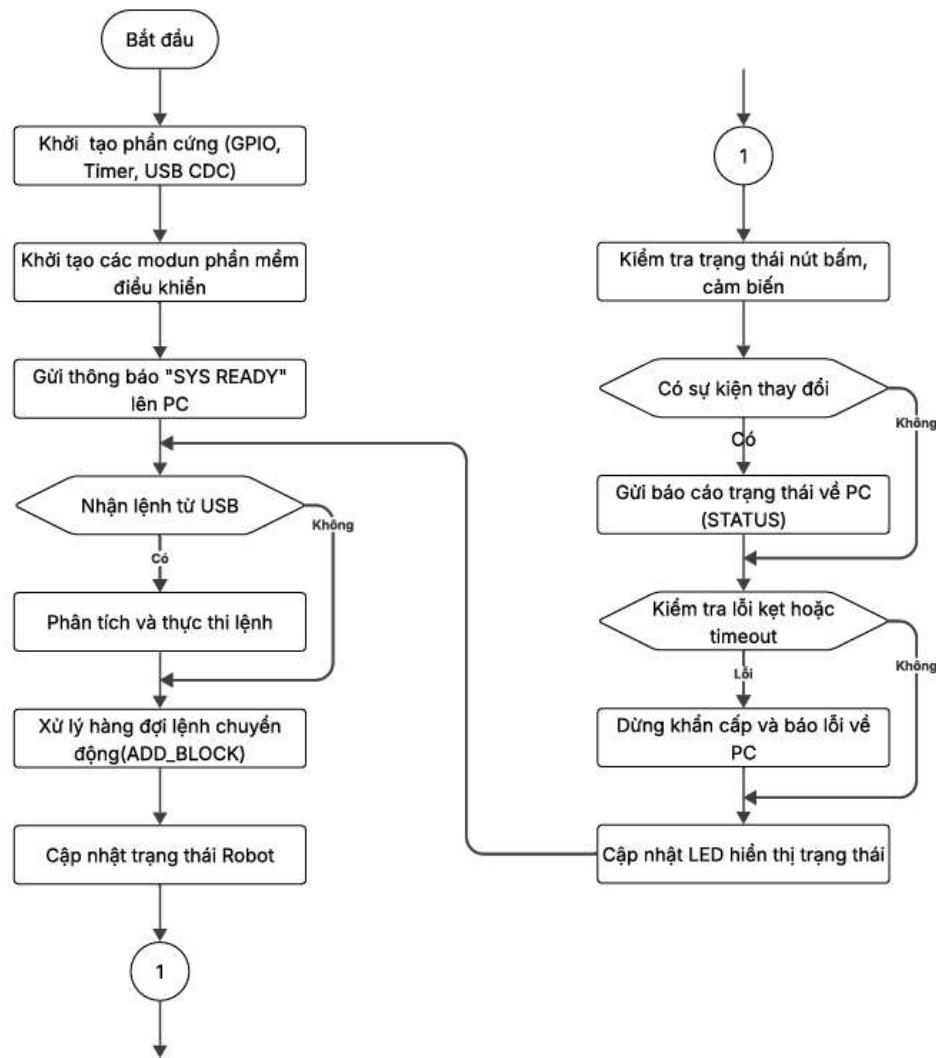
4.3.2.1. Phân công đầu vào đầu ra

Bảng 4-1: Bảng phân công đầu vào đầu ra

	STT	Tên chức năng	Ký hiệu trong code	Cổng	Chân	Cấu hình nội trở (Đầu vào)
Đầu ra	1	PUL Motor1	M1_PUL	GPIOB	Pin 1	
	2	DIR Motor1	M1_DIR	GPIOB	Pin 10	
	3	ENA Motor1	M1_ENA	GPIOB	Pin 11	
	4	PUL Motor2	M2_PUL	GPIOB	Pin 0	
	5	DIR Motor2	M2_DIR	GPIOA	Pin 5	
	6	ENA Motor2	M2_ENA	GPIOA	Pin 6	
	7	PUL Motor3	M3_PUL	GPIOA	Pin 7	
	8	DIR Motor3	M3_DIR	GPIOA	Pin 3	
	9	ENA Motor3	M3_ENA	GPIOA	Pin 4	
	10	PUL MotorBT	CONV_PUL	GPIOA	Pin 0	
	11	DIR MotorBT	CONV_DIR	GPIOA	Pin 1	
	12	ENA MotorBT	CONV_ENA	GPIOA	Pin 2	
	13	Bơm hút chân không	PUMP	GPIOA	Pin 9	
	14	Đèn trợ sáng camera	CAMERA_LIGHT	GPIOB	Pin 4	
	15	Đèn báo xanh	GREEN_LED	GPIOA	Pin 10	
	16	Đèn báo đỏ	RED_LED	GPIOB	Pin 15	
	17	Đèn nút Start	BUTTON_LED	GPIOB	Pin 3	
Đầu vào	18	Cảm biến khay	CONV_SENSOR	GPIOB	Pin 15	Pull_Down
	19	CBHT1	LS1	GPIOB	Pin 12	Pull_up
	20	CBHT2	LS2	GPIOB	Pin 13	Pull_up
	21	CBHT3	LS3	GPIOB	Pin 14	Pull_up
	22	E STOP	ESTOP	GPIOA	Pin 8	Pull_Up
	23	START	BTN_START	GPIOB	Pin 6	Pull_Down
	24	STOP	BTN_STOP	GPIOB	Pin 7	Pull_Down

4.3.3. Xây dựng chương trình chính

Chương trình chính trong tệp main.c là trung tâm điều khiển của hệ thống phần cứng trên vi điều khiển STM32F103. Nó chịu trách nhiệm khởi tạo các thành phần phần cứng và phần mềm, đồng thời quản lý vòng lặp chính để xử lý các sự kiện thời gian thực. Chương trình được thiết kế theo mô hình không chẵn, sử dụng các hàm cập nhật định kỳ để đảm bảo hệ thống phản ứng nhanh chóng với các đầu vào từ nút bấm, cảm biến, và giao tiếp USB, đồng thời duy trì tính an toàn và hiệu suất cao.



Hình 4-2: Lưu đồ thuật toán vòng lặp xử lý chính của STM32

Chương trình chính được chia làm hai giai đoạn: giai đoạn khởi tạo và vòng lặp chính.

a. Giai đoạn khởi tạo (Initialization)

Ngay khi cấp điện, hệ thống thực hiện các bước sau:

1. HAL_Init & SystemClock_Config: Cấu hình thư viện HAL và thiết lập bộ dao động thạch anh ngoại (HSE) để hệ thống chạy ở tốc độ 72MHz.

2. Khởi tạo Ngoại vi (Peripheral Init):

- MX_GPIO_Init: Cấu hình các chân I/O cho Step/Dir, bơm hút, cảm biến hành trình, nút bấm.
- MX_TIMx_Init: Cấu hình Timer 2 (PWM băng tải), Timer 4 (PWM Servo), và Timer 3 (Bộ đếm bước).
- MX_USB_DEVICE_Init: Kích hoạt giao tiếp USB CDC (Virtual COM Port).

3. Khởi tạo Module phần mềm: Gọi các hàm init của từng module con (robot_init, conveyor_init, queue_init) để thiết lập các biến trạng thái ban đầu.

4. Thông báo sẵn sàng: Sau khi ổn định (delay 100ms), gửi chuỗi "SYS_READY" lên máy tính để báo hiệu quá trình khởi động hoàn tất.

b. Vòng lặp chính (Main Loop)

Vòng lặp while(1) thực hiện các tác vụ tuần tự với tốc độ rất cao:

1. Xử lý giao tiếp (Communication):

- Hàm cdc_handler_get_command kiểm tra xem có dữ liệu mới từ bộ đệm USB không.
- Nếu có lệnh, parse_command sẽ phân tích chuỗi ký tự (ví dụ: ADD_BLOCK, CONVEYOR, HOME) và thực thi hành động tương ứng ngay lập tức.

2. Quản lý hàng đợi chuyển động (Queue Management):

- queue_process(): Kiểm tra xem robot có đang rảnh không và hàng đợi có lệnh chờ không. Nếu có, nó nạp lệnh tiếp theo vào bộ điều khiển robot.
- queue_handle_done_messages(): Xử lý các tín hiệu từ ngắt (Interrupt Service Routine - ISR) báo về khi một chuyển động hoàn tất, gửi phản hồi ACK hoặc DONE lên PC.

3. Giám sát đầu vào (Input Monitoring):

- Nút bấm & E-Stop: button_handler_update() đọc trạng thái nút nhấn và nút dừng khẩn cấp, có áp dụng thuật toán chống rung (debounce). Nếu E-Stop bị kích hoạt, trạng thái sẽ được gửi ngay lập tức về PC.
- Cảm biến khay (Tray Sensor): Giám sát chân PB15. Nếu trạng thái khay thay đổi (có/không có khay), hệ thống gửi gói tin STATUS cập nhật cho phần mềm điều khiển.

4. Logic robot & an toàn (Safety Logic):

- robot_update_homing_state(): Quản lý máy trạng thái của quy trình về Home (Homing sequence).

- Kiểm tra quá trình về home: Nếu quá trình về Home kéo dài quá lâu (do kẹt cơ khí hoặc hỏng cảm biến), hệ thống tự động hủy và báo lỗi.
- Công tắc hàng trình: Kiểm tra xem các công tắc hành trình có bị kẹt (stuck) hay không. Nếu phát hiện bất thường, robot sẽ dừng khẩn cấp (robot_abort) để bảo vệ phần cứng.

5. Chỉ thị trạng thái: status_led_update() điều khiển đèn LED trên mạch nhấp nháy theo các tần số khác nhau tùy thuộc vào trạng thái hiện tại (đang chạy, lỗi, hoặc chờ), giúp người vận hành dễ dàng nhận biết trực quan.

4.3.4. Xử lý lệnh từ PC

4.3.4.1. Mục đích

Trong quá trình hoạt động tự động, máy tính (PC) gửi xuống vi điều khiển hàng loạt các tọa độ chuyển động liên tiếp (Path Planning). Tốc độ gửi dữ liệu qua USB (12 Mbps) nhanh hơn rất nhiều so với tốc độ cơ học của robot (thường mất vài trăm mili-giây để thực hiện một chuyển động).

Nếu robot thực hiện xong một lệnh rồi mới chờ nhận lệnh tiếp theo, chuyển động sẽ bị giật cục (Stop-and-Go). Để giải quyết vấn đề này, hệ thống sử dụng cơ chế hàng đợi lệnh theo cấu trúc vào trước thì xử lý trước và bộ đệm vòng điêu này cho phép vi điều khiển nhận trước các lệnh tương lai trong khi đang thực hiện lệnh hiện tại.

4.3.4.2. Cấu trúc dữ liệu

Mỗi lệnh chuyển động được đóng gói thành một cấu trúc dữ liệu MotionBlock. Cấu trúc này chứa đầy đủ thông tin cần thiết để bộ điều khiển robot thực thi mà không cần tính toán lại.

```
typedef struct {  
    uint32_t id; là ID của block (để đồng bộ với PC)  
    int32_t motor_steps[t1,t2,t3]; là số bước cần di chuyển cho 3 trục  
    uint32_t duration; là thời gian thực hiện chuyển động (ms)  
    uint16_t servo_pulse; là góc quay Servo (độ rộng xung PWM)  
    bool pump_state; là trạng thái bơm hút (ON/OFF)
```

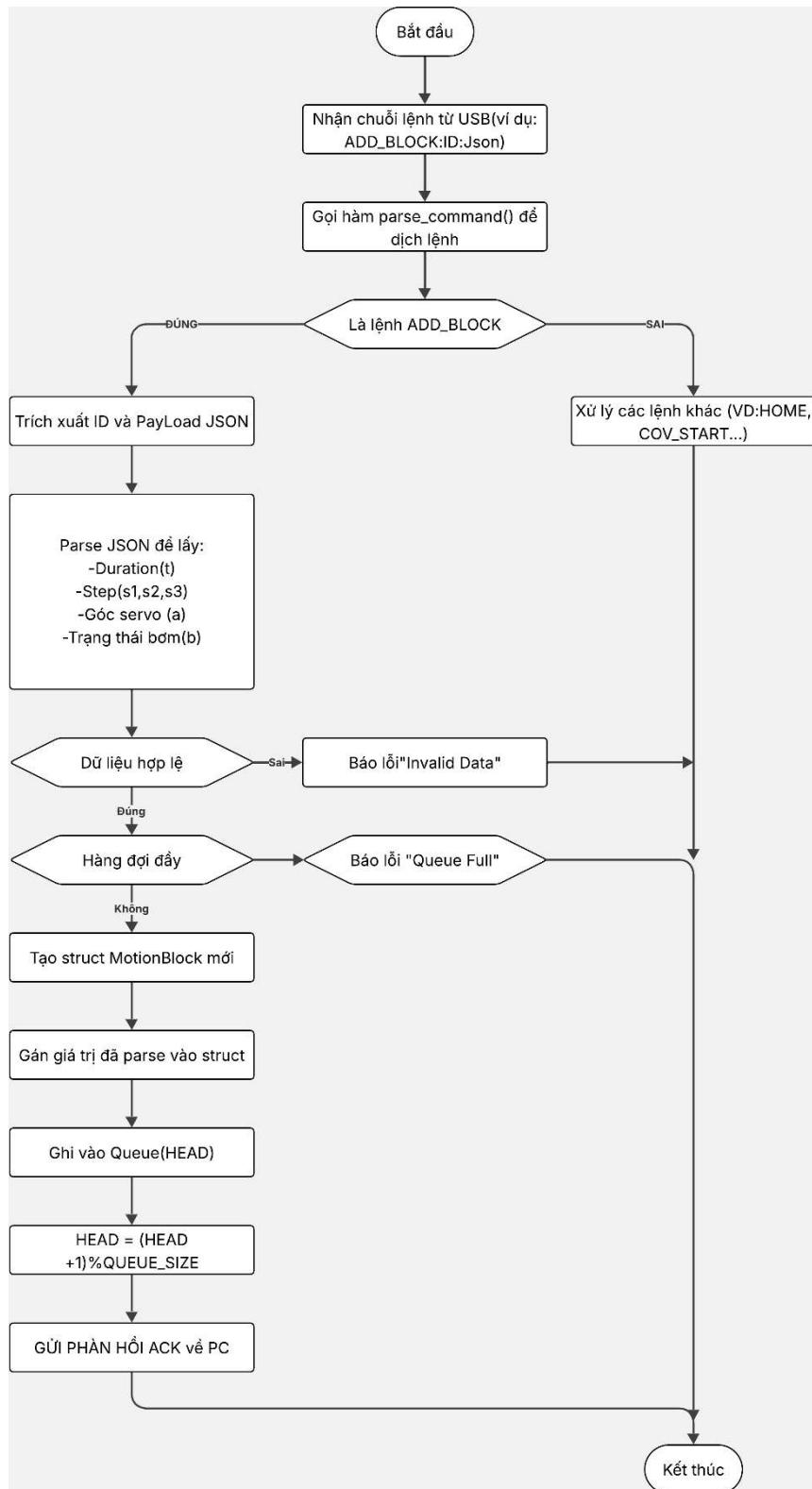
4.3.4.3. Quy trình xử lý

Quy trình từ lúc nhận chuỗi ký tự thô từ USB đến khi đưa vào hàng đợi diễn ra như sau:

1. Phân tích cú pháp (Parsing): Hệ thống nhận chuỗi lệnh ADD_BLOCK định dạng

JSON (ví dụ:0:{ "t":500,"s":[100,200,300]...}).

