ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA ĐIỆN

ĐỒ ÁN TỐT NGHIỆP
CAPSTONE PROJECT

NGÀNH: KỸ THUẬT ĐIỀU KHIỂN VÀ TỰ ĐỘNG HÓA

ĐỀ TÀI:

PHÁT TRIỂN HỆ THỐNG TƯƠNG TÁC NGƯỜI-MÁY CHO ROBOT ABB CRB15000

Người hướng dẫn: **TS. GIÁP QUANG HUY**
KS. LÊ PHƯỚC SINH

Sinh viên thực hiện:

1. ĐỖ ĐĂNG DUY TÚ – MSSV: 105200391 – LỚP: 20TDHCLC1

2. QUÁCH THIỆN ĐỨC – MSSV: 105200358 – LỚP: 20TDHCLC1

3. NGUYỄN HỮU NAM THÀNH – MSSV: 105200386 – LỚP: 20TDHCLC1

Đà Nẵng, 6/2025

TÓM TẮT

Tên đề tài: Phát triển hệ thống tương tác người-máy cho robot ABB CRB15000

Sinh viên thực hiện: Đỗ Đăng Duy Tú Số thẻ SV: 105200391

Quách Thiện Đức Số thẻ SV: 105200358

Nguyễn Hữu Nam Thành Số thẻ SV: 105200386

Lớp: 20TDHCLC1

Trong thời kỳ công nghiệp hóa – hiện đại hóa, robot cộng tác ngày càng đóng vai trò quan trọng trong dây chuyền sản xuất thông minh nhờ khả năng làm việc linh hoạt và an toàn bên cạnh con người. Tuy nhiên, để robot thực sự phát huy được hiệu quả trong môi trường thực tế, việc thiết kế một hệ thống tương tác người – máy thân thiện, trực quan và hiệu quả là điều cần thiết. Xuất phát từ nhu cầu đó, nhóm chúng tôi thực hiện đề tài "Phát triển hệ thống tương tác người – máy cho robot ABB CRB15000", nhằm xây dựng một hệ thống phần mềm điều khiển giúp người dùng dễ dàng điều khiển và giám sát robot qua máy tính cá nhân mà không cần chuyên sâu về kỹ thuật lập trình.

Nội dung đề tài tập trung vào nghiên cứu và triển khai các công nghệ chính như giao thức Externally Guided Motion (EGM) để điều khiển vị trí robot theo thời gian thực thông qua máy tính, sử dụng Robot Web Services (RWS) để quản lý và giám sát hoạt động của robot qua REST API, cũng như lập trình giao diện người dùng bằng Python và thư viện PyQt5. Đồng thời, nhóm cũng phối hợp với nhóm phát triển hệ thống nhận diện cử chỉ tay sử dụng MediaPipe và OpenCV nhằm tích hợp tính năng điều khiển robot bằng thao tác tay trong thời gian thực. Hệ thống được thử nghiệm thông qua mô phỏng trên phần mềm RobotStudio và triển khai thực nghiệm trên robot thật tại Công ty TNHH Tự động hóa Nhật Tri. Qua đó, nhóm đánh giá được hiệu quả vận hành, khả năng phản hồi, tính ổn định và độ chính xác của hệ thống khi tương tác với người dùng.

Đề tài không chỉ giúp nhóm chúng tôi nâng cao kiến thức chuyên ngành về điều khiển – tự động hóa và robot công nghiệp mà còn mang lại giải pháp khả thi có thể ứng dụng vào thực tiễn, đặc biệt trong các nhà máy sản xuất có yêu cầu cao về tự động hóa linh hoạt, tiết kiệm chi phí vận hành và tăng hiệu quả lao động.

ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA ĐIỆN

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập - Tự do - Hạnh phúc

NHIỆM VỤ ĐỒ ÁN TỐT NGHIỆP

TT	Họ tên sinh viên	Số thẻ SV	Lớp	Ngành
1	Đỗ Đăng Duy Tú	105200391	20TDHCLC1	Kỹ thuật điều khiển & Tự động hóa
2	Quách Thiện Đức	105200358	20TDHCLC1	Kỹ thuật điều khiển & Tự động hóa
3	Nguyễn Hữu Nam Thành	105200386	20TDHCLC1	Kỹ thuật điều khiển & Tự động hóa

1. *Tên đề tài đồ án:*

   Phát triển hệ thống tương tác người - máy cho robot ABB CRB15000

2. *Đề tài thuộc diện:* ☐ *Có ký kết thỏa thuận sở hữu trí tuệ đối với kết quả thực hiện*

3. *Các số liệu và dữ liệu ban đầu*

4. *Nội dung các phần thuyết minh và tính toán:*

a. Phần chung:

TT	Họ tên sinh viên	Nội dung
1	Đỗ Đăng Duy Tú	- Tìm hiểu và nghiên cứu các thông tin tài liệu, các bài báo khoa học và tạp chí liên quan về các kỹ thuật, phương pháp điều khiển robot. - Viết báo cáo thuyết minh.
2	Quách Thiện Đức	
3	Nguyễn Hữu Nam Thành	

b. Phần riêng:

TT	Họ tên sinh viên	Nội dung
1	Đỗ Đăng Duy Tú	- Nghiên cứu, lập trình giao tiếp giữa PC và Robot dựa trên nền tảng Robot Web Services và Externally Guided Motion sử dụng ngôn ngữ Python - Thiết kế giao diện tương tác người-máy sử dụng ngôn ngữ Python.

2	Quách Thiện Đức	- Nghiên cứu, lập trình xử lý ảnh, nhận diện cử chỉ tay dựa trên thư viện Mediapipe sử dụng ngôn ngữ Python.
3	Nguyễn Hữu Nam Thành	- Nghiên cứu, lập trình chương trình chính giao tiếp giữa Robot với PC dựa trên giao thức Externally Guided Motion bằng ngôn ngữ lập trình RAPID.