2. Trích xuất dữ liệu: Các trường thông tin như thời gian t, số bước s, góc servo a được tách ra và chuyển đổi thành số nguyên.
3. Kiểm tra tính hợp lệ: Đảm bảo các giá trị nằm trong giới hạn an toàn (ví dụ: góc servo không quá giới hạn cơ khí, thời gian không bằng 0).
4. Đẩy vào hàng đợi: Nếu hàng đợi chưa đầy, dữ liệu được ghi vào vị trí Head, sau đó tăng chỉ số Head.



Hình 4-3: Lưu đồ thuật toán xử lý lệnh từ PC

4.3.4.4. Nguyên lý bộ đệm vòng

Hàng đợi được quản lý bởi hai biến chỉ số:

- head: Vị trí sẽ ghi lệnh mới vào.
- tail: Vị trí lệnh đang được robot lấy ra để thực thi.

Quy tắc hoạt động:

1. Thêm lệnh (PC gửi xuống): Ghi vào buffer[head] và tăng head. Nếu head đuổi kịp tail, hàng đợi đầy (Full).
2. Thực thi lệnh (Robot chạy): Trong vòng lặp chính, kiểm tra nếu head != tail (hàng đợi có dữ liệu) và robot đang rảnh (Idle), hệ thống sẽ lấy lệnh tại buffer[tail], nạp vào bộ tạo xung và tăng tail.

4.3.4.5. Ý nghĩa trong hệ thống

Việc thiết kế tách biệt giữa nhận lệnh và thực thi lệnh thông qua hàng đợi giúp hệ thống đạt được các lợi ích:

- Đa nhiệm: Vì điều khiển có thể xử lý việc nhận dữ liệu USB ngay cả khi động cơ đang quay.
- Ổn định: Robot di chuyển liên tục theo quỹ đạo được quy hoạch trước mà không bị phụ thuộc vào độ trễ truyền thông của máy tính.
- An toàn: Dữ liệu được kiểm tra kỹ lưỡng trước khi vào hàng đợi, ngăn chặn các lệnh sai gây hỏng phần cứng.

4.3.5. Xử lý ngắt phát xung điều khiển động cơ

4.3.5.1. Giới thiệu

Đây là thành phần cốt lõi trong firmware của bộ điều khiển robot, chịu trách nhiệm chuyển đổi các thông số quỹ đạo thành các tín hiệu điện để điều khiển chính xác 3 động cơ bước theo thời gian thực. Hệ thống sử dụng Timer phần cứng TIM3 hoạt động ở chế độ ngắt chu kỳ với tần số 80kHz (tương đương chu kỳ ngắt T = 12.5μs).

4.3.5.2. Giải thuật phát xung DDS (Direct Digital Synthesis)

Để đảm bảo robot di chuyển mượt mà với vận tốc chính xác, hệ thống không sử dụng các vòng lặp delay truyền thông (vốn gây lãng phí CPU và khó đồng bộ nhiều trực). Thay vào đó, thuật toán DDS (hay còn gọi là thuật toán Bresenham mở rộng cho vận tốc) được áp dụng.

- Nguyên lý hoạt động:

Mỗi động cơ được quản lý bởi một biến tích lũy 32-bit (accumulator) và một giá trị gia tốc (speed_addend) đại diện cho vận tốc góc mong muốn.

Tại mỗi chu kỳ ngắt Timer (12.5μs), hệ thống thực hiện phép tính:

$$Accumulator_{new} = Accumulator_{old} + Speed_{Addend}$$

- Nếu biến accumulator bị tràn (tức giá trị quay vòng từ lớn nhất về nhỏ):
- Một xung điều khiển mức cao được gửi ra chân STEP.

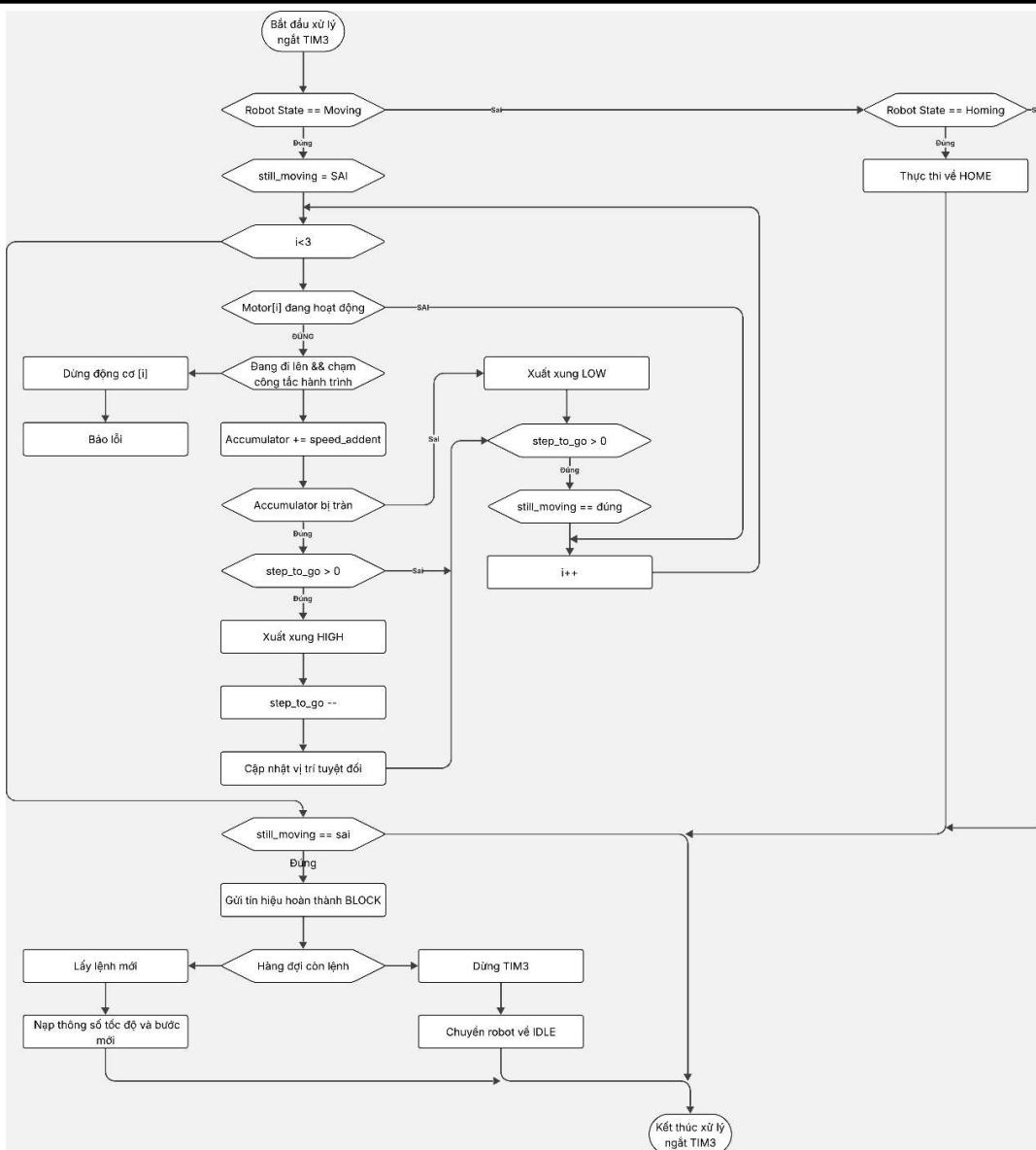
- Số bước cần đi (steps_to_go) giảm đi 1.
- Nếu không tràn: Chân STEP được giữ hoặc kéo về mức thấp.

Giải thuật này cho phép tạo ra xung điều khiển với độ phân giải thời gian rất cao, giúp khử triệt để hiện tượng rung động (jitter) khi động cơ chạy ở tốc độ thấp.

4.3.5.3. Cấu trúc bộ điều khiển trong ngắt

Hàm xử lý ngắt TIM3_IRQHandler thực hiện các tác vụ điều khiển tuần tự:

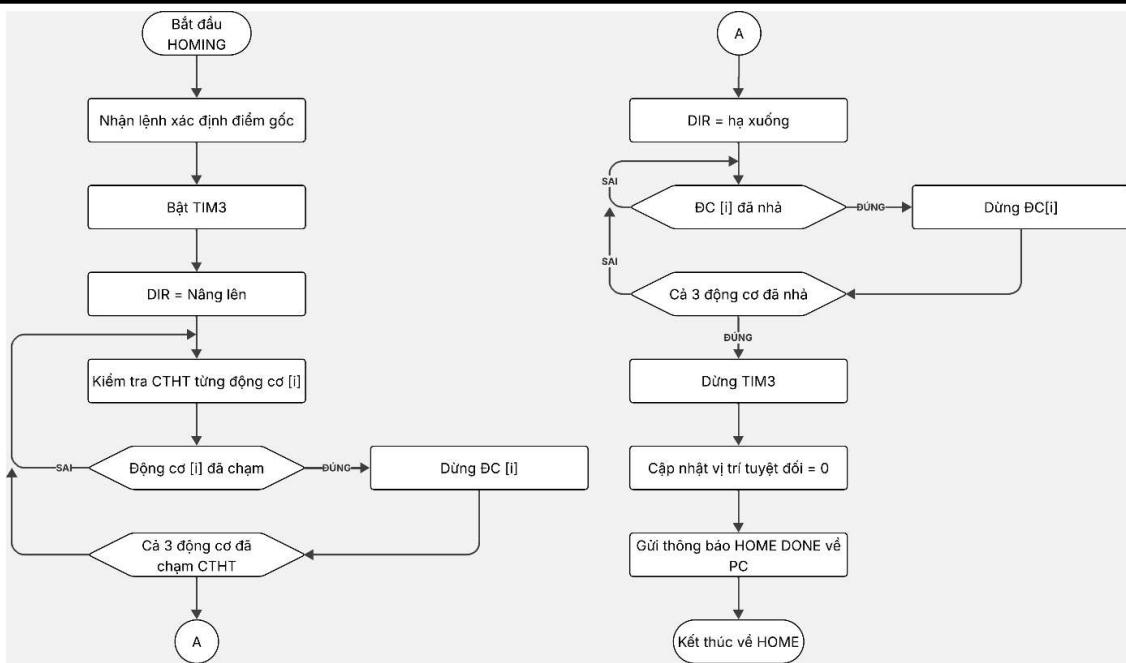
1. Kiểm tra trạng thái hệ thống: Ngắt chỉ thực thi logic điều khiển khi robot ở trạng thái MOVING hoặc HOMING.
2. Vòng lặp điều khiển đa trực: Quét lần lượt qua 3 động cơ để xử lý độc lập.
3. Giám sát an toàn: Trước khi phát xung, hệ thống đọc trạng thái các công tắc hành trình. Nếu phát hiện robot đang di chuyển lên mà chạm công tắc, động cơ tương ứng sẽ bị dừng khẩn cấp.
4. Phát xung: Thực thi thuật toán DDS để quyết định việc đảo trạng thái chân GPIO.
5. Quản lý chuyển tiếp lệnh: Khi tất cả động cơ hoàn thành số bước của lệnh hiện tại, hệ thống tự động lấy lệnh tiếp theo từ hàng đợi. Việc nạp lệnh mới diễn ra ngay trong ngắt, đảm bảo robot chuyển động liên tục mà không bị dừng giữa các đoạn đường.



Hình 4-4: Lưu đồ thuật toán phát xung điều khiển động cơ

4.3.6. Quy trình xác định điểm gốc hệ tọa độ

Quy trình xác định điểm gốc là bước bắt buộc ngay sau khi khởi động hệ thống hoặc sau khi xảy ra sự cố dừng khẩn cấp. Mục đích của quy trình này là đồng bộ hóa vị trí vật lý của ba cánh tay robot với hệ tọa độ tính toán trong phần mềm.



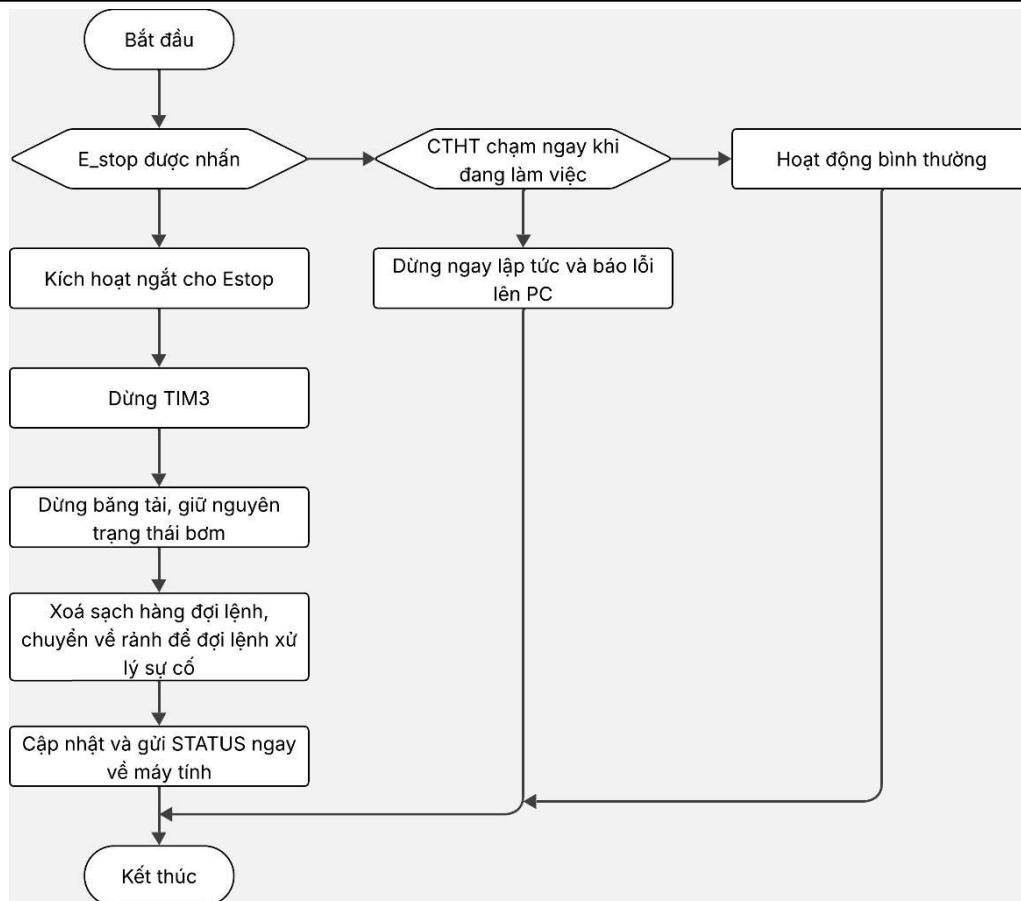
Hình 4-5: Quy trình về home

Quy trình bắt đầu bằng việc kích hoạt các động cơ quay theo chiều nâng trực lên phía trên. Hệ thống liên tục kiểm tra tín hiệu từ ba công tắc hành trình. Khi tất cả các trục đã tiếp xúc với công tắc, robot sẽ dừng lại và chuyển sang giai đoạn lùi trực. Vì để giảm thời gian về HOME, tốc độ nâng lên khác nhanh, nên khi chạm vào công tắc hành trình bị quá và không chính xác, vì vậy việc lùi trực với tốc độ chậm đảm bảo robot ở đúng vị trí home khi công tắc vừa nhả ra. Khi công tắc đã trở về trạng thái mở, bộ định thời sẽ bị ngắt và vị trí này chính thức được coi là mốc tọa độ không của hệ thống.