5. *Họ và tên người hướng dẫn:* **TS. Giáp Quang Huy**

KS. Lê Phước Sinh

6. *Ngày giao nhiệm vụ đồ án:* *15/03/2025.*

7. *Ngày hoàn thành đồ án:* *02/06/2025*

Trưởng Bộ môn Tự Động Hóa	**Người hướng dẫn**	**Đồng hướng dẫn**
TS. Giáp Quang Huy	**TS. Giáp Quang Huy**	**KS. Lê Phước Sinh**

ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA ĐIỆN

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập - Tự do - Hạnh phúc

PHIẾU KIỂM SOÁT TIẾN ĐỘ LÀM ĐỒ ÁN TỐT NGHIỆP

(Phiếu dành cho người hướng dẫn/sinh viên)

Họ tên sinh viên	Số thẻ SV	Lớp	Ngành
Đỗ Đăng Duy Tú	105200391	20TDHCLC1	Kỹ thuật điều khiển & Tự động hóa
Quách Thiện Đức	105200358	20TDHCLC1	Kỹ thuật điều khiển & Tự động hóa
Nguyễn Hữu Nam Thành	105200386	20TDHCLC1	Kỹ thuật điều khiển & Tự động hóa

Tên đề tài ĐATN: Phát triển hệ thống tương tác người - máy cho robot ABB CRB15000
Họ tên người HD: TS. Giáp Quang Huy Đơn vị: Đại học Bách Khoa ĐN

Tuần	**Ngày**	**Khối lượng**		**GVHD ký tên**
		Đã thực hiện (%)	**Tiếp tục thực hiện (%)**	
1	15/3-23/3	- Nhận đề tài	- Tìm hiểu và nghiên cứu lý thuyết liên quan về đề tài.	
2	24/3-30/3	- Nghiên cứu, tìm hiểu về các ngôn ngữ lập trình cần thiết cho đồ án. - Nghiên cứu, tìm hiểu cách sử dụng các phần mềm chính.	- Tiếp tục nghiên cứu, thực hiện dự án.	
3	31/3-6/4	- Tìm hiểu về các giao thức giao tiếp, điều khiển robot thông qua PC. - Tìm hiểu về xử lý ảnh, nhận dạng cử chỉ tay.	- Viết thuyết minh. - Nghiên cứu lập trình trên RobotStudio.	
4	6/4-7/4	Duyệt lần 1: Đánh giá khối lượng hoàn thành 25%: Được tiếp tục làm ĐATN □ Không tiếp tục thực hiện ĐATN □		

5	7/4-13/4	- Hoàn thành thuyết minh chương 1. - Lập trình điều khiển Robot cơ bản bằng ngôn ngữ Rapid trên phần mềm RobotStudio.	- Nghiên cứu lập trình giao tiếp, điều khiển Robot bằng PC. Thông qua giao thức EGM.	
6	14/4-20/4	- Lập trình điều khiển vị trí, góc quay robot thông qua giao thức EGM bằng ngôn ngữ Python.	- Tiếp tục nghiên cứu thêm về EGM. - Tiếp tục bổ sung thuyết minh.	
7	20/4-27/4	- Hoàn thành việc lập trình EGM giao tiếp điều khiển vị trí của robot với giao diện đơn giản gồm các nút nhấn.	- Nghiên cứu lập trình xử lý ảnh, nhận dạng cử chỉ tay để điều khiển vị trí robot.	
8	27/4-28/4	Duyệt lần 2: Đánh giá khối lượng hoàn thành 50%: Được tiếp tục làm ĐATN □ Không tiếp tục thực hiện ĐATN □		
9	28/4-4/5	- Lập trình hoàn thành nhận dạng cử chỉ tay, điều khiển vị trí robot bằng các nút nhấn áo, đóng mở gripper.	- Nghiên cứu, lập trình RWS. - Thiết kế giao diện tương tác người-robot.	
10	5/5-11/5	- Phát triển một số tính năng tương tác với robot thông qua RWS như: chọn chế độ hoạt động, bật tắt động cơ, điều chỉnh tốc độ. - Phát triển giao diện đơn giản cho RWS.	- Tiếp tục phát triển giao diện. - Tiếp tục hoàn thành thuyết minh chương 1 và 2.	
11	12/5-18/5	- Hoàn thành báo cáo chương 1 và 2. - Phát triển giao diện gồm 3 phần EGM, Camera và RWS.	- Tiếp tục phát triển thêm một số chức năng như kết nối/ngắt kết nối cho EGM, Camera và RWS. - Hoàn thiện giao diện. - Thử nghiệm qua mô phỏng trên phần mềm RobotStudio.	
12	18/5-19/5	Duyệt lần 3: Đánh giá khối lượng hoàn thành 90%: Được tiếp tục làm ĐATN □ Không tiếp tục thực hiện ĐATN □		

13	20/5-1/6	- Hoàn thiện giao diện. - Tối ưu hóa code. - Mô phỏng thành công, kiểm tra các chức năng EGM, RWS.	- Tiến hành thực nghiệm trên Robot thật tại công ty.	
14	2/6-8/6	- Hoàn thành việc thử nghiệm trên robot thật. - Hoàn thành quay video demo.	- Hoàn thiện thuyết minh.	
15	9/6-15/6	- Hoàn thành đồ án.		

LỜI NÓI ĐẦU VÀ CẢM ƠN

Với sự đồng ý và tạo điều kiện từ Khoa Điện, nhóm chúng tôi đã thực hiện nghiên cứu đề tài: "Phát triển hệ thống tương tác người - máy cho robot ABB CRB15000"

Trong quá trình thực hiện đồ án, chúng tôi vinh dự nhận được sự hướng dẫn tận tình, sát sao từ TS. Giáp Quang Huy, giảng viên hướng dẫn chuyên môn, cùng với sự hỗ trợ quý báu của anh Lê Phước Sinh, kỹ sư thuộc Công ty TNHH Tự Động Hóa Nhật Tri.