4.3.7. Các cơ chế bảo vệ khi

Để đảm bảo robot hoạt động ổn định và tránh hỏng hóc cơ khí, phần mềm điều khiển được tích hợp các lớp bảo vệ đa tầng.

Lưu đồ cơ chế dừng khẩn cấp và bảo vệ:

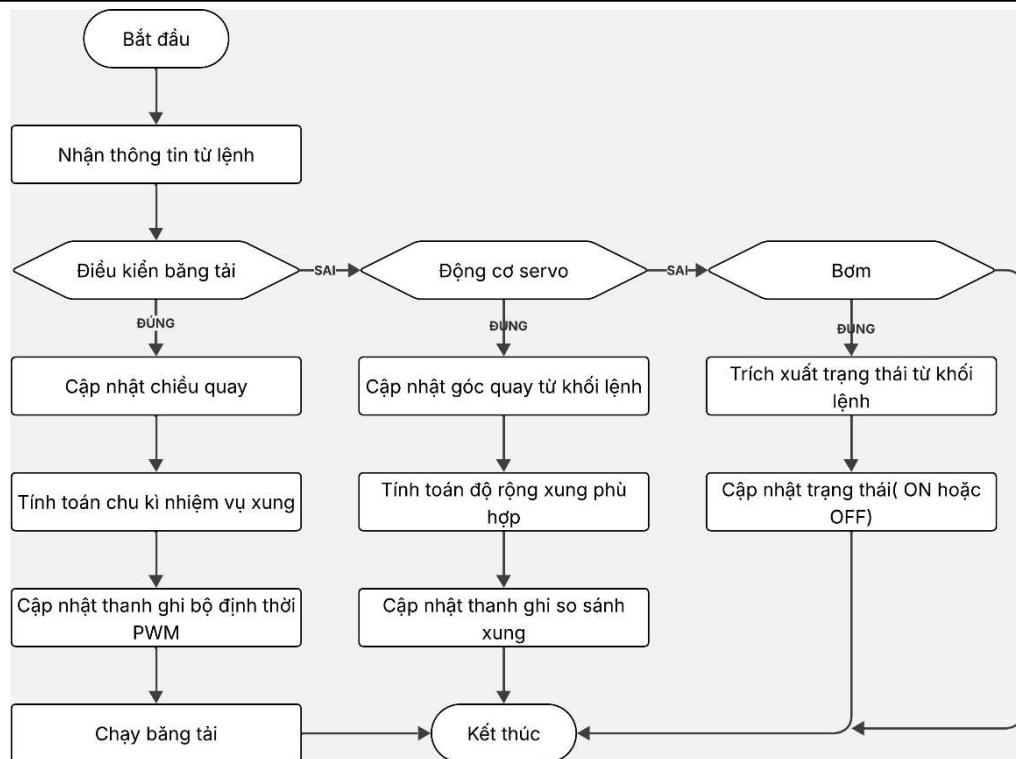


Hình 4-6: Lưu đồ cơ chế dừng an toàn của hệ thống

Hệ thống giám sát an toàn hoạt động dựa trên thứ tự ưu tiên. Mức ưu tiên cao nhất thuộc về nút dừng khẩn cấp vật lý; khi được kích hoạt, động cơ robot và băng tải sẽ dừng ngay lập tức, bơm sẽ giữ nguyên trạng thái để nếu vật đang hút không rơi. Mức ưu tiên thứ hai là bảo vệ quá hành trình, ngăn chặn cánh tay robot va đập vào khung cơ khí khi có sai số tính toán.

4.3.8. Điều khiển các thiết bị ngoại vi

Vì điều khiển quản lý đồng bộ các thiết bị hỗ trợ để hoàn thành chu trình phân loại sản phẩm một cách tự động.



Hình 4-7: Lưu đồ thuật toán điều khiển ngoại vi

Việc điều khiển thiết bị ngoại vi được phân chia theo đặc tính kỹ thuật của từng loại:

- Đối với băng tải và động cơ xoay sản phẩm, hệ thống sử dụng phương pháp điều chế độ rộng xung thông qua các bộ định thời phần cứng độc lập.
- Riêng với động cơ xoay, phần mềm thực hiện thêm bước tính toán để chuyển đổi giá trị góc quay từ lệnh máy tính sang độ rộng xung tương ứng trong dải micro giây.
- Bơm hút sản phẩm có cấu tạo đơn giản hơn nên được điều khiển trực tiếp ON/OFF qua chân đầu ra của STM32

Toàn bộ các thao tác này được nạp sẵn và thực thi đồng bộ cùng với các khối lệnh di chuyển chính của robot.

4.4. Xây dựng chương trình điều khiển trên máy tính

Phần mềm điều khiển trên máy tính đóng vai trò là trung tâm xử lý trí tuệ của toàn bộ hệ thống. Nhiệm vụ chính của phần mềm là thu nhận hình ảnh từ camera, thực hiện các thuật toán nhận diện vật thể sử dụng trí tuệ nhân tạo, tính toán động học nghịch và gửi lệnh điều khiển xuống vi điều khiển để vận hành cánh tay robot.

4.4.1. Môi trường phát triển và công cụ

Để đáp ứng yêu cầu về tốc độ xử lý ảnh và khả năng tính toán phức tạp, nhóm thực hiện đã cân nhắc kỹ lưỡng và quyết định lựa chọn các công cụ phát triển phù hợp.

Về ngôn ngữ lập trình, Python phiên bản 3.11 trở lên được lựa chọn làm ngôn ngữ

chủ đạo. Lý do chính cho quyết định này nằm ở sự phổ biến vượt trội của Python trong lĩnh vực trí tuệ nhân tạo và thị giác máy tính. Python sở hữu một hệ sinh thái thư viện khổng lồ, cộng đồng hỗ trợ lớn và khả năng phát triển ứng dụng nhanh chóng. Mặc dù là ngôn ngữ thông dịch, nhưng với sự hỗ trợ của các công cụ tăng tốc hiện đại, Python hoàn toàn có thể đáp ứng tốt các bài toán thời gian thực trong phạm vi đồ án.

Môi trường soạn thảo mã nguồn được sử dụng là Visual Studio Code, một công cụ nhẹ, linh hoạt và hỗ trợ tốt việc quản lý dự án cũng như gỡ lỗi.

Các thư viện nòng cốt được sử dụng trong dự án bao gồm:

- PyQt6 là thư viện dùng để xây dựng giao diện người dùng đồ họa hiện đại. Thư viện này hỗ trợ cơ chế xử lý tín hiệu và khe (Signal & Slot), giúp tách biệt luồng giao diện và luồng xử lý, đảm bảo ứng dụng hoạt động mượt mà không bị treo khi thực hiện các tác vụ nặng.
- OpenCV và Ultralytics là hai thư viện quan trọng nhất cho nhiệm vụ thị giác máy tính. OpenCV đảm nhiệm việc đọc, tiền xử lý hình ảnh và các phép biến đổi hình học. Trong khi đó, Ultralytics cung cấp nền tảng tối ưu để triển khai mô hình học sâu YOLOv8, giúp việc nhận diện vật thể đạt độ chính xác cao và tốc độ nhanh.
- NumPy và Numba đóng vai trò nền tảng cho các tính toán toán học. NumPy hỗ trợ các phép toán ma trận hiệu năng cao cần thiết cho xử lý ảnh. Đặc biệt, thư viện Numba được sử dụng để biên dịch các hàm tính toán động học nghịch sang mã máy ngay tại thời điểm chạy. Điều này giúp tốc độ xử lý các phép toán lượng giác phức tạp tăng lên đáng kể, đạt mức hiệu năng tương đương với ngôn ngữ C/C++.
- PySerial là thư viện quản lý giao tiếp nối tiếp, đóng vai trò cầu nối truyền nhận dữ liệu lệnh và phản hồi giữa máy tính và vi điều khiển STM32 thông qua cổng USB.

4.4.2. Cấu trúc tổng thể chương trình

Chương trình được thiết kế theo tư duy hướng đối tượng và kiến trúc mô-đun hóa. Việc chia tách mã nguồn thành các tệp chức năng riêng biệt giúp mã nguồn dễ dàng quản lý, bảo trì và mở rộng các tính năng mới sau này. Cấu trúc mã nguồn trong thư mục dự án được tổ chức thành ba tầng xử lý chính.

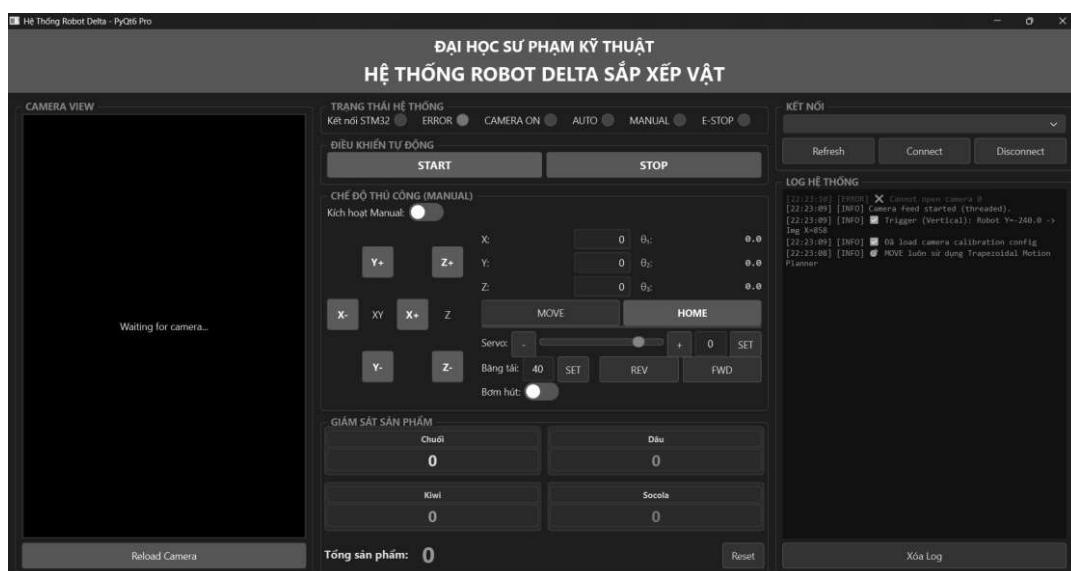
- Tầng ứng dụng và giao diện bao gồm tệp khởi chạy main.py và tệp giao diện pyqt_delta_gui.py. Tệp main.py chịu trách nhiệm khởi tạo toàn bộ hệ thống, thiết lập các luồng xử lý song song và kích hoạt vòng lặp sự kiện. Tệp pyqt_delta_gui.py chứa mã nguồn thiết kế các thành phần trực quan như nút bấm,

khung hiển thị camera, ô nhập liệu thông số và bảng hiển thị trạng thái, giúp người dùng tương tác dễ dàng với hệ thống.

- Tầng xử lý và tính toán bao gồm các mô-đun vision_system.py, deep_learning_processor.py và kinematics.py. Các mô-đun xử lý ảnh quản lý kết nối với camera công nghiệp, chạy luồng thu nhận hình ảnh riêng biệt, thực thi mô hình nhận diện YOLOv8 và thực hiện các phép biến đổi tọa độ từ ảnh sang không gian thực. Mô-đun kinematics.py chứa các hàm toán học thuận túy để giải bài toán động học nghịch, áp dụng kỹ thuật tối ưu hóa để đảm bảo các phép tính được thực hiện trong thời gian ngắn nhất (dưới 1 mili-giây).
- Tầng điều khiển giao tiếp bao gồm robot_controller.py, auto_mode_controller.py và connection_manager.py. Trong đó, robot_controller.py đóng vai trò là bộ não trung tâm, điều phối luồng dữ liệu từ hệ thống thị giác sang bộ phận điều khiển động cơ. Mô-đun auto_mode_controller.py chứa logic điều khiển riêng cho chế độ tự động, quản lý hàng đợi các vật thể cần gấp và tính toán thời điểm đồng bộ hóa. Cuối cùng, connection_manager.py là lớp vật lý xử lý việc mở, đóng cổng kết nối và truyền nhận các gói tin dữ liệu xuống vi điều khiển STM32 một cách tin cậy.

4.4.3. Thiết kế giao diện người dùng

Giao diện người dùng được xây dựng dựa trên thư viện PyQt6, áp dụng kiến trúc đa luồng để đảm bảo hiệu năng. Luồng chính của ứng dụng chỉ tập trung vào việc cập nhật hình ảnh và phản hồi thao tác của người dùng, trong khi các tác vụ nặng như xử lý ảnh hay giao tiếp dữ liệu được đẩy xuống các luồng phụ chạy nền.



Hình 4-8: Giao diện điều khiển trên máy tính

Giao diện phần mềm cung cấp đầy đủ các công cụ giám sát và điều khiển cần thiết cho quá trình vận hành.

- Khu vực hiển thị trực quan nằm ở trung tâm, hiển thị video thời gian thực từ camera với các khung bao quanh vật thể đã được nhận diện, kèm theo thông tin về loại vật thể và độ tin cậy của thuật toán.
- Bảng điều khiển vận hành bao gồm các nút chức năng cơ bản như khởi động, dừng, về gốc (Homing) và các nút điều khiển tinh chỉnh vị trí robot theo từng trực (Jogging), giúp người dùng dễ dàng thao tác kiểm tra máy.
- Khu vực cài đặt thông số cho phép người dùng cấu hình các tham số quan trọng như cổng kết nối COM, tọa độ làm việc giới hạn và vận tốc băng tải thực tế.
- Bảng giám sát trạng thái hiển thị liên tục tọa độ hiện tại của đầu công tác robot (X, Y, Z), trạng thái kết nối của các thiết bị phần cứng và nhật ký hoạt động chi tiết của hệ thống để phục vụ việc theo dõi và gỡ lỗi.

4.4.4. Xây dựng dữ liệu và huấn luyện mô hình học sâu

Để hệ thống có khả năng nhận diện chính xác 4 loại hương vị bánh lương khô và xác định đúng góc nghiêng của chúng, quá trình xây dựng mô hình học sâu được thực hiện qua các bước nghiêm ngặt sau:

4.4.4.1. Thu thập dữ liệu

Dữ liệu đầu vào đóng vai trò quyết định đến độ chính xác của mô hình. Nhóm thực hiện đã tiến hành thu thập bộ dữ liệu gồm 200 hình ảnh góc mô tả sản phẩm trong môi trường thực tế.