Thông qua quá trình làm việc, trao đổi với giảng viên và kỹ sư hướng dẫn, chúng tôi không chỉ tiếp thu được nhiều kiến thức chuyên môn mới mẻ, thực tiễn mà còn học hỏi được phong cách làm việc chuyên nghiệp, tác phong nghiêm túc và tinh thần trách nhiệm – những yếu tố quan trọng đối với một kỹ sư ngành điều khiển và tự động hóa.

Chúng tôi xin gửi lời cảm ơn chân thành và sâu sắc đến giảng vi hướng dẫn, các thầy cô trong Khoa Điện - Trường Đại học Bách Khoa, cũng như anh hướng dẫn và toàn thể quý Công ty TNHH Tự Động Hóa Nhật Tri đã đồng hành, hỗ trợ và tạo điều kiện cho chúng tôi trong suốt quá trình thực hiện đề tài.

Chúng tôi nhận thức rõ rằng, để hoàn thành một đồ án tốt nghiệp kỹ sư không chỉ cần kiến thức được học trên giảng đường mà còn đòi hỏi tinh thần cầu tiến, thái độ nghiêm túc, cùng với kinh nghiệm thực tế và sự hướng dẫn tận tâm từ những người đi trước. Chúng tôi mong tiếp tục nhận được sự chỉ dẫn, góp ý quý báu từ thầy cô và các anh/chị kỹ sư để hoàn thiện bản thân hơn nữa trong tương lai.

Chặng đường 5 năm học tập tại trường Đại học đã khép lại. Sau khi hoàn thành đồ án tốt nghiệp này, chúng tôi – những kỹ sư trẻ – mang theo hành trang là kiến thức, kỹ năng và cả thái độ chuyên nghiệp, sẵn sàng bước vào môi trường làm việc thực tế. Những gì chúng tôi đạt được hôm nay là thành quả của sự giảng dạy, dìu dắt tận tâm của các thầy cô Khoa Điện – Trường Đại học Bách Khoa và sự hỗ trợ thiết thực từ Công ty TNHH Tự Động Hóa Nhật Tri. Chúng tôi xin trân trọng cảm ơn.

Chúng tôi xin chân thành cảm ơn!

LỜI CAM ĐOAN LIÊM CHÍNH HỌC THUẬT

Chúng tôi xin cam đoan rằng đồ án tốt nghiệp với đề tài "Phát triển hệ thống tương tác người - máy cho robot ABB CRB15000" là nghiên cứu độc lập của chúng tôi với sự hỗ trợ từ giảng viên hướng dẫn TS. Giáp Quang Huy và KS. Lê Phước Sinh thuộc Công ty TNHH Tự Động Hóa Nhật Tri.

Chúng tôi xin cam đoan toàn bộ số liệu được cung cấp từ báo cáo đều là của công ty và đây là kết quả nghiên cứu hoàn toàn trung thực, không sao chép từ bất kì một công trình nghiên cứu khác nào. Những tài liệu trích dẫn đều đã được ghi rõ nguồn gốc.

Chúng tôi xin chịu hoàn toàn trách nhiệm nếu có bất kỳ sự sao chép, gian dối kết quả nào trong sản phẩm đồ án này.

Đà Nẵng, ngày … tháng 06 năm 2025

Nhóm đồng tác giả thực hiện

Sinh viên 1	**Sinh viên 2**	**Sinh viên 3**
Đỗ Đăng Duy Tú	**Quách Thiện Đức**	**Nguyễn Hữu Nam Thành**

MỤC LỤC

DANH SÁCH CÁC BẢNG, HÌNH VẼ

DANH SÁCH CÁC KÝ HIỆU, CHỮ VIẾT TẮT

CHỮ VIẾT TẮT:

EGM: Externally Guided Motion

RWS: Robot Web Services

HTTP: HyperText Transfer Protocol

API: Application Programming Interface

URL: Uniform Resource Locator

REST: Representational State Transfer

MỞ ĐẦU

1. Lý do chọn đề tài

Trong xu thế chuyển đổi số và tự động hóa, giao diện tương tác người – máy (Human-Machine Interface – HMI) ngày càng đóng vai trò quan trọng trong việc điều khiển, giám sát và phối hợp giữa con người và hệ thống tự động. Đặc biệt trong lĩnh vực robot công nghiệp, việc phát triển một giao diện trực quan, dễ sử dụng sẽ giúp nâng cao hiệu suất làm việc, giảm thiểu sai sót và mở rộng khả năng ứng dụng của robot trong thực tế.

ABB CRB15000 (GoFa) là một Robot cộng tác hiện đại, được thiết kế để hoạt động an toàn và linh hoạt bên cạnh con người. Tuy nhiên, để phát huy tối đa tiềm năng của thiết bị này, người dùng cần có một giao diện tương tác hiệu quả nhằm truyền lệnh, giám sát trạng thái và kiểm soát chuyển động một cách thuận tiện. Xuất phát từ nhu cầu đó, nhóm chúng tôi chọn thực hiện đề tài *"Thiết kế giao diện tương tác người – máy cho robot ABB CRB15000"*, với mục tiêu xây dựng một phương pháp để có thể giao tiếp đơn giản, trực quan và đáp ứng tốt yêu cầu sử dụng trong môi trường thực tế.

2. Mục tiêu đề tài

+ Thiết kế và xây dựng giao diện người dùng giúp giao tiếp với robot ABB CRB15000 thông qua máy tính.

+ Hỗ trợ các chức năng cơ bản như: điều khiển chuyển động robot, giám sát trạng thái bộ điều khiển, giám sát và điều khiển tốc độ của robot, gửi lệnh thực thi chương trình RAPID, hiển thị dữ liệu tốc độ, góc quay phản hồi từ robot.

+ Đảm bảo giao diện thân thiện, trực quan và dễ sử dụng đối với người vận hành không chuyên về kỹ thuật lập trình.

+ Sử dụng các công nghệ như Python, giao thức hướng dẫn chuyển động bên ngoài (EGM), giao thức HTTP REST API (Robot Web Service – RWS), kết hợp mô phỏng với RobotStudio.

+ Nhận diện cử chỉ tay để điều khiển một số hoạt động của Robot như vị trí, góc quay, đóng mở công cụ.

3. Phạm vi và đối tượng nghiên cứu

3.1. Đối tượng nghiên cứu:

Robot ABB CRB15000, Externally Guided Motion (EGM), Robot Web Service (RWS), ngôn ngữ lập trình Python, thư viện giao diện người dùng và các công cụ hỗ trợ lập trình giao tiếp robot.

3.2. Phạm vi đề tài:

+ Tập trung vào thiết kế phần mềm giao diện giao tiếp giữa người dùng và robot thông qua máy tính.

+ Thử nghiệm trên RobotStudio hoặc bộ điều khiển thật của robot ABB.

4. Phương pháp nghiên cứu

+ Tìm hiểu đặc điểm kỹ thuật robot ABB CRB15000, giao thức EGM và giao thức RWS.

+ Phân tích yêu cầu của người dùng để xây dựng các chức năng cần thiết cho giao diện.

+ Thiết kế, lập trình giao diện bằng ngôn ngữ Python kết hợp thư viện phù hợp (PyQt).

+ Thử nghiệm giao diện với mô phỏng robot trong RobotStudio hoặc hệ thống thực tế.

+ Đánh giá hiệu quả và cải tiến giao diện dựa trên phản hồi người dùng.

5. Cấu trúc của đồ án

- TÓM TẮT
- NHIỆM VỤ ĐỒ ÁN TỐT NGHIỆP
- PHIẾU KIỂM SOÁT TIẾN ĐỘ LÀM ĐỒ ÁN TỐT NGHIỆP
- LỜI NÓI ĐẦU VÀ CẢM ƠN
- LỜI CAM ĐOAN LIÊM CHÍNH HỌC THUẬT
- MỤC LỤC
- DANH SÁCH CÁC BẢNG, HÌNH VẼ
- DANH SÁCH CÁC KÝ HIỆU, CHỮ VIẾT TẮT
- MỞ ĐẦU

- Chương 1: NGHIÊN CỨU TỔNG QUAN
- Chương 2: XÂY DỰNG HỆ THỐNG NHẬN DIỆN CỬ CHỈ TAY ĐỂ ĐIỀU KHIỂN ROBOT
- Chương 3: XÂY DỰNG HỆ THỐNG GIAO TIẾP GIỮA CAMERA-PC -ROBOT
- Chương 4: KIỂM NGHIỆM VÀ ĐÁNH GIÁ KẾT QUẢ
- KẾT LUẬN
- TÀI LIỆU THAM KHẢO
- PHỤ LỤC

Chương 1: NGHIÊN CỨU TỔNG QUAN

1.1. Tổng quan về robot ABB GOFA CRB150000

1.1.1. Robot ABB GOFA CRB15000

ABB GOFA CRB15000 là một robot cộng tác (collaborative robot - cobot) tiên tiến, mạnh mẽ và an toàn được thiết kế để xử lý tải trọng lên đến 5kg của ABB. GOFA kết hợp một loạt các tính năng cho phép sử dụng an toàn, trực tiếp cùng với người dùng mà không cần thiết kế và xây dựng các rào chắn hoặc hàng rào cồng kềnh. GOFA có thể liên tục chia sẻ không gian làm việc với mọi người, mang lại sự linh hoạt và hiệu quả tối đa. Robot và con người hợp tác thực hiện cùng một nhiệm vụ, mà không gây nguy hiểm và vẫn cho hiệu năng tốt. GOFA có thể thực hiện được nhiều ứng dụng khác nhau từ dòng sản xuất tự động cho đến đóng gói sản phẩm và hỗ trợ lắp ráp.

Hình 1 1 Robot ABB Gofa CRB15000

Một hệ Robot bao gồm các thành phần:

1. Cánh tay Robot
2. Bộ điều khiển Robot
3. Thiết bị lập trình cầm tay (FlexPendant)
4. Cáp và phần mềm lập trình (Robotstudio)

Thông số kỹ thuật:

+ Số trục: 6.
+ Cấp bảo vệ: IP54.
+ Lắp đặt: Có thể lắp đặt ở nhiều vị trí: tường, trần nhà, bàn làm việc…
+ Bộ điều khiển: OmniCore C30.
+ Tín hiệu kết nối: 4 tín hiệu gồm IO, Fieldbus hoặc Ethernet.
+ Trọng lượng: 27 kg.
+ Tầm với của robot: 950 mm.
+ Tải trọng tối đa: 5 kg.
+ Tốc độ tối đa: 2.2 m/s.
+ Chiều dài cáp robot: 7 m.

1.1.2. Bộ điều khiển Omnicore C30 và FlexPendant

Hình 1 2 Bộ điều khiển Omnicore C30 và FlexPendant

1.1.2.1. Bộ điều khiển Omnicore C30

OmniCore C30 là bộ điều khiển chính của robot GoFa CRB 15000, được thiết kế với kích thước nhỏ gọn nhưng tích hợp đầy đủ khả năng điều khiển chuyển động chính xác và linh hoạt, phù hợp với các yêu cầu tự động hóa hiện đại.

Các đặc điểm chính của bộ điều khiển:

+ Hỗ trợ nhiều giao thức truyền thông công nghiệp như Ethernet/IP, ProfiNet, DeviceNet...

+ Thiết kế dạng desktop với quản lý dây gọn gàng, dễ tích hợp vào không gian hẹp.

+ Tích hợp các tính năng an ninh mạng để ngăn chặn truy cập trái phép hoặc phần mềm không xác thực.

Thông số kỹ thuật:

+ Trọng lượng: 11kg.

+ Kích thước: 509 mm × 514 mm × 193 mm.

+ Cấp bảo vệ: IP20.

+ Giao tiếp: Ethernet: Mạng LAN riêng, IO LAN, và WAN.

+ Cổng IO tùy chọn: Ethernet IP, ProfiNet, ProfiSafe, CC-Link, DeviceNet.