- Phương pháp thu thập: Sử dụng chính camera của hệ thống để chụp ảnh các viên lương khô đặt trên băng tải đang di chuyển để ảnh được sét với thực tế khi chạy chế độ tự động.
 - Điều kiện chụp: Các bức ảnh được chụp ở nhiều góc độ xoay ngẫu nhiên, thay đổi vị trí trên băng tải và dưới các điều kiện ánh sáng khác nhau (sáng, tối, bóng đèn) để giúp mô hình học được tính tổng quát, tránh hiện tượng học vẹt. Ảnh được đưa vào huấn luyện có độ phân giải 640x640px.
 - Phân bố dữ liệu: 200 ảnh được chia đều cho 4 lớp (classes): Chuối, Dâu, Kiwi, Socola.

4.4.4.2. Gán nhãn dữ liệu với Roboflow

Khác với các bài toán nhận diện thông thường chỉ cần vẽ khung chữ nhật đứng, bài toán gấp sản phẩm hình chữ nhật yêu cầu robot phải biết chính xác góc xoay.

- Công cụ thực hiện: Nhóm sử dụng nền tảng Roboflow, một công cụ mạnh mẽ hỗ trợ quản lý và gán nhãn dữ liệu thị giác máy tính.

- Kỹ thuật gán nhãn: Áp dụng phương pháp Oriented Bounding Box (OBB).

Thay vì vẽ khung bao hình chữ nhật song song với trục ảnh (AABB), nhóm thực hiện vẽ các khung hình chữ nhật có thể xoay, ôm sát biên dạng của từng viên lương khô.

- Ý nghĩa: Việc gán nhãn OBB giúp mô hình không chỉ học được đặc trưng bề mặt để phân loại hương vị, mà còn học được góc nghiêng của vật thể, làm cơ sở để điều khiển động cơ servo xoay đầu hút.

4.4.4.3. Tăng cường dữ liệu

Vì số lượng ảnh gốc (200 ảnh) là khá nhỏ đối với các mô hình học sâu, nhóm đã sử dụng các thuật toán tăng cường dữ liệu tích hợp sẵn trên Roboflow để mở rộng bộ dữ liệu huấn luyện.

Các kỹ thuật áp dụng bao gồm:

- Xoay ảnh: Xoay ảnh ở các góc độ nhất định (90, 180 độ)
- Thay đổi độ sáng: Tăng giảm độ sáng $\pm 15\%$ để mô phỏng điều kiện ánh sáng môi trường thay đổi.
- Nhiều hạt: Thêm nhiều Gaussian để mô hình hoạt động tốt ngay cả khi camera bị nhiễu tín hiệu.

Kết quả sau khi tăng cường, bộ dữ liệu được mở rộng lên khoảng gần 600 ảnh, đảm bảo đủ đa dạng để mô hình hội tụ tốt.

4.4.4.4. Quá trình huấn luyện trên Google Colab

Do yêu cầu về tài nguyên tính toán lớn của quá trình huấn luyện mô hình học sâu, nhóm sử dụng môi trường đám mây Google Colab để tận dụng sức mạnh của GPU Tesla T4 miễn phí.

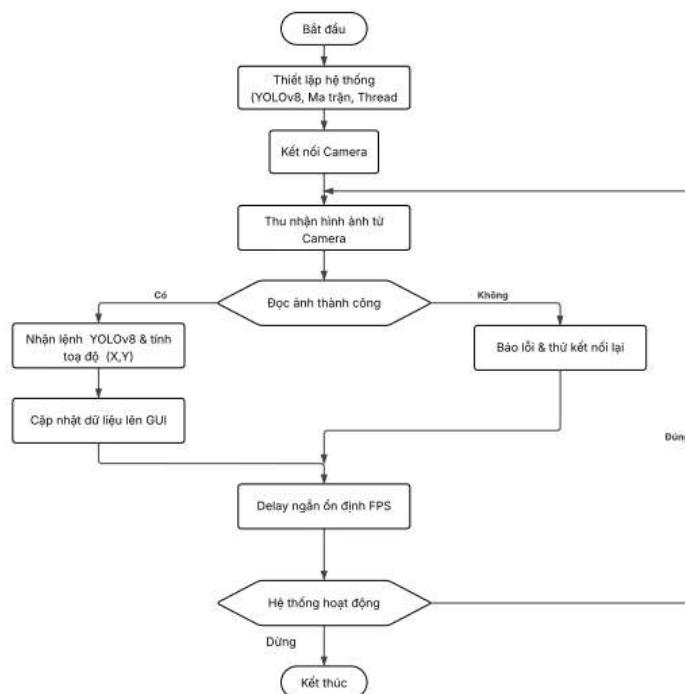
- Framework: Ultralytics YOLOv8.
- Mô hình cơ sở (Pre-trained): `yolov8s-obb.pt` (phiên bản Small - nhỏ sau phiên bản Nano). Nhóm chọn phiên bản Small để tối ưu hóa tốc độ xử lý thời gian thực (FPS cao) nhưng vẫn đảm bảo độ chính xác khi chạy trên máy tính điều khiển.
- Cấu hình tham số huấn luyện (Hyperparameters):
 - Epochs: 100 (Số lần mô hình học lại toàn bộ dữ liệu).
 - Batch size: 16 (Số ảnh xử lý trong một lần cập nhật trọng số).
 - Image size: 640x640 (Độ phân giải đầu vào chuẩn).
 - Optimizer: SGD

4.4.4.5. Kết quả huấn luyện

Sau quá trình huấn luyện kéo dài khoảng 1-2 giờ trên Google Colab, hệ thống xuất ra tệp trọng số tốt nhất ('best.pt'). Các chỉ số đánh giá như mAP50-95 (Mean Average Precision) cho thấy mô hình đạt độ chính xác cao (hơn 90%) trong việc phân loại và định vị góc xoay. Tệp trọng số này sau đó được tải về máy tính và tích hợp vào module xử lý ảnh của phần mềm điều khiển.

4.4.5. Mô-đun thị giác máy tính và nhận diện

Mô-đun thị giác máy tính được xây dựng dựa trên tệp mã nguồn vision_system.py, đóng vai trò là đôi mắt của hệ thống robot. Để đảm bảo khả năng xử lý thời gian thực và độ chính xác cao, quy trình xử lý ảnh được thiết kế theo một lưu đồ thuật toán chặt chẽ, hoạt động trên một luồng riêng biệt song song với giao diện chính.



Hình 4-9: Lưu đồ xử lý thị giác của hệ thống

Quy trình thuật toán được mô tả chi tiết qua các bước sau:

- Bước khởi tạo và kết nối: Khi hệ thống bắt đầu, chương trình sẽ khởi tạo một luồng xử lý video riêng biệt. Tại đây, mô hình trí tuệ nhân tạo YOLOv8 và các ma trận biến đổi tọa độ (được lưu từ quá trình hiệu chuẩn trước đó) sẽ được nạp vào bộ nhớ RAM. Sau đó, hệ thống tiến hành kết nối với camera công nghiệp thông qua giao thức tối ưu để đảm bảo độ trễ thấp nhất.
- Vòng lặp thu nhận và xử lý: Hệ thống đi vào một vòng lặp vô hạn để liên tục thu nhận các khung hình từ camera. Mỗi bức ảnh thu được sẽ trải qua bước tiền xử lý để điều chỉnh kích thước và chuẩn hóa màu sắc phù hợp với đầu vào

của mô hình học sâu.

- Giai đoạn nhận diện: Bức ảnh sau khi xử lý được đưa qua mạng nơ-ron tích chập YOLOv8. Kết quả đầu ra là danh sách các vật thể phát hiện được, bao gồm tọa độ khung bao hình chữ nhật, loại vật thể và độ tin cậy của dự đoán. Để loại bỏ nhiễu và các nhận diện sai, thuật toán sẽ lọc bỏ tất cả các kết quả có độ tin cậy thấp hơn một ngưỡng cài đặt trước (ví dụ 70%).

```
def process_frame(self, frame, draw=True):
    # Cắt ảnh (ROI) để tối ưu tốc độ xử lý
    crop_size = 640
    x_start = max(0, frame.shape[1] - crop_size)
    crop_frame = frame[:, x_start:]

    # Thực thi mô hình (Inference)
    results = self.model.predict(crop_frame, verbose=False, conf=0.7)
    result = results[0]

    detections = []
    if result.obb is not None: # Xử lý Oriented Bounding Box
        for obb in result.obb:
            # Lấy thông tin tọa độ, góc xoay, loại vật thể
            r = obb.xywhr[0].cpu().numpy()
            cx, cy, w, h, rot_rad = r

            # Chuyển đổi góc radian sang độ
            angle_deg = np.degrees(rot_rad)

            detection = {
                'bbox': [int(cx), int(cy), int(w), int(h)],
                'angle': angle_deg,
                'class_name': result.names[int(obb.cls)],
                'confidence': float(obb.conf)
            }
            detections.append(detection)

    return frame, detections
```

- Giai đoạn chuyển đổi tọa độ: Đây là bước quan trọng nhất để robot có thể tương tác với vật thể. Tọa độ tâm của vật thể trên ảnh (đơn vị điểm ảnh) sẽ được nhân với ma trận biến đổi đồng nhất (Homography Matrix). Phép toán đại số tuyến tính này sẽ ánh xạ vị trí từ không gian ảnh 2D sang không gian làm việc thực tế của robot (đơn vị mm), giúp robot xác định chính xác vị trí gấp trên băng tải.

```
def pixel_to_mm(self, pixel_x, pixel_y):
    if self._pixel_to_mm_matrix is None:
        return None
```

```
# Biểu diễn điểm dưới dạng vector đồng nhất [x, y, 1]
point = np.array([pixel_x, pixel_y, 1.0], dtype=np.float64)

# Nhân với ma trận biến đổi
result = self._pixel_to_mm_matrix @ point

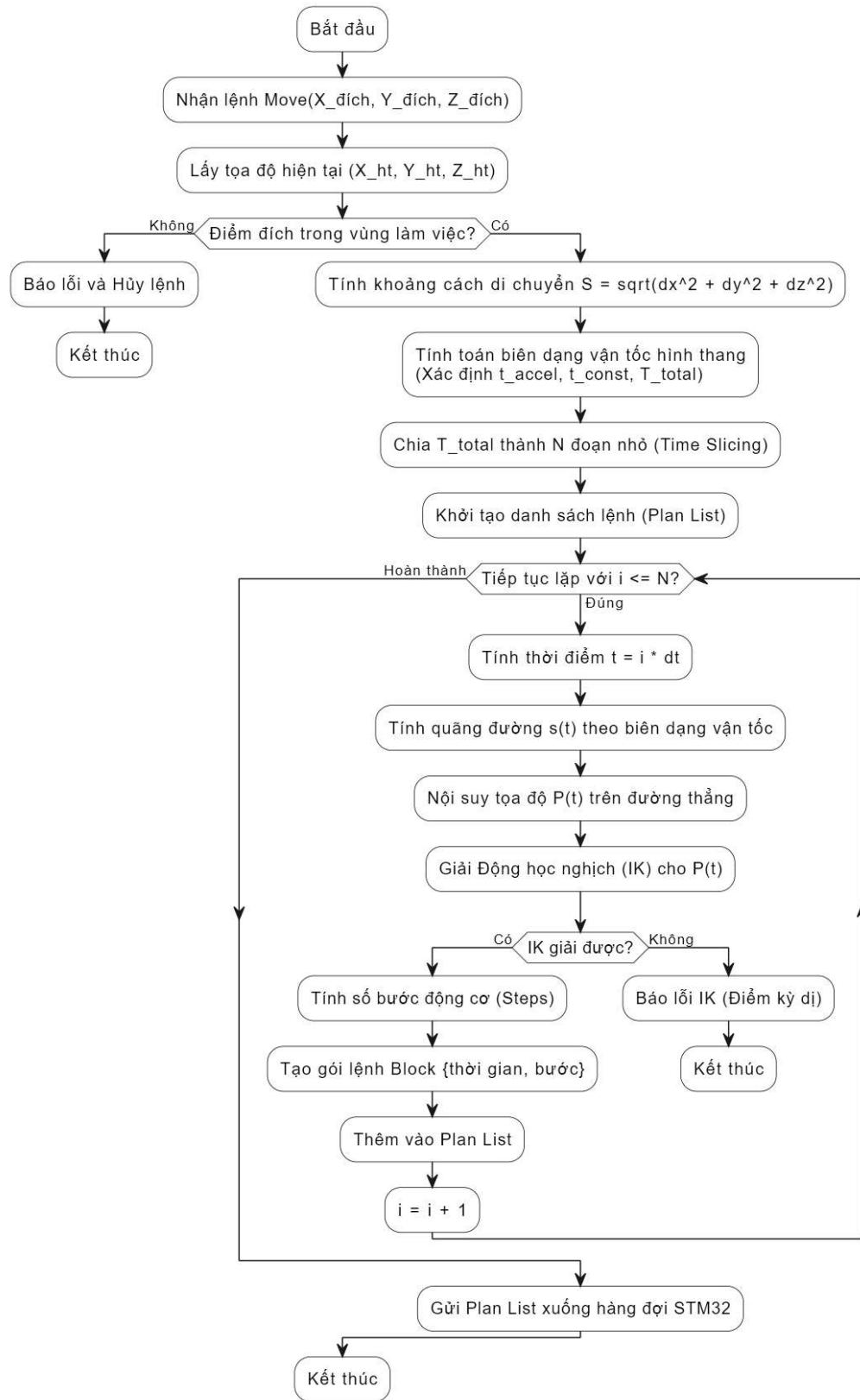
# Chuẩn hóa vector kết quả
robot_x = result[0] / result[2]
robot_y = result[1] / result[2]

return (robot_x, robot_y)
```

- Đóng gói và hiển thị: Cuối cùng, dữ liệu tọa độ thực tế cùng với loại vật thể sẽ được đóng gói và đẩy vào một hàng đợi an toàn để chuyển sang mô-đun điều khiển robot. Đồng thời, hệ thống sẽ vẽ các khung bao và thông tin lên hình ảnh gốc rồi gửi tín hiệu cập nhật lên giao diện người dùng, giúp người vận hành giám sát trực quan quá trình hoạt động.

4.4.6. Mô-đun tính toán động học và quy hoạch quỹ đạo

Mô-đun động học và quy hoạch quỹ đạo đóng vai trò quyết định đến độ chính xác và độ mượt mà trong chuyển động của robot. Để giải quyết bài toán gấp và đặt, robot Delta di chuyển qua các điểm trong không gian, hệ thống không gửi trực tiếp tọa độ đích mà thực hiện một quy trình tính toán phức tạp gồm nhiều bước.