+ Yêu cầu nguồn vào: Một pha 100 –230 VAC, 50 – 60 Hz.

+ Ngõ vào/ra dạng digital: 24 VDC, 16 ngõ vào/16 ngõ ra

+ Thiết bị dạy lập trình: ABB FlexPendant với cáp 3 m, USB 3.0, màn hình cảm ứng màu 8” với joystick và bàn phím màng 12 nút.

1.1.2.2. Thiết bị lập trình cầm tay (FlexPendant)

FlexPendant là thiết bị đầu cuối dùng để điều khiển và lập trình robot ABB. Nó cung cấp giao diện người dùng cho phép tương tác và lập trình cho robot ABB một cách trực quan, hiệu quả. Với màn hình cảm ứng đa điểm và phần mềm lập trình đồ họa ABB Wizard, FlexPendant cho phép người sử dụng dễ dàng lập trình và giám sát robot mà không cần kỹ năng lập trình chuyên sâu

Các đặc điểm chính của thiết bị lập trình cầm tay:

+ Hiện đại: FlexPendant được thiết kế với giao diện dễ sử dụng hiện đại cung cấp trải nghiệm điều khiển robot mượt mà và hiệu quả.

+ Màn hình cảm ứng đa điểm lớn với các cử chỉ chuẩn (nhấn, vuốt, chạm): Màn hình của FlexPendant cho phép người dùng tương tác với robot bằng các thao tác cảm ứng quen thuộc, giống như sử dụng điện thoại thông minh, giúp việc điều khiển trở nên dễ dàng và trực quan hơn.

+ Có thể thay thế: Tính năng này cho phép người dùng tháo rời FlexPendant trong khi hệ thống đang hoạt động và chia sẻ giữa các robot mà không làm gián đoạn quá trình làm việc.

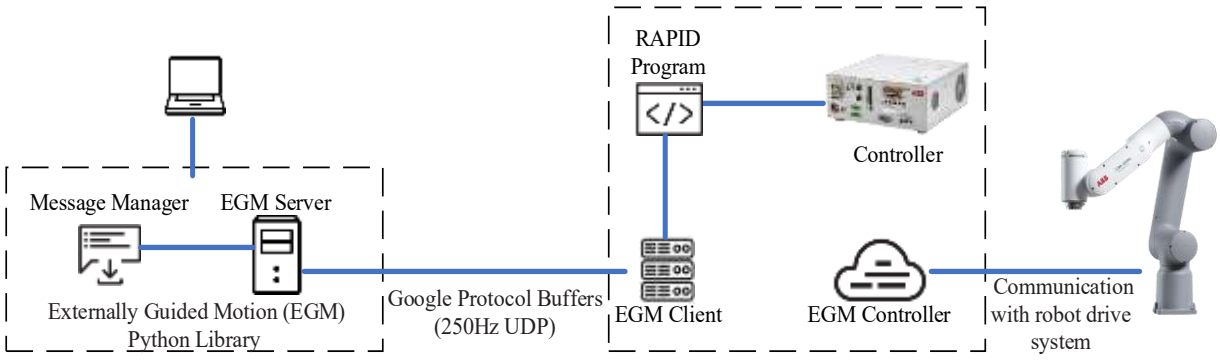
1.2. Giới thiệu chung về chuyển động hướng dẫn bên ngoài (EGM)

1.2.1. Tổng quan về EGM

1.2.1.1. Chuyển động hướng dẫn bên ngoài (EGM)

EGM là một module được ABB phát hành cùng với RobotWare 6, cho phép điều khiển chuyển động của robot theo thời gian thực bằng dữ liệu vị trí/vận tốc từ một thiết bị bên ngoài như cảm biến hoặc máy tính. Module này cung cấp phản hồi tốt hơn đối với tín hiệu từ cảm biến bên ngoài. EGM cho phép điều chỉnh đường đi của robot trong khoảng 10 - 20 ms. EGM đi trực tiếp vào bộ phát sinh tín hiệu động cơ, và bằng cách bỏ qua bộ lập kế hoạch đường đi, robot có thể phản ứng nhanh hơn với dữ liệu cảm biến mới. Tất cả việc lọc, giám sát tham chiếu và xử lý trạng thái đều do EGM đảm nhiệm.

Tham chiếu có thể được đưa ra dưới dạng giá trị khớp (joint values) hoặc theo tư thế (pose). Sử dụng tư thế yêu cầu phải có các khung tham chiếu. Các phép đo từ cảm biến và hướng thay đổi vị trí chỉ có thể được đưa ra tương đối với các khung tham chiếu. Khi sử dụng giá trị khớp, không cần khung tham chiếu vì cả giá trị cảm biến và giá trị vị trí đều là góc trục được cho theo độ so với vị trí hiệu chuẩn của mỗi trục. Cài đặt sử dụng giá trị khớp khá tương đồng với cài đặt theo tư thế, ngoại trừ việc không cần khung tham chiếu.



Hình 1 3 Sơ đồ tổng quan của EGM

Externally Guided Motion (EGM) bao gồm 3 tính năng khác nhau:

- **EGM Position Stream:**

Vị trí hiện tại và kế hoạch của các đối tượng cơ khí trong RAPID Task được gửi đến một thiết bị bên ngoài.

- **EGM Position Guidance:**

Robot không thực hiện theo đường đi được lập trình trong RAPID mà thực hiện theo đường đi do một thiết bị bên ngoài tạo ra

- **EGM Path Correction:**

Đường đi của robot được lập trình được sửa đổi/ hiệu chỉnh bởi dữ liệu của các thiết bị bên ngoài

1.2.2. Hướng dẫn vị trí EGM (EGM Position Guidance)

1.2.2.1. Giới thiệu

EGM Position Guidance được thiết kế cho người dùng nâng cao và cung cấp giao diện cấp thấp cho bộ điều khiển robot, bằng cách bỏ qua việc lập kế hoạch đường đi, nó có thể được sử dụng khi cần chuyển động robot có độ phản hồi cao và chuyển động mượt mà.

EGM Position Guidance có thể được sử dụng để đọc vị trí và ghi vị trí đến hệ thống chuyển động của robot với tốc độ cao. Điều này có thể được thực hiện 4ms một lần với độ trễ điều khiển vào khoảng 10-20 ms tùy thuộc vào loại robot. Các tham chiếu có thể được xác định bằng cách sử dụng các giá trị của trục hoặc tư thế. Tư thế có thể được định nghĩa trong bất kỳ work object nào không di chuyển trong quá trình di chuyển EGM Position Guidance

Ưu điểm chính của EGM Position Guidance là tốc độ cao và độ trễ thấp so với các phương tiện điều khiển chuyển động bên ngoài khác. Khoảng thời gian từ khi viết một vị trí mới cho đến khi vị trí đã cho đó bắt đầu ảnh hưởng đến vị trí robot thực tế thường là khoảng 20 ms. Ngoài ra, EGM Position Guidance cung cấp độ chính xác tuyệt đối, nó đảm bảo robot đạt được vị trí mục tiêu như được chỉ định.

1.2.2.2. Thiết lập EGM Position Guidance

Các bước thực hiện thiết lập EGM Position Guidance như sau: [1]

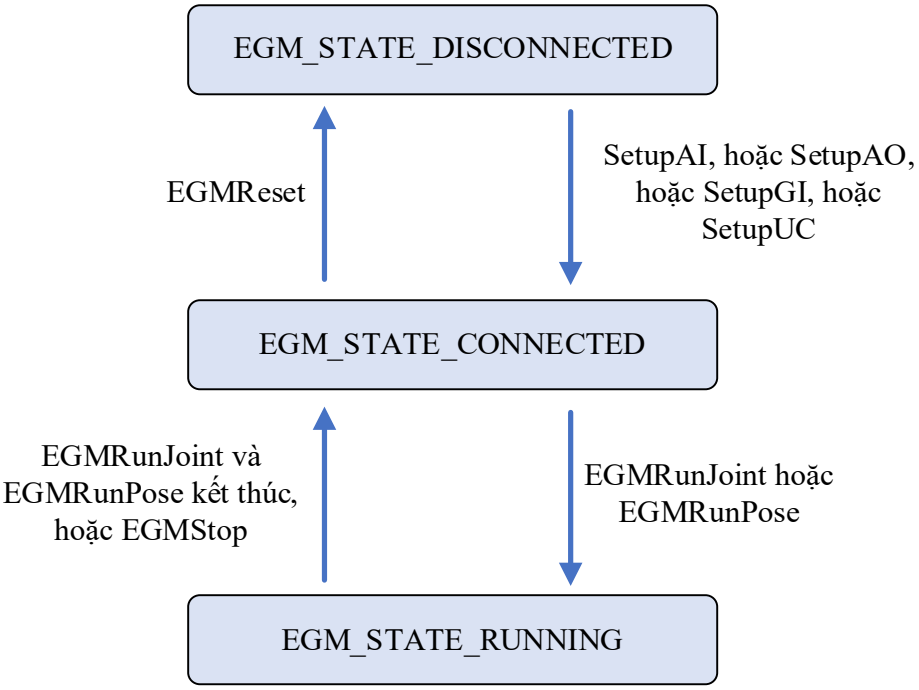
1. Thiết lập RobotWare System, cho phép tùy chọn RobotWare Externally Guided Motion.
2. Thiết lập thiết bị bên ngoài với giao thức cảm biến EGM để thiết bị có thể giao tiếp với bộ điều khiển robot.
3. Thiết lập RAPID code xác định các chi tiết của chức năng Externally Guided Motion.

4. Thiết bị cung cấp dữ liệu đầu vào cho EGM phải được cấu hình là thiết bị UdpUc. Cấu hình này được thực hiện trong RobotStudio, sử dụng các thông số hệ thống của UDP Unicast Device trong mục Communication. Xác định tên, địa chỉ IP, cổng giao tiếp của thiết bị và đặt giao thức truyền tải thành UdpUc.

1.2.2.3. Tiếp cận cơ bản EGM Position Guidance

Các bước thực hiện: [1]

1. Di chuyển robot đến vị trí xác định.
2. Đăng ký EGM Client và nhận EGM identity. Identity này sau đó được sử dụng để liên kết thiết lập, kích hoạt, di chuyển, hủy kích hoạt… tới cách sử dụng EGM nhất định. Trạng thái EGM lúc này vẫn là EGM_STATE_DISCONNECTED.
3. Gọi lệnh thiết lập EGM EGMSetupUC để thiết lập thiết bị bên ngoài sử dụng kết nối giao thức UdpUc. Lúc này trạng thái EGM chuyển sang EGM_STATE_CONNECTED.
4. Chọn xem vị trí được đưa ra ở chế độ khớp hay chế độ tư thế và đưa ra tiêu chí hội tụ vị trí, có nghĩa là tiêu chí để xem khi nào robot đến vị trí được xem là đúng.
5. Nếu chọn chế độ tư thế, cần xác định hệ quy chiếu nào được dùng để xác định vị trí mục tiêu và chuyển động sẽ được áp dụng trong hệ quy chiếu nào.
6. Đặt chế độ dừng, thời gian chờ tùy chọn và tự thực hiện chuyển động. Lúc này trạng thái EGM là EGM_STATE_RUNNING. Đây là lúc robot đang di chuyển.
7. Chuyển động EGM sẽ dừng khi được coi là đã đạt đến vị trí mục tiêu, tức là các tiêu chí hội tụ được đáp ứng. Bây giờ, trạng thái EGM đã thay đổi lại thành EGM_STATE_CONNECTED.