Hình 4-10: Lưu đồ thuật toán quy hoạch quỹ đạo di chuyển

Quy trình xử lý chi tiết:

- Đầu tiên là bước Tiếp nhận và Kiểm tra hợp lệ. Khi nhận được yêu cầu di chuyển đến tọa độ đích từ bộ điều khiển trung tâm, hệ thống sẽ kiểm tra xem tọa độ này có nằm trong vùng làm việc an toàn của robot hay không. Nếu điểm

đích nằm ngoài tầm với hoặc có nguy cơ va chạm, lệnh sẽ bị hủy ngay lập tức để bảo vệ phần cứng.

2. Tiếp theo là Tính toán biên dạng vận tốc. Hệ thống tính toán khoảng cách Euclidean giữa điểm hiện tại và điểm đích. Dựa trên các thông số gia tốc và vận tốc tối đa đã cài đặt, thuật toán sẽ xây dựng một biên dạng vận tốc hình thang. Biên dạng này xác định rõ thời gian cần thiết cho giai đoạn tăng tốc, giai đoạn di chuyển đều và giai đoạn giảm tốc, đảm bảo tổng thời gian di chuyển là tối ưu nhất.
3. Bước quan trọng nhất là Chia nhỏ quỹ đạo theo thời gian (Time Slicing). Tổng thời gian di chuyển sẽ được chia thành hàng trăm lát cắt nhỏ, mỗi lát cắt tương ứng với một khoảng thời gian cố định (ví dụ 15 mili-giây).
4. Tại mỗi lát cắt thời gian, hệ thống thực hiện vòng lặp tính toán nội suy. Dựa vào biên dạng vận tốc, chương trình xác định quãng đường $s(t)$ mà robot đã đi được tại thời điểm t . Từ đó, tọa độ không gian tức thời $P(t)$ được nội suy tuyến tính nằm trên đường thẳng nối điểm đầu và điểm cuối.
5. Ngay sau đó, hàm Động học nghịch (được tối ưu hóa bằng Numba) sẽ được gọi để chuyển đổi tọa độ $P(t)$ thành góc quay của ba khớp động cơ. Kết quả là số bước xung cụ thể cần phát cho từng động cơ trong khoảng thời gian 15ms đó. Các dữ liệu này được đóng gói thành một cấu trúc Block lệnh.
6. Cuối cùng, toàn bộ danh sách các Block lệnh sẽ được đẩy xuống hàng đợi truyền thông để gửi dần xuống vi điều khiển STM32, giúp robot thực thi chuyển động một cách liên tục và mượt mà.

```
# Trích đoạn logic từ lớp MotionPlannerTrapezoidal
def plan_cartesian_move_time_sliced(self, start_pos, end_pos, ...):
    # 1. Tính biên dạng vận tốc hình thang
    params = self._compute_profile_parameters(distance, v_max, accel)

    # 2. Chia nhỏ quỹ đạo thành các lát cắt
    num_slices = int(params["total_time"] / self.SEGMENT_TIME)

    plan = []
    for i in range(1, num_slices + 1):
        t_current = i * self.SEGMENT_TIME

        # 3. Tính quãng đường đi được tại thời điểm t
        dist_at_t = self._calculate_position_at_t(t_current, params)

        # 4. Nội suy tọa độ (x, y, z) hiện tại trên đường thẳng
        ratio = dist_at_t / total_distance
        current_x = start_x + (end_x - start_x) * ratio

        # 5. Tính động học nghịch và tạo lệnh điều khiển
```

```

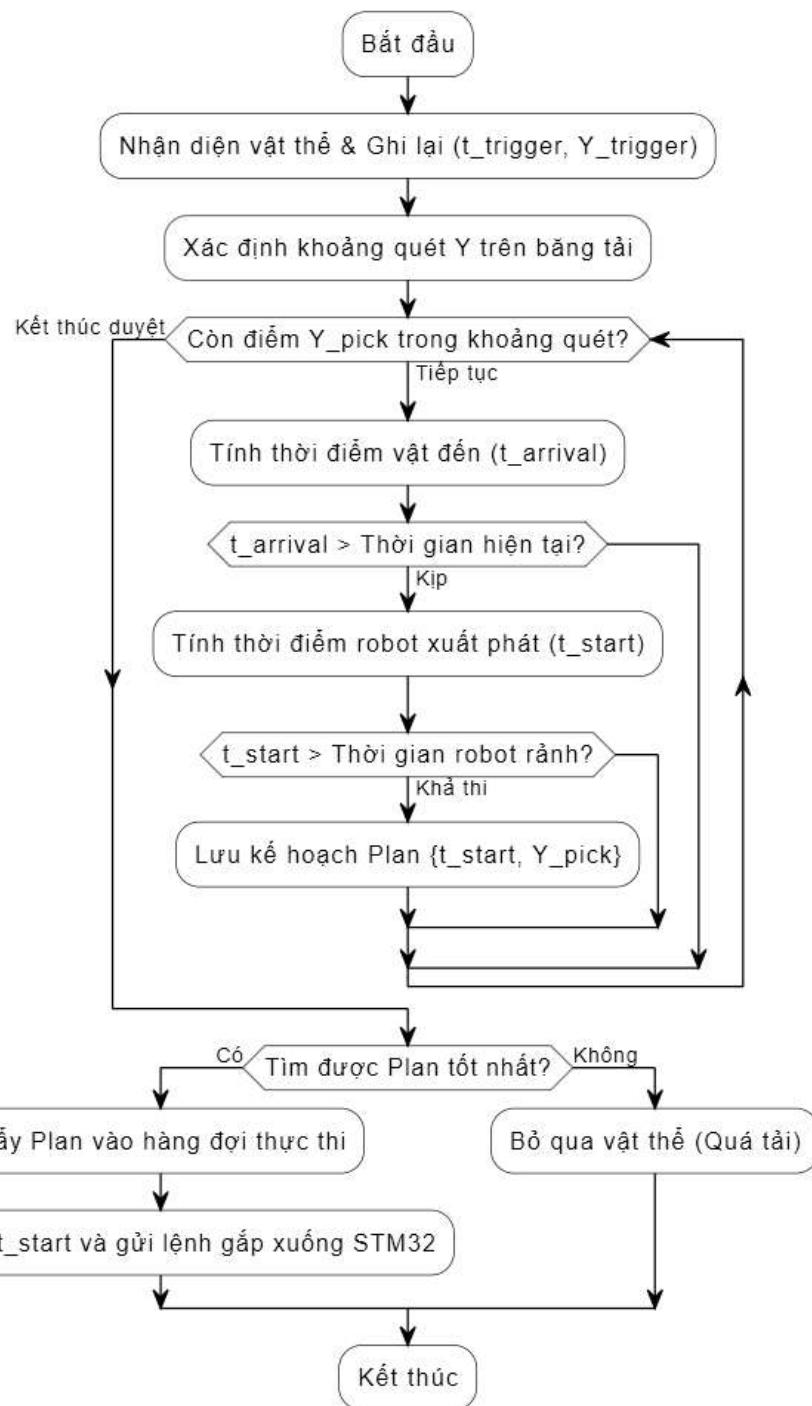
angles = kinematics.inverse_kinematics(current_x, ... )
plan.append({"t": segment_time, "s": angles_to_steps(angles)})

return plan

```

4.4.7. Chiến lược điều khiển và logic tự động

Chiến lược điều khiển tự động là phần phức tạp nhất của hệ thống, được cài đặt trong tệp `auto_mode_controller.py`. Hệ thống sử dụng kỹ thuật đồng bộ hóa dựa trên thời gian để giải quyết bài toán bắt vật thể động trên băng tải mà không cần sử dụng encoder phản hồi vị trí băng tải.



Hình 4-11: Lưu đồ thuật toán lập lịch gấp và đồng bộ hóa

Công nghệ cốt lõi và Quy trình xử lý:

1. Định thời chính xác:

Thay vì xử lý tức thời ngay khi camera phát hiện vật, hệ thống ghi lại chính xác thời điểm xuất hiện (`t_trigger`) sử dụng đồng hồ đơn điệu `time.perf_counter()`. Vị trí của vật thể tại bất kỳ thời điểm tương lai nào đều được dự đoán dựa trên công thức:

$$Y(t) = Y_{\text{trigger}} + V_{\text{băng_tải}} * (t - t_{\text{trigger}})$$

Điều này cho phép hệ thống biết trước vị trí của vật thể ngay cả khi vật đã trôi ra khỏi tầm nhìn của camera.

2. Thuật toán Tìm kiếm điểm đón:

Để robot bắt kịp vật thể, bộ điều khiển phải giải phương trình: *Thời điểm robot đến điểm P = Thời điểm vật thể đến điểm P*.

Hệ thống sử dụng thuật toán quét lưới dọc theo chiều dài băng tải (từ y = {-95:75} là vùng mà robot hoạt động ổn định nhất). Tại mỗi điểm ứng viên, chương trình mô phỏng ngược thời gian di chuyển của robot (sử dụng Motion Planner) để kiểm tra xem robot có kịp di chuyển từ vị trí hiện tại đến đó hay không. Điểm khả thi đầu tiên tìm được sẽ được chọn làm điểm đón

3. Lập lịch thông minh:

Hệ thống duy trì một biến trạng thái `robot_next_free_perf` để biết khi nào robot sẽ hoàn thành tác vụ hiện tại. Khi có vật thể mới, thuật toán sẽ tính toán thời điểm xuất phát `t_start` sao cho `t_start >= robot_next_free_perf`. Điều này đảm bảo robot hoạt động liên tục, gói đầu các tác vụ lên nhau mà không bị xung đột, tối ưu hóa năng suất gấp.

4. Bù trừ độ trễ:

Trong môi trường thực tế, luôn có độ trễ do xử lý ảnh, truyền tin USB và quán tính cơ khí. Hệ thống tích hợp một tham số bù trễ (khoảng 100-200ms) vào công thức tính toán thời gian xuất phát, đảm bảo đầu hút đến điểm gấp sớm hơn một chút để đón đầu vật thể chính xác.

Đoạn mã minh họa thuật toán lập lịch:

```
```python
Trích đoạn logic từ AutoModeController
def _schedule_pick_candidates(self):
 # Quét dọc theo trục Y để tìm điểm gấp khả thi
 y_range = range(self.y_pick_min, self.y_pick_max, self.y_pick_step)
```

```
for pick_y in y_range:
 # 1. Tính thời gian vật thể trôi đến điểm pick_y
 dist_obj_travel = pick_y - trigger_y
 time_travel = abs(dist_obj_travel) / self.conveyor_speed_mm_s
 t_arrival = t_trigger + time_travel # Thời điểm Gặp Nhau

 # 2. Mô phỏng thời gian robot di chuyển
 t_robot_move = self._simulate_robot_move_time(current_pos, (robot_x, pick_y,
 z_pick))

 # 3. Tính thời điểm robot cần xuất phát (có bù trễ)
 t_required_start = t_arrival - t_robot_move - LATENCY_COMPENSATION

 # 4. Kiểm tra tính khả thi: Robot có kịp xuất phát không?
 if t_required_start >= self.robot_next_free_perf:
 # Tìm thấy phương án khả thi!
 best_plan = {
 'pick_y': pick_y,
 't_start_action': t_required_start,
 't_arrival': t_arrival
 }
 # Cập nhật thời gian bận của robot cho vật tiếp theo
 self.robot_next_free_perf = t_arrival + estimate_drop_time()
 break
```

#### 4.4.8. Quản lý trạng thái và xử lý lỗi

Để đảm bảo hệ thống hoạt động ổn định và tin cậy trong môi trường công nghiệp, phần mềm tích hợp các cơ chế quản lý trạng thái và xử lý lỗi đa tầng.

##### 4.4.8.1. Cơ chế điều khiển luồng cửa sổ trượt

Do bộ nhớ đệm trên vi điều khiển stm32f103c8t6 có hạn, máy tính không thể gửi toàn bộ hàng trăm lệnh di chuyển của một quỹ đạo dài xuống cùng lúc. Để giải quyết vấn đề này, phần mềm sử dụng cơ chế cửa sổ trượt để đồng bộ hóa dữ liệu. Phần mềm trên máy tính duy trì một bộ đệm lệnh ảo. Khi bắt đầu di chuyển, máy tính sẽ gửi trước một gói lệnh gồm khoảng 16 đến 32 lệnh xuống vi điều khiển. Sau đó, mỗi khi vi điều khiển thực hiện xong một lệnh, nó sẽ gửi tín hiệu phản hồi ngược về máy tính. Nhận được tín hiệu này, phần mềm sẽ lập tức gửi bổ sung một lệnh tiếp theo từ hàng đợi xuống để lấp vào chỗ trống. Cơ chế này đảm bảo bộ đệm trên vi điều khiển luôn đầy ở mức an toàn, giúp robot di chuyển liên tục mà không bị ngắt quãng do chờ dữ liệu hoặc tràn bộ nhớ.

##### 4.4.8.2. Giám sát kết nối và bộ định thời gian giám sát

Hệ thống liên tục giám sát trạng thái kết nối giữa máy tính và các thiết bị ngoại vi. Nếu tín hiệu từ camera bị mất hoặc cổng usb bị ngắt kết nối đột ngột, phần mềm sẽ tự động kích hoạt quy trình xử lý lỗi gồm dừng băng tải, đưa robot về trạng thái an toàn và thử kết nối lại. Đối với các tác vụ quan trọng như về gốc tọa độ, hệ thống sử dụng một

bộ định thời gian giám sát. Nếu robot không hoàn thành việc về gốc trong thời gian quy định, phần mềm sẽ nhận định có sự cố cơ khí và lập tức dừng hệ thống để bảo vệ động cơ.

#### 4.4.8.3. Các lớp bảo vệ an toàn

Ngoài nút dừng khẩn cấp vật lý, phần mềm còn thiết lập các giới hạn mềm cho vùng làm việc. Trước khi gửi bất kỳ lệnh di chuyển nào, thuật toán sẽ kiểm tra xem tọa độ đích có nằm trong vùng làm việc an toàn hay không. Nếu phát hiện lệnh di chuyển ra ngoài giới hạn cho phép, phần mềm sẽ từ chối thực thi và cảnh báo người vận hành, ngăn ngừa nguy cơ va chạm cơ khí.

#### 4.4.9. Quy trình hiệu chuẩn hệ thống

Quy trình hiệu chuẩn là bước bắt buộc để đảm bảo sự đồng bộ giữa hệ thống thị giác và cơ cấu chấp hành của robot. Quy trình này bao gồm hai giai đoạn chính được thực hiện thông qua giao diện phần mềm.

Giai đoạn thứ nhất là hiệu chuẩn ống kính camera. Hình ảnh thu được từ camera thường bị biến dạng cong ở các góc do đặc tính của thấu kính. Phần mềm sử dụng các tham số hình học được tính toán trước để nắn chỉnh hình ảnh về dạng phẳng, đảm bảo tỉ lệ xích trên toàn bộ khung hình là đồng nhất.

Giai đoạn thứ hai là hiệu chuẩn không gian để thiết lập mối quan hệ giữa điểm ảnh và milimet thực tế. Người vận hành sẽ xác định bốn điểm chuẩn tương ứng với bốn góc của vùng làm việc trên băng tải. Tọa độ điểm ảnh của các điểm này được ghi nhận cùng với tọa độ thực tế mà robot có thể chạm tới. Dựa trên dữ liệu này, phần mềm sử dụng thuật toán ánh xạ đồng nhất để tạo ra ma trận biến đổi tọa độ. Ma trận này cho phép chuyển đổi trực tiếp vị trí tâm vật thể từ đơn vị điểm ảnh sang tọa độ không gian chính xác mà robot cần di chuyển tới để thực hiện thao tác gấp.

## CHƯƠNG 5: THỬ NGHIỆM ĐÁNH GIÁ KẾT QUẢ

Sau khi hoàn thành việc chế tạo mô hình và xây dựng chương trình điều khiển, nhóm thực hiện tiến hành các đợt thử nghiệm thực tế để đánh giá hiệu suất của hệ thống.

### 5.1. Thiết lập môi trường thực nghiệm

Quá trình thực nghiệm được thực hiện trên mô hình thực tế với các điều kiện cố định. Hệ thống camera được lắp ở độ cao 450 milimet so với mặt băng tải. Nguồn sáng được bố trí đồng đều để giảm thiểu bóng đổ gây nhiễu cho thuật toán nhận diện. Các sản phẩm thử nghiệm bao gồm bốn loại mô hình bánh lương khô với 4 vị là chuối, dâu, kiwi và socola. Băng tải được thiết lập vận hành ở dài tốc độ từ 30 đến 60 milimet trên giây để kiểm tra khả năng đáp ứng thời gian thực của hệ thống.

### 5.2. Đánh giá hệ thống nhận diện vật thể

Mô hình học sâu yolov8 được đánh giá dựa trên khả năng nhận diện chính xác loại vật thể và vị trí của chúng khi đang di chuyển trên băng tải.

- Độ chính xác nhận diện: Qua quá trình chạy thử với 100 mẫu vật khác nhau, mô hình đạt độ chính xác trung bình trên 90 phần trăm. Một số sai số nhỏ xảy ra khi các vật thể nằm quá sát nhau hoặc bị phản xạ ánh sáng mạnh từ bề mặt băng tải.
- Tốc độ xử lý ảnh: Nhờ việc tối ưu hóa mã nguồn và sử dụng luồng xử lý riêng, tốc độ nhận diện đạt mức 22 khung hình trên giây. Thời gian xử lý từ lúc camera thu nhận ảnh đến khi có tọa độ vật thể dao động trong khoảng 30 đến 50 mili-giây, hoàn toàn đáp ứng tốt yêu cầu điều khiển thời gian thực.

### 5.3. Đánh giá khả năng điều khiển và độ chính xác gấp thża

Khả năng điều khiển được đánh giá dựa trên sự phối hợp giữa quy hoạch quỹ đạo và đồng bộ hóa thời gian.

- Độ chính xác vị trí: Sai số vị trí tĩnh của đầu gấp sau khi về gốc tọa độ nằm trong khoảng dưới 3 milimet. Khi vận hành ở chế độ tự động, nhờ kỹ thuật nội suy biến dạng hình thang, đầu gấp di chuyển mượt mà và tiếp cận vật thể chính xác theo đường thẳng.
- Mặc dù hộp số động cơ có độ rõ nhất định nhưng không ảnh hưởng đến độ chính xác robot
- Hiệu quả đồng bộ hóa: Thuật toán dự đoán điểm đón dựa trên thời gian thực tế hoạt động ổn định. Tại tốc độ băng tải 40 milimet trên giây, robot có thể xác định và gấp trúng vật thể với tỉ lệ thành công cao. Sai số xảy ra chủ yếu do quán tính của vật thể khi trượt trên băng tải hoặc sự thay đổi đột ngột của tốc

độ động cơ băng tải.

#### **5.4. Đánh giá hiệu suất tổng thể hệ thống**

Hệ thống được vận hành liên tục để kiểm tra tính ổn định và năng suất phân loại.

##### **Năng suất hoạt động:**

Trong điều kiện vận hành tối ưu, hệ thống đạt năng suất gấp và phân loại khoảng 20 đến 25 sản phẩm trong một phút. Chu kỳ của một lần gấp thả bao gồm di chuyển đến điểm đón, hạ xuống gấp, nhắc lên, di chuyển đến khay chứa và quay về vị trí chờ mót khoảng 2,5 giây.

##### **Tỉ lệ phân loại đúng:**

Tỉ lệ gấp và thả đúng khay quy định đạt trên 85 phần trăm. Các trường hợp thất bại thường rơi vào tình huống vật thể nằm ngoài vùng làm việc an toàn của robot hoặc có quá nhiều vật thể xuất hiện cùng lúc trên băng tải dẫn đến quá tải bộ đệm lệnh.

Tổng kết lại, mô hình robot delta kết hợp xử lý ảnh học sâu đã hoạt động đúng theo mục tiêu đề ra, minh chứng được tính hiệu quả của các thuật toán quy hoạch quỹ đạo và đồng bộ hóa thời gian thực trong bài toán tự động hóa sản xuất.

## HƯỚNG PHÁT TRIỂN ĐỀ TÀI

Mặc dù hệ thống hiện tại đã đáp ứng được các mục tiêu cơ bản về phân loại sản phẩm sử dụng robot Delta kết hợp thị giác máy, nhưng để nâng cao hiệu suất và khả năng ứng dụng trong môi trường công nghiệp thực tế, nhóm thực hiện đề xuất các hướng phát triển trong tương lai như sau:

### 1. Nâng cấp về Thuật toán Điều khiển và Quy hoạch quỹ đạo

Tích hợp bộ điều khiển quỹ đạo S-Curve: Hiện tại hệ thống đang sử dụng biên dạng vận tốc hình thang. Trong tương lai, có thể áp dụng biên dạng S-Curve để kiểm soát độ giật, giúp robot di chuyển mượt mà hơn, giảm rung động cơ khí và tăng tuổi thọ thiết bị khi hoạt động ở tốc độ cao.

Tối ưu hóa bài toán lập lịch (Scheduling Optimization): Khi mật độ sản phẩm trên băng tải quá dày đặc, thuật toán hiện tại có thể bỏ sót vật thể. Cần nghiên cứu áp dụng các thuật toán tối ưu hóa (như giải thuật tham lam hoặc quy hoạch động) để robot chọn thứ tự gấp tối ưu nhất, giảm thiểu quãng đường di chuyển thừa.

### 2. Cải tiến hệ thống Thị giác máy và Trí tuệ nhân tạo

Tích hợp Camera 3D (Stereo Vision hoặc Depth Camera): Hệ thống 2D hiện tại gặp khó khăn khi xác định chiều cao vật thể hoặc khi các vật thể bị xếp chồng lên nhau. Việc sử dụng Camera 3D sẽ giúp robot xác định chính xác tọa độ (x, y, z) và hướng gấp trong không gian 3 chiều.

Kiểm tra chất lượng sản phẩm (QA/QC): Mở rộng mô hình học sâu (Deep Learning) không chỉ để phân loại loại bánh mà còn để phát hiện các lỗi ngoại quan như: bánh bị vỡ, cháy, méo mó hoặc thiếu tem nhãn, từ đó loại bỏ các sản phẩm không đạt chuẩn.

### 3. Tối ưu hóa Phần cứng và Hệ thống nhúng

Chuyển đổi sang hệ thống nhúng Edge AI: Thay vì phụ thuộc vào máy tính cá nhân (PC/Laptop), hệ thống có thể được tích hợp lên các bo mạch nhúng mạnh mẽ như NVIDIA Jetson Nano hoặc Jetson Orin. Điều này giúp thu gọn kích thước tủ điện, giảm chi phí và biến robot thành một thiết bị hoạt động độc lập.

Thiết kế đầu gấp linh hoạt (Soft Gripper): Nghiên cứu các loại tay gấp dạng mềm hoặc cơ cấu kẹp thích ứng để có thể gấp được đa dạng các loại sản phẩm có hình dáng phức tạp hoặc dễ vỡ mà không cần thay đầu hút.

### 4. Tích hợp IoT và Hệ thống giám sát (SCADA)

Xây dựng giao diện Web/App giám sát từ xa: Phát triển hệ thống IoT cho phép người quản lý theo dõi năng suất, trạng thái hoạt động và cảnh báo lỗi của robot thông qua

thiết bị di động hoặc trình duyệt web.

Lưu trữ và phân tích dữ liệu: Tích hợp cơ sở dữ liệu (Database) để lưu lại lịch sử sản xuất, số lượng sản phẩm đạt/lỗi theo thời gian thực, phục vụ cho việc báo cáo và phân tích hiệu quả sản xuất.

## TÀI LIỆU THAM KHẢO

- [1] R. Clavel, "Device for the movement and positioning of an element in space," U.S. Patent 4 976 582, Dec. 11, 1990.
- [2] L. W. Tsai, *Robot Analysis: The Mechanics of Serial and Parallel Manipulators*. New York, NY, USA: Wiley, 1999
- [3] R. L. Williams II, "The Delta Parallel Robot: Kinematics Solutions," *Mechanical Engineering Faculty Publications*, vol. 18, 2016
- [4] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 4th ed. New York, NY, USA: Pearson, 2018.
- [5] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.
- [6] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Las Vegas, NV, USA, 2016, pp. 779–788.
- [8] STMicroelectronics, "STM32F103x8/B Datasheet - Medium-density performance line ARM-based 32-bit MCU," 2015. [Online]. Available: <https://www.st.com>.

## PHỤ LỤC 1: LẬP TRÌNH ĐỘNG HỌC

Chương trình trên file kinematics.py

```
import math
import numpy as np
from numba import jit
import constants as C

=====
NUMBA-OPTIMIZED CORE FUNCTIONS (20-100x faster)
=====
Numba không hỗ trợ methods với 'self', nên tách thành static functions
Các hằng số phải truyền vào như parameters

@jit(nopython=True, cache=True)
def _calc_angle_yz_numba(x0, y0, z0, tan30, f, e, rf, re, pi):
 """
 Numba-optimized: Tính góc theta cho 1 cánh tay.
 Tốc độ: ~50-100x nhanh hơn Python thuần.
 """

 # Hằng số f/2 * tan30
 y1 = -0.5 * tan30 * f

 # Trừ offset effector
 y0_mod = y0 - 0.5 * tan30 * e

 if abs(z0) < 1e-9:
 return np.nan # Tránh chia cho 0

 # z = a + b*y
 a = (x0**2 + y0_mod**2 + z0**2 + rf**2 - re**2 - y1**2) / (2.0 * z0)
 b = (y1 - y0_mod) / z0

 # discriminant
 d = -(a+b*y1)**2 + rf*(b*b*rf + rf)

 # Nếu d < 0, điểm không tồn tại
 if d < 1e-9:
 return np.nan

 # choosing outer point
 yj = (y1 - a * b - math.sqrt(d)) / (b**2 + 1.0)
 zj = a + b * yj

 # Ngăn chia cho 0 nếu yj == y1
 if abs(y1 - yj) < 1e-9:
 if zj < 0:
 theta_deg = -90.0
 else:
 theta_deg = 90.0
```

```
else:
 theta_rad = math.atan(-zj / (y1 - yj))
 theta_deg = 180.0 * theta_rad / pi

 # Xử lý góc phần tư
 if yj > y1:
 theta_deg += 180.0

return theta_deg

@jit(nopython=True, cache=True)
def inverse_kinematics_numba(x0, y0, z0, tan30, f, e, rf, re, pi, sqrt3,
 cos120, sin120, arm_angle_min, arm_angle_max):
 """
 Numba-optimized: Tính động học nghịch cho TÂM BÊ.
 Trả về (theta1, theta2, theta3) hoặc (nan, nan, nan) nếu thất bại.
 """

 # Đảo dấu X để phù hợp với hệ tọa độ robot
 x0_corrected = x0
 y0_corrected = -y0

 # Cánh tay 1
 theta1 = _calc_angle_yz_numba(x0_corrected, y0_corrected, z0,
 tan30, f, e, rf, re, pi)

 # Cánh tay 2 (xoay -120 độ)
 x_rot2 = x0_corrected * cos120 + y0_corrected * sin120
 y_rot2 = y0_corrected * cos120 - x0_corrected * sin120
 theta2 = _calc_angle_yz_numba(x_rot2, y_rot2, z0,
 tan30, f, e, rf, re, pi)

 # Cánh tay 3 (xoay +120 độ)
 x_rot3 = x0_corrected * cos120 - y0_corrected * sin120
 y_rot3 = y0_corrected * cos120 + x0_corrected * sin120
 theta3 = _calc_angle_yz_numba(x_rot3, y_rot3, z0,
 tan30, f, e, rf, re, pi)

 # Kiểm tra thất bại
 if np.isnan(theta1) or np.isnan(theta2) or np.isnan(theta3):
 return np.nan, np.nan, np.nan

 # Kiểm tra giới hạn góc
 if not (arm_angle_min <= theta1 <= arm_angle_max):
 return np.nan, np.nan, np.nan
 if not (arm_angle_min <= theta2 <= arm_angle_max):
 return np.nan, np.nan, np.nan
 if not (arm_angle_min <= theta3 <= arm_angle_max):
 return np.nan, np.nan, np.nan

return theta1, theta2, theta3
```

```
@jit(nopython=True, cache=True)
def inverse_kinematics_tool_numba(xt, yt, zt, alpha_deg,
 tool_offset_radius, tool_offset_z,
 tan30, f, e, rf, re, pi, sqrt3,
 cos120, sin120, arm_angle_min,
 arm_angle_max):
 """
 Numba-optimized: Tính động học nghịch cho ĐẦU HÚT với tool offset.
 """
 alpha_rad = math.radians(alpha_deg)

 # Tính (x0, y0, z0) của tâm bệ TÙ (xt, yt, zt) của đầu hút
 x0 = xt - tool_offset_radius * math.cos(alpha_rad)
 y0 = yt - tool_offset_radius * math.sin(alpha_rad)
 z0 = zt + tool_offset_z

 # Gọi hàm IK chuẩn
 return inverse_kinematics_numba(x0, y0, z0, tan30, f, e, rf, re, pi,
 sqrt3,
 cos120, sin120, arm_angle_min,
 arm_angle_max)

=====
DELTA KINEMATICS CLASS (Wrapper cho Numba functions)
=====

class DeltaKinematics:

 def __init__(self, e_side=None, f_side=None, re_lower=None, rf_upper=None,
 tool_offset_radius=15, tool_offset_z=53):
 """
 Khởi tạo với các thông số hình học của robot.
 Nếu không truyền tham số, sẽ lấy mặc định từ constants.py
 """
 # Lấy từ tham số hoặc constants
 self.e = e_side if e_side is not None else C.ROBOT_E_SIDE
 self.f = f_side if f_side is not None else C.ROBOT_F_SIDE
 self.re = re_lower if re_lower is not None else C.ROBOT_RE_LOWER
 self.rf = rf_upper if rf_upper is not None else C.ROBOT_RF_UPPER

 # Hằng số lượng giác
 self.sqrt3 = math.sqrt(3.0)
 self.pi = math.pi
 self.sin120 = self.sqrt3 / 2.0
 self.cos120 = -0.5
 self.tan30 = 1.0 / self.sqrt3
```

```
Hằng số 't' từ C++ forward kinematics
self.t = (self.f - self.e) * self.tan30 / 2.0
self.dtr = self.pi / 180.0 # Độ sang Radian

Tool offset (QUAN TRỌNG)
self.tool_offset_radius = tool_offset_radius
self.tool_offset_z = tool_offset_z

Hằng số động cơ bước - Lấy từ constants.py để dễ calib
self.gear_ratio = C.MOTOR_GEAR_RATIO
self.steps_per_revolution = C.MOTOR_STEPS_PER_REV

Độ phân giải cuối cùng: số xung động cơ để quay cánh tay robot 1 độ.
self.steps_per_arm_degree = (self.steps_per_revolution *
self.gear_ratio) / 360.0

def warmup(self):
 """
 Kích hoạt JIT compilation (warm-up) cho các hàm Numba bằng cách gọi
 chúng với dữ liệu mẫu.
 Giúp tránh lag ở lần chạy đầu tiên.
 """
 # Dummy coordinates (gần vị trí Home)
 x, y, z = 0.0, 0.0, -380.0
 alpha = -90.0

 # Warmup inverse_kinematics_numba
 inverse_kinematics_numba(
 x, y, z,
 self.tan30, self.f, self.e, self.rf, self.re, self.pi, self.sqrt3,
 self.cos120, self.sin120, C.ARM_ANGLE_MIN, C.ARM_ANGLE_MAX
)

 # Warmup inverse_kinematics_tool_numba
 inverse_kinematics_tool_numba(
 x, y, z, alpha,
 self.tool_offset_radius, self.tool_offset_z,
 self.tan30, self.f, self.e, self.rf, self.re, self.pi, self.sqrt3,
 self.cos120, self.sin120, C.ARM_ANGLE_MIN, C.ARM_ANGLE_MAX
)

 # Warmup _calc_angle_yz_numba implicitly via above calls

 print("☑ DeltaKinematics: Numba functions warmed up!")

def forward_kinematics(self, theta1, theta2, theta3):
 """
 Tính toán tọa độ TÂM BỆ DI ĐỘNG (x0, y0, z0) từ các góc động cơ.
 Logic dựa trên delta_calcForward.