Hình 1 4 Sơ đồ chuyển đổi trạng thái của EGM

1.2.2.4. Dữ liệu đầu vào

Dữ liệu đầu vào của EGM được sử dụng để hướng dẫn chuyển động robot từ một thiết bị bên ngoài thông qua các giao thức giao tiếp cụ thể, bao gồm giao diện tín hiệu và giao thức UDP Unicast (UdpUc). Việc lựa chọn nguồn dữ liệu được thực hiện thông qua các lệnh thiết lập EGM như sau:

Bảng 1 1 Danh sách các lệnh thiết lập dữ liệu đầu vào cho EGM

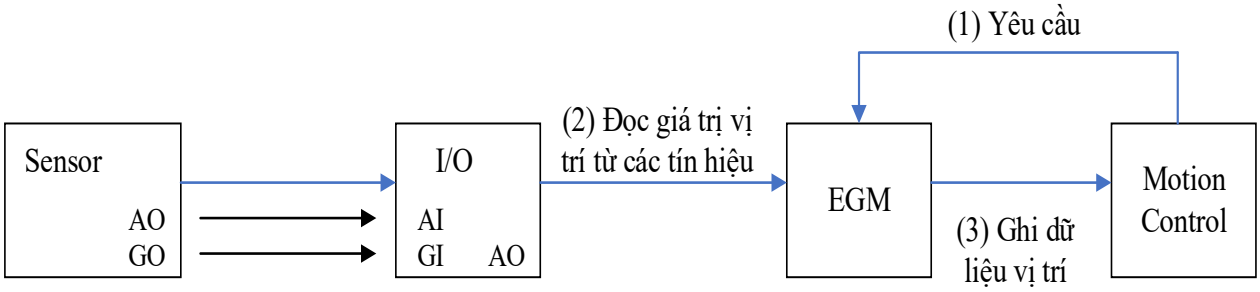
Lệnh	Chức năng
EGMSetupAI	Thiết lập tín hiệu đầu vào analog.
EGMSetupAO	Thiết lập tín hiệu đầu ra analog.
EGMSetupGI	Thiết lập tín hiệu group input.
EGMSetupUC	Thiết lập giao thức UdpUc.

a) Dữ liệu đầu vào qua giao diện tín hiệu

Trong chế độ này, robot nhận dữ liệu vị trí từ các tín hiệu I/O (AI, AO, GI) được cập nhật theo chu kỳ 4 ms:

+ Bộ cảm biến gửi giá trị vị trí (tọa độ hoặc góc khớp) tới các tín hiệu I/O.
+ EGM đọc các giá trị này và truyền tới hệ thống điều khiển chuyển động của robot.

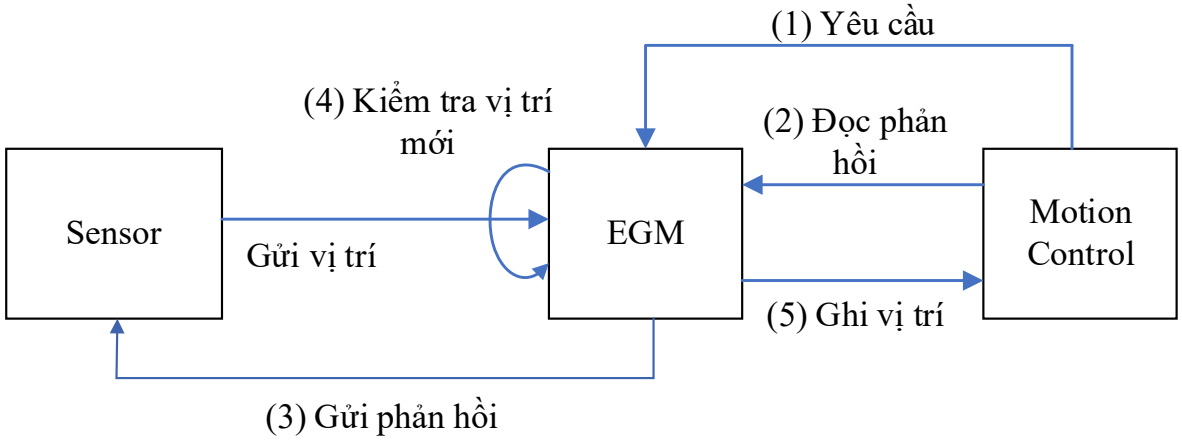
Số lượng dữ liệu đầu vào tối đa là 6 giá trị cho robot (có thể là 6 khớp hoặc 3 tọa độ x, y, z và 3 góc Euler rx, ry, rz) và tối đa 6 giá trị cho các trục phụ. Trong trường hợp robot 7 trục, trục phụ đầu tiên sẽ nhận giá trị từ tín hiệu đầu vào phụ.



Hình 1 5 Luồng dữ liệu cho giao diện tín hiệu

b) Dữ liệu đầu vào qua giao thức UDPUC

Khi sử dụng giao thức UdpUc, dữ liệu được truyền qua mạng dưới dạng gói UDP, với chu kỳ tối thiểu 4 ms:



Hình 1 6 Luồng dữ liệu cho giao diện UdpUc

1. Hệ thống điều khiển yêu cầu cập nhật vị trí.
2. EGM đọc dữ liệu phản hồi từ điều khiển chuyển động.
3. EGM gửi phản hồi về cho cảm biến.
4. EGM kiểm tra hàng đợi UDP để nhận dữ liệu mới.
5. Nếu có gói dữ liệu mới, EGM lấy dữ liệu và cập nhật vị trí cho điều khiển chuyển động. Nếu không, sử dụng giá trị trước đó.