 :return: (x0, y0, z0) hoặc None nếu không có nghiệm

```

```

"""
theta1_rad = theta1 * self.dtr
theta2_rad = theta2 * self.dtr
theta3_rad = theta3 * self.dtr

y1 = -(self.t + self.rf * math.cos(theta1_rad))
z1 = -self.rf * math.sin(theta1_rad)

y2 = (self.t + self.rf * math.cos(theta2_rad)) * 0.5 # 0.5 = sin30
x2 = y2 * self.sqrt3 # tan60
z2 = -self.rf * math.sin(theta2_rad)

y3 = (self.t + self.rf * math.cos(theta3_rad)) * 0.5
x3 = -y3 * self.sqrt3
z3 = -self.rf * math.sin(theta3_rad)

dnm = (y2 - y1) * x3 - (y3 - y1) * x2

if abs(dnm) < 1e-9:
 return None # Cấu hình kỳ dị

w1 = y12 + z12
w2 = x22 + y22 + z22
w3 = x32 + y32 + z32

x = (a1*z + b1)/dnm
a1 = (z2 - z1) * (y3 - y1) - (z3 - z1) * (y2 - y1)
b1 = -((w2 - w1) * (y3 - y1) - (w3 - w1) * (y2 - y1)) / 2.0

y = (a2*z + b2)/dnm;
a2 = -(z2 - z1) * x3 + (z3 - z1) * x2
b2 = ((w2 - w1) * x3 - (w3 - w1) * x2) / 2.0

a*z^2 + b*z + c = 0
a = a12 + a22 + dnm2
b = 2.0 * (a1 * b1 + a2 * (b2 - y1 * dnm) - z1 * dnm * dnm)
c = (b2 - y1 * dnm)^2 + b12 + dnm2 * (z12 - self.re2)

discriminant
d = b2 - 4.0 * a * c
if d < 0:
 return None # non-existing point

z0 = -0.5 * (b + math.sqrt(d)) / a
x0 = (a1 * z0 + b1) / dnm
y0 = (a2 * z0 + b2) / dnm

FIX: Đảo dấu X để phù hợp với hệ tọa độ robot
Update 28/11/2025: Xoay hệ tọa độ 180 độ (X -> -X, Y -> -Y)
User report: X+ bị ngược (ở bên trái) -> Đổi lại thành x0
return (x0, -y0, z0)

```

---

```
def forward_kinematics_tool(self, theta1, theta2, theta3, alpha_deg):
 """
 Tính toán tọa độ ĐẦU HÚT (xt, yt, zt) từ các góc động cơ
 VÀ góc servo. (HÀM MỚI)

 :param theta1, theta2, theta3: Góc 3 động cơ (độ)
 :param alpha_deg: Góc của servo (độ) dùng cho tool offset
 :return: (xt, yt, zt) của đầu hút, hoặc None nếu không có nghiệm
 """

 # 1. Tính tọa độ TÂM BỆ DI ĐỘNG (effector)
 effector_coords = self.forward_kinematics(theta1, theta2, theta3)

 if effector_coords is None:
 # Không thể tính được vị trí tâm bệ
 return None

 x0, y0, z0 = effector_coords

 # 2. Từ tâm bệ, tính ra vị trí đầu hút (xt, yt, zt)
 alpha_rad = math.radians(alpha_deg)

 # Tool offset theo alpha (góc quay của servo)
 # Quy ước servo: 0° → X+, 90° → Y+, 180° → X-, 270° → Y-
 # Update 02/12: Fix dấu Y - phải CỘNG cả X và Y
 xt = x0 + self.tool_offset_radius * math.cos(alpha_rad)
 yt = y0 + self.tool_offset_radius * math.sin(alpha_rad)

 # zt = z_effector - offset_z (vì tool thấp hơn)
 zt = z0 - self.tool_offset_z

 return (xt, yt, zt)

def _calc_angle_yz(self, x0, y0, z0):
 """
 Hàm trợ giúp tính góc theta cho 1 cánh tay.
 Tính cho TÂM BỆ (x0, y0, z0).
 """
 # Hằng số f/2 * tan30
 y1 = -0.5 * self.tan30 * self.f

 # Trừ offset effector
 y0_mod = y0 - 0.5 * self.tan30 * self.e

 if abs(z0) < 1e-9:
 return None # Tránh chia cho 0

 # z = a + b*y
 a = (x0 + y0_mod2 + z0 + self.rf2 - self.re2 - y12) / (2.0 * z0)
 b = (y1 - y0_mod) / z0
```

```
discriminant
d = -(a+b*y1)**2 + self.rf*(b**2*self.rf + self.rf)

Nếu d < 0, điểm không tồn tại.
Nếu d = 0, điểm nằm trên biên của không gian làm việc (điểm kỳ dị).
Sử dụng một sai số nhỏ để tránh các vấn đề về số thực và coi các
điểm rất gần kỳ dị là không thể đạt tới.
if d < 1e-9:
 return None # Điểm không tồn tại hoặc là điểm kỳ dị

choosing outer point
yj = (y1 - a * b - math.sqrt(d)) / (b**2 + 1.0)
zj = a + b * yj

Ngăn chia cho 0 nếu yj == y1
if abs(y1 - yj) < 1e-9:
 if zj < 0:
 theta_deg = -90.0
 else:
 theta_deg = 90.0
else:
 theta_rad = math.atan(-zj / (y1 - yj))
 theta_deg = 180.0 * theta_rad / self.pi

Xử lý góc phần tư
if yj > y1:
 theta_deg += 180.0

return theta_deg

def inverse_kinematics(self, x0, y0, z0):
 """
 Tính toán các góc động cơ (theta1, 2, 3) cho một tọa độ TÂM BÊ (x0,
 y0, z0).
 ↳ OPTIMIZED: Sử dụng Numba JIT compilation (20-100x nhanh hơn).

 :return: (theta1, theta2, theta3) trong độ, hoặc None nếu không thể
 đạt
 """
 # Gọi Numba-optimized function
 theta1, theta2, theta3 = inverse_kinematics_numba(
 x0, y0, z0,
 self.tan30, self.f, self.e, self.rf, self.re, self.pi, self.sqrt3,
 self.cos120, self.sin120, C.ARM_ANGLE_MIN, C.ARM_ANGLE_MAX
)

 # Kiểm tra thất bại (Numba trả về nan)
 if np.isnan(theta1):
 return None
```

```
 return (theta1, theta2, theta3)

def inverse_kinematics_tool(self, xt, yt, zt, alpha_deg):
 """
 Tính động học nghịch cho ĐẦU HÚT (xt, yt, zt) có offset servo.
 ↪ OPTIMIZED: Sử dụng Numba JIT compilation (20-100x nhanh hơn).
 """
 # Gọi Numba-optimized function trực tiếp
 theta1, theta2, theta3 = inverse_kinematics_tool_numba(
 xt, yt, zt, alpha_deg,
 self.tool_offset_radius, self.tool_offset_z,
 self.tan30, self.f, self.e, self.rf, self.re, self.pi, self.sqrt3,
 self.cos120, self.sin120, C.ARM_ANGLE_MIN, C.ARM_ANGLE_MAX
)

 # Kiểm tra thất bại (Numba trả về nan)
 if np.isnan(theta1):
 return None

 return (theta1, theta2, theta3)

--- CÁC HÀM TIỆN ÍCH CHUYỂN ĐỔI SANG XUNG ---
def angles_to_steps(self, theta1, theta2, theta3):
 """
 Chuyển đổi góc khớp (độ) thành số xung động cơ.
 """
 steps1 = theta1 * self.steps_per_arm_degree
 steps2 = theta2 * self.steps_per_arm_degree
 steps3 = theta3 * self.steps_per_arm_degree

 return (round(steps1), round(steps2), round(steps3))

def steps_to_angles(self, steps1, steps2, steps3):
 """
 Chuyển đổi số xung động cơ thành góc khớp (độ).
 Đây là hàm ngược lại của angles_to_steps.
 """
 theta1 = steps1 / self.steps_per_arm_degree
 theta2 = steps2 / self.steps_per_arm_degree
 theta3 = steps3 / self.steps_per_arm_degree

 return (theta1, theta2, theta3)

def inverse_kinematics_to_steps(self, x0, y0, z0):
 """
 Tính số xung động cơ trực tiếp từ tọa độ TÂM BÊ (effector).
 (Hàm này chỉ dùng để test, controller nên dùng _tool_)
 """
 angles = self.inverse_kinematics(x0, y0, z0)
 if angles is None:
```

```
 return None
 return self.angles_to_steps(*angles)

def inverse_kinematics_tool_to_steps(self, xt, yt, zt, alpha_deg):
 """
 Tính số xung động cơ từ tọa độ ĐẦU HÚT.
 Đây là hàm mà robot_controller nên sử dụng.
 """
 angles = self.inverse_kinematics_tool(xt, yt, zt, alpha_deg)
 if angles is None:
 return None
 return self.angles_to_steps(*angles)
```

## PHỤ LỤC 2: CHƯƠNG TRÌNH CHÍNH TRÊN STM32

Chương trình file main.c

```
#include "main.h"
#include "usb_device.h"

/* Private includes -----
*/
/* USER CODE BEGIN Includes */
#include "cdc_handler.h"
#include "command_queue.h"
#include "robot_control.h"
#include "conveyor.h"
#include "status_led.h"
#include "command_parser.h"
#include "button_handler.h"
#include "stm32f1xx_hal.h"
#include <stdbool.h>
#include <stdint.h>
/* USER CODE END Includes */

/* Private typedef -----
*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define -----
*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -----
*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables -----
*/
TIM_HandleTypeDef htim2;
TIM_HandleTypeDef htim3;
TIM_HandleTypeDef htim4;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----
```

```
/*
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_TIM2_Init(void);
static void MX_TIM3_Init(void);
static void MX_TIM4_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code -----
*/
/* USER CODE BEGIN 0 */
static char main_command_buffer[RX_BUFFER_SIZE]; // NOLINT
/* USER CODE END 0 */

/
 * @brief The application entry point.
 * @retval int
 */
int main(void)
{
 /* USER CODE BEGIN 1 */

 /* USER CODE END 1 */

 /* MCU Configuration-----
*/
 /* Reset of all peripherals, Initializes the Flash interface and the
Systick. */
 HAL_Init();

 /* USER CODE BEGIN Init */

 /* USER CODE END Init */

 /* Configure the system clock */
 SystemClock_Config();

 /* USER CODE BEGIN SysInit */

 /* USER CODE END SysInit */

 /* Initialize all configured peripherals */
 MX_GPIO_Init();
 MX_TIM2_Init();
 MX_TIM3_Init();
 MX_TIM4_Init();
 MX_USB_DEVICE_Init();
}
```

```
/* USER CODE BEGIN 2 */
// Khởi tạo tất cả các module phần mềm
status_led_init();
cdc_handler_init();
queue_init();
robot_init();
conveyor_init();
button_handler_init(); // Khởi tạo module xử lý nút bấm và debounce

// Đợi một chút để USB CDC ổn định
HAL_Delay(100);

// Gửi thông báo hệ thống đã sẵn sàng
cdc_handler_send_response("SYS_READY");
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
 /* USER CODE END WHILE */

 /* USER CODE BEGIN 3 */
 // Xử lý lệnh từ USB (NON-BLOCKING)
 // Kiểm tra xem cdc_handler có lệnh mới không
 if (cdc_handler_get_command(main_command_buffer, RX_BUFFER_SIZE))
 {
 // Nếu có, thực thi lệnh trong main loop (an toàn!)
 parse_command(main_command_buffer);
 }

 // Xử lý hàng đợi lệnh: kiểm tra và gửi lệnh cho robot nếu rảnh
 queue_process();

 // Xử lý các thông báo DONE từ hàng đợi (do ISR đẩy vào)
 queue_handle_done_messages();

 // Cập nhật trạng thái nút bấm và E-Stop (có debounce)
 button_handler_update();

 // Nếu E-Stop vừa thay đổi (được phát hiện bên trong
 button_handler_update),
 // ta nên gửi status ngay. Tuy nhiên button_handler_update gọi callback
 robot_estop_triggered.

 // Để đơn giản, ta kiểm tra cạnh ở đây.
 static bool last_estop_val = false;
 bool current_estop_val = estop_is_triggered();
 if (current_estop_val != last_estop_val) {
 last_estop_val = current_estop_val;
 send_status_report(); // Gửi status ngay khi E-Stop đổi trạng thái
 }
 }
```

```
// --- Xử lý sự kiện nút bấm ---
if (button_just_pressed(BUTTON_1) || button_just_pressed(BUTTON_2)) {
 send_status_report(); // Gửi trạng thái đầy đủ ngay khi nhấn nút
}

// --- Giám sát cảm biến khay (Tray Sensor) ---
static int last_tray_state = -1;
int current_tray_state = (HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_15) ==
GPIO_PIN_RESET) ? 1 : 0;

if (last_tray_state == -1) {
 last_tray_state = current_tray_state;
}
else if (current_tray_state != last_tray_state) {
 HAL_Delay(20);
 int confirm_state = (HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_15) ==
GPIO_PIN_RESET) ? 1 : 0;
 if (confirm_state == current_tray_state) {
 last_tray_state = current_tray_state;
 send_status_report(); // Gửi trạng thái đầy đủ khi cảm biến khay
thay đổi
 }
}

// Cập nhật trạng thái homing (gọi mỗi vòng lặp)
robot_update_homing_state();

// Kiểm tra Servo Idle để ngắt xung (Auto-Detach)
robot_poll_servo_idle();

// Clear flags để tránh memory leak (không log nhưng vẫn clear)
for (int i = 0; i < 3; i++) {
 robot_get_and_clear_flag_homing_ls_trig(i);
}

// Cập nhật LED trạng thái (gọi mỗi vòng lặp để nhấp nháy được smooth)
status_led_update();

// --- Xử lý Homing Watchdog và các cờ báo hiệu từ ISR ---
// Watchdog cho Homing (chỉ kiểm tra khi đang homing)
if (robot_get_state() == ROBOT_STATE_HOMING && robot_get_homing_state() ==
HOMING_STATE_RAISING)
{
 if ((HAL_GetTick() - robot_get_homing_start_tick()) >
HOMING_WATCHDOG_MS)
 {
 robot_abort(); // Dừng robot
 cdc_handler_send_response("ERROR:Homing timed out");
 }
}
```

```
}

// Các cờ khác
if (robot_get_and_clear_flag_homing_timeout_backing_off())
cdc_handler_send_response("ERROR:HOMING_TIMEOUT_BACKING_OFF");
if (robot_get_and_clear_flag_homing_done())
cdc_handler_send_response("HOME_DONE");

int misconfig_motor;
if (robot_get_and_clear_flag_homing_dir_misconfig(&misconfig_motor)) {
 cdc_handler_send_response("ERROR:HOMING_DIR_MISCONFIG on motor %d",
misconfig_motor);
}

// SAFETY: Kiểm tra limit switch bị kẹt/hỗng
for (int i = 0; i < 3; i++) {
 if (robot_get_and_clear_flag_ls_stuck(i)) {
 cdc_handler_send_response("ERROR:LIMIT_SWITCH_STUCK:M%d (backoff
>2000 steps, sensor may be broken)", i + 1);
 robot_abort(); // Dừng khẩn cấp
 }
}

for (int i = 0; i < 3; i++) {
 if (robot_get_and_clear_flag_up_blocked(i))
cdc_handler_send_response("WARN:Motor %d upward movement blocked by limit
switch", i + 1);
}
/* USER CODE END 3 */
}

/
* @brief System Clock Configuration
* @retval None
*/
void SystemClock_Config(void)
{
RCC_OscInitTypeDef RCC_OscInitStruct = {0};
RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};

/* Initializes the RCC Oscillators according to the specified parameters
* in the RCC_OscInitTypeDef structure.
*/
RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
RCC_OscInitStruct.HSEState = RCC_HSE_ON;
RCC_OscInitStruct.HSEPredivValue = RCC_HSE_PREDIV_DIV1;
RCC_OscInitStruct.HSISState = RCC_HSI_ON;
RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
```

```
RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL9;
if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
{
 Error_Handler();
}

/* Initializes the CPU, AHB and APB buses clocks
*/
RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
 |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
{
 Error_Handler();
}
PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_USB;
PeriphClkInit.UsbClockSelection = RCC_USBCLKSOURCE_PLL_DIV1_5;
if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
{
 Error_Handler();
}
}

/*
 * @brief TIM2 Initialization Function
 * @param None
 * @retval None
 */
static void MX_TIM2_Init(void)
{
 /* USER CODE BEGIN TIM2_Init 0 */

 /* USER CODE END TIM2_Init 0 */

 TIM_ClockConfigTypeDef sClockSourceConfig = {0};
 TIM_MasterConfigTypeDef sMasterConfig = {0};

 /* USER CODE BEGIN TIM2_Init 1 */

 /* USER CODE END TIM2_Init 1 */
 htim2.Instance = TIM2;
 htim2.Init.Prescaler = 71;
 htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
 htim2.Init.Period = 99;
 htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
 htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
}
```

```
if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
{
 Error_Handler();
}
sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
{
 Error_Handler();
}
sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
{
 Error_Handler();
}
/* USER CODE BEGIN TIM2_Init 2 */
// FIX: Cấu hình GPIO cho TIM2_CH1 (PA0) để xuất PWM cho băng tải
HAL_TIM_MspPostInit(&htim2);
// Cấu hình channel PWM sẽ được thực hiện trong conveyor_init
// sMasterConfig và HAL_TIMEx_MasterConfigSynchronization đã được chuyển vào
conveyor_init
/* USER CODE END TIM2_Init 2 */

}

/
* @brief TIM3 Initialization Function
* @param None
* @retval None
*/
static void MX_TIM3_Init(void)
{
 /* USER CODE BEGIN TIM3_Init 0 */

 /* USER CODE END TIM3_Init 0 */

 TIM_ClockConfigTypeDef sClockSourceConfig = {0};
 TIM_MasterConfigTypeDef sMasterConfig = {0};

 /* USER CODE BEGIN TIM3_Init 1 */
 // Việc khởi tạo chi tiết timer này (prescaler, period, interrupt callback)
 // đã được chuyển vào hàm robot_init() trong file robot_control.c
 // để tránh bị CubeMX ghi đè.
 // Hàm này được giữ lại ở đây để duy trì cấu trúc dự án của CubeMX.
 /* USER CODE END TIM3_Init 1 */
 htim3.Instance = TIM3;
 htim3.Init.Prescaler = 71;
 htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
 htim3.Init.Period = 99;
 htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
```

```
htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
if (HAL_TIM_Base_Init(&htim3) != HAL_OK)
{
 Error_Handler();
}
sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) != HAL_OK)
{
 Error_Handler();
}
sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig) != HAL_OK)
{
 Error_Handler();
}
/* USER CODE BEGIN TIM3_Init 2 */

/* USER CODE END TIM3_Init 2 */

}

/
* @brief TIM4 Initialization Function
* @param None
* @retval None
*/
static void MX_TIM4_Init(void)
{
 /* USER CODE BEGIN TIM4_Init 0 */

 /* USER CODE END TIM4_Init 0 */

 TIM_ClockConfigTypeDef sClockSourceConfig = {0};
 TIM_MasterConfigTypeDef sMasterConfig = {0};
 TIM_OC_InitTypeDef sConfigOC = {0};

 /* USER CODE BEGIN TIM4_Init 1 */
 // Việc khởi tạo chi tiết timer này cho Servo (PWM)
 // đã được chuyển vào hàm robot_init() trong file robot_control.c
 // để tránh bị CubeMX ghi đè.
 /* USER CODE END TIM4_Init 1 */
 htim4.Instance = TIM4;
 htim4.Init.Prescaler = 71;
 htim4.Init.CounterMode = TIM_COUNTERMODE_UP;
 htim4.Init.Period = 19999;
 htim4.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
 htim4.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
 if (HAL_TIM_Base_Init(&htim4) != HAL_OK)
 {
```

```
 Error_Handler();
}
sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
if (HAL_TIM_ConfigClockSource(&htim4, &sClockSourceConfig) != HAL_OK)
{
 Error_Handler();
}
if (HAL_TIM_PWM_Init(&htim4) != HAL_OK)
{
 Error_Handler();
}
sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
if (HAL_TIMEX_MasterConfigSynchronization(&htim4, &sMasterConfig) != HAL_OK)
{
 Error_Handler();
}
sConfigOC.OCMode = TIM_OCMODE_PWM1;
sConfigOC.Pulse = 1500;
sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
if (HAL_TIM_PWM_ConfigChannel(&htim4, &sConfigOC, TIM_CHANNEL_4) != HAL_OK)
{
 Error_Handler();
}
/* USER CODE BEGIN TIM4_Init_2 */

/* USER CODE END TIM4_Init_2 */
HAL_TIM_MspPostInit(&htim4);

}

/
 * @brief GPIO Initialization Function
 * @param None
 * @retval None
 */
static void MX_GPIO_Init(void)
{
 GPIO_InitTypeDef GPIO_InitStruct = {0};
/* USER CODE BEGIN MX_GPIO_Init_1 */

/* USER CODE END MX_GPIO_Init_1 */

/* GPIO Ports Clock Enable */
__HAL_RCC_GPIOD_CLK_ENABLE();
__HAL_RCC_GPIOA_CLK_ENABLE();
__HAL_RCC_GPIOB_CLK_ENABLE();

/*Configure GPIO pin Output Level */
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_3|GPIO_PIN_5
```

```
|GPIO_PIN_7|GPIO_PIN_9, GPIO_PIN_RESET); // PA9
(PUMP) = 0V = TẮT

/*Configure GPIO pin Output Level */
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_2|GPIO_PIN_4|GPIO_PIN_6
|GPIO_PIN_10|GPIO_PIN_15, GPIO_PIN_SET);

/*Configure GPIO pin Output Level */
HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_10, GPIO_PIN_RESET);

/*Configure GPIO pin Output Level */
HAL_GPIO_WritePin(GPIOB, GPIO_PIN_11|GPIO_PIN_3, GPIO_PIN_SET);

/*Configure GPIO pin Output Level */
HAL_GPIO_WritePin(CAMERA_LIGHT_GPIO_Port, CAMERA_LIGHT_Pin,
GPIO_PIN_RESET); // Đèn trợ sáng BẬT khi khởi động (Relay thường đóng: LOW =
đóng relay = bật đèn)

/*Configure GPIO pins : PA0 PA1 PA2 PA3
PA4 PA5 PA6 PA7
PA9 PA10 PA15 */
GPIO_InitStruct.Pin = GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3
|GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7
|GPIO_PIN_9|GPIO_PIN_10|GPIO_PIN_15;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

/*Configure GPIO pins : PB0 PB1 PB10 PB11
PB3 PB4 */
GPIO_InitStruct.Pin = GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_10|GPIO_PIN_11
|GPIO_PIN_3|GPIO_PIN_4;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

/*Configure GPIO pins : PB12 PB13 PB14 (Reserved - NC with PULLUP) */
GPIO_InitStruct.Pin = GPIO_PIN_12|GPIO_PIN_13|GPIO_PIN_14;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_PULLUP;
HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

/*Configure GPIO pins : PB15 (Conveyor Sensor - NPN NO with PULLUP) */
GPIO_InitStruct.Pin = GPIO_PIN_15;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_PULLUP;
HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

/*Configure GPIO pins : PB6, PB7 (START and STOP buttons - NO with VCC) */
```

---

```
/* Logic: Pressed = 1 (VCC), Released = 0 (Internal PULLDOWN) */
GPIO_InitStruct.Pin = GPIO_PIN_6|GPIO_PIN_7;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_PULLDOWN;
HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

/*Configure GPIO pin : PA8 (E-Stop - NC with PULLUP, interrupt on RISING
edge) */
GPIO_InitStruct.Pin = GPIO_PIN_8;
GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
GPIO_InitStruct.Pull = GPIO_PULLUP;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

/* EXTI interrupt init*/
HAL_NVIC_SetPriority(EXTI9_5_IRQn, 2, 0);
HAL_NVIC_EnableIRQ(EXTI9_5_IRQn);

/* USER CODE BEGIN MX_GPIO_Init_2 */
// Các cấu hình GPIO khác như công tắc hành trình, nút nhấn...
// nên được thực hiện trong các module tương ứng (ví dụ: robot_init)
// để đảm bảo tính đóng gói và dễ quản lý.

/* USER CODE END MX_GPIO_Init_2 */
}

/* USER CODE BEGIN 4 */

/
* @brief GPIO EXTI Callback - Xử lý ngắt E-Stop
* @param GPIO_Pin: Pin gây ra ngắt
* @note: Đã chuyển sang xử lý debounce trong button_handler_update()
* Ngắt này chỉ để phản ứng nhanh, logic thực sự xử lý ở polling
*/
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
 // Ngắt E-Stop - không xử lý gì ở đây
 // Debounce và xử lý logic sẽ được thực hiện trong button_handler_update()
 // Điều này tránh việc gọi hàm phức tạp từ ISR
 (void)GPIO_Pin; // Tránh warning unused parameter
}

/* USER CODE END 4 */

/
* @brief This function is executed in case of error occurrence.
* @retval None
*/
void Error_Handler(void)
{
/* USER CODE BEGIN Error_Handler_Debug */
/* User can add his own implementation to report the HAL error return state
```

```
/*
__disable_irq();
while (1)
{
}
/* USER CODE END Error_Handler_Debug */
}
```

## PHỤ LỤC 3: CHƯƠNG TRÌNH CHÍNH TRÊN MÁY TÍNH

### Chương trình file main.py

```
import sys
import os
from PyQt6.QtWidgets import QApplication
from PyQt6.QtCore import QTimer
from pyqt_delta_gui import DeltaRobotGUI
from connection_manager import ConnectionManager
from robot_controller import RobotController
import constants as C

... [Các phần import khác và cấu hình môi trường] ...

def main():
 # Cấu hình môi trường Qt để hiển thị tốt trên màn hình High-DPI
 os.environ["QT_LOGGING_RULES"] = "*.debug=false;qt.qpa.*=false"

 global qt_app
 qt_app = QApplication(sys.argv)

 try:
 # 1. Khởi tạo Giao diện chính (GUI)
 app = DeltaRobotGUI()

 # 2. Khởi tạo Bộ điều khiển Robot & Quản lý kết nối
 app.robot_controller = RobotController(app, app.conn_manager)

 # 3. Liên kết các sự kiện nút bấm trên GUI với logic điều khiển
 app.set_button_commands({
 'run': app.on_run if hasattr(app, 'on_run') else lambda:
print("Run"),
 'stop': app.on_stop if hasattr(app, 'on_stop') else lambda:
print("Stop"),
 'connect_stm': lambda: on_connect_stm(app),
 # ... [Các lệnh bind nút bấm khác: Move, Home, Pump, Conveyor...]
 })

 # 4. Liên kết nút bấm cứng (Hardware Buttons) với Auto Mode
 app.robot_controller.hardware_start_pressed.connect(lambda:
app.robot_controller.set_auto_mode(True))
 app.robot_controller.hardware_stop_pressed.connect(lambda:
app.robot_controller.set_auto_mode(False))

 # 5. Đăng ký lắng nghe phản hồi từ STM32
 app.conn_manager.add_message_listener(app.robot_controller.handle_stm_
message)

 # 6. Hiển thị ứng dụng và bắt đầu xử lý Video
 app.show()

```

```
Khởi động luồng Camera sau 100ms (để GUI hiện lên trước)
QTimer.singleShot(100, lambda: start_video_feed(app))

Chạy vòng lặp sự kiện chính của Qt
sys.exit(qt_app.exec())

except KeyboardInterrupt:
 print("\nInterrupted.")
except Exception as e:
 print(f"Error: {e}")

if __name__ == "__main__":
 main()
```

## **PHỤ LỤC 4: ĐƯỜNG DẪN MÃ NGUỒN TRÊN GITHUB**

Đường link githb:[https://github.com/TrieuCris/Cao\\_Van\\_Nhat\\_Trieb](https://github.com/TrieuCris/Cao_Van_Nhat_Trieb)

Mã QR