1.2.2.5. Các hệ quy chiếu cho EGM Position Guidance

EGM có thể được vận hành ở hai chế độ khác nhau bao gồm chế độ khớp và chế độ tư thế. Các hệ quy chiếu dưới đây chỉ áp dụng cho chế độ tư thế. Lệnh Rapid EgmActPose định nghĩa tất cả các hệ quy chiếu có sẵn trong EGM

Bảng 1 2 Các hệ quy chiếu có sẵn trong EGM

Hệ quy chiếu	Mô tả
Tool	Dữ liệu tool sử dụng cho quá trình EGM được xác định bởi tham số \Tool.
Work object	Dữ liệu work object sử dụng cho quá trình EGM được xác định bởi tham số \Wobj
Correction	Hệ quy chiếu được sử dụng để xác định hướng di chuyển cuối cùng, được xác định bởi tham số bắt buộc CorrFrame
Sensor	Hệ quy chiếu được sử dụng để diễn tả dữ liệu cảm biến, được xác định bởi tham số bắt buộc SensorFrame

Đối với các hệ quy chiếu bắt buộc như CorrFrame và SensorFrame, cần phải xác định rõ xem chúng được định nghĩa tương ứng hệ quy chiếu nào. Thông tin này được xác định bằng cách sử dụng các hệ quy chiếu có sẵn, được định nghĩa trước trong kiểu dữ liệu egmframetype.

Bảng 1 3 Các hệ quy chiếu được định nghĩa trong kiểu dữ liệu egmframetype

Giá trị	Mô tả
EGM_FRAME_BASE	Hệ quy chiếu được định nghĩa so với base frame (pose mode)
EGM_FRAME_TOOL	Hệ quy chiếu được định nghĩa so với tool0 (pose mode)
EGM_FRAME_WOBJ	Hệ quy chiếu được định nghĩa so với work object đang hoạt động (pose mode)
EGM_FRAME_WORLD	Hệ quy chiếu được định nghĩa so với world frame (pose mode)
EGM_FRAME_JOINT	Các giá trị là giá trị khớp (joint mode)

1.2.2.6. Ưu điểm của EGM Position Guidance

1. Tốc độ cập nhật cao và độ trễ thấp

+ EGM Position Guidance cho phép đọc và ghi vị trí vào hệ thống chuyển động của robot với tần suất cao, mỗi 4 ms một lần, và độ trễ điều khiển vào khoảng 10–20 ms tùy thuộc vào loại robot.

+ Khoảng thời gian từ khi ghi một vị trí mới cho đến khi vị trí đó bắt đầu ảnh hưởng đến vị trí thực tế của robot thường là khoảng 20 ms.

2. Bỏ qua lập kế hoạch đường đi

+ EGM Position Guidance cung cấp giao diện cấp thấp cho bộ điều khiển robot, bằng cách bỏ qua kế hoạch đường đi, cho phép điều khiển chuyển động của robot một cách linh hoạt và phản hồi nhanh chóng.

+ Điều này đặc biệt hữu ích trong các ứng dụng yêu cầu phản ứng nhanh như hàn laser, theo dõi đường may, hoặc thao tác với các đối tượng chuyển động.

3. Linh hoạt trong chế độ điều khiển

+ EGM hỗ trợ hai chế độ điều khiển: Joint mode (theo góc trục) và Pose mode (theo tư thế).

+ Trong Pose mode, người dùng có thể xác định khung tham chiếu như Tool, Work Object, Correction Frame và Sensor Frame để điều khiển chuyển động của robot một cách chính xác.

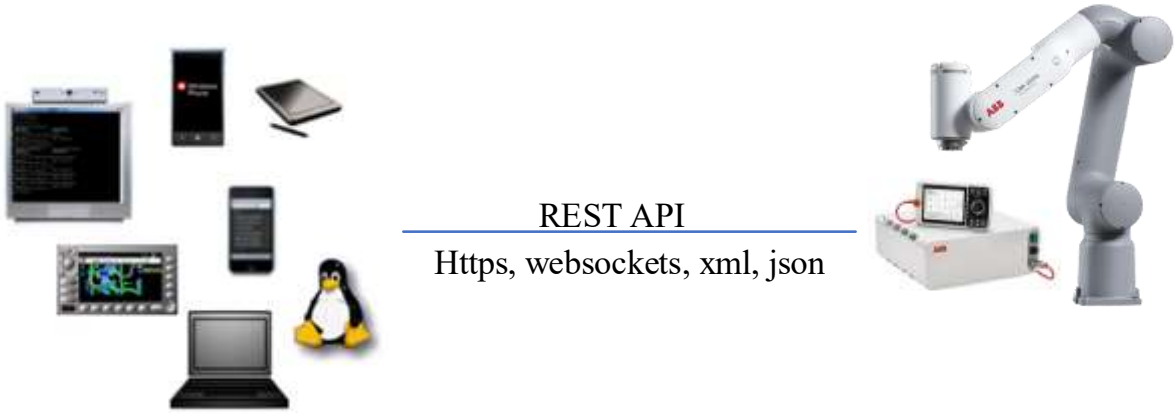
4. Tích hợp với các chế độ điều khiển khác

+ EGM Position Guidance có thể được kết hợp với các cài đặt logic hoặc kích hoạt các chế độ điều khiển khác như Force Control bằng cách sử dụng các tham số tùy chọn như \NoWaitCond.

5. Hỗ trợ giao tiếp linh hoạt

+ EGM hỗ trợ giao tiếp với thiết bị bên ngoài thông qua giao thức UdpUc, cho phép truyền và nhận dữ liệu vị trí một cách hiệu quả.

1.3. Tổng quan về Robot Web Services (RWS)



Hình 1 7 Mô hình giao tiếp RWS giữa robot ABB và thiết bị điều khiển qua REST API

1.3.1. Giới thiệu về Robot Web Services

Robot Web Services (RWS) là một hệ thống RESTful API do hãng ABB phát triển, cho phép các nhà phát triển xây dựng các ứng dụng tùy chỉnh để tương tác với bộ điều khiển robot thông qua giao thức HTTPS. Đây là thành phần quan trọng trong bộ điều khiển Omnicore và IRC5, cho phép các ứng dụng bên ngoài (PC, server, thiết bị điều khiển) giao tiếp với robot mà không cần phần mềm độc quyền như RobotStudio. [2]

RWS tuân theo kiến trúc REST API, một phong cách thiết kế phần mềm cho các API web. Cho phép lập trình viên truy cập và điều khiển robot ABB thông qua các URL và phương thức HTTP như:

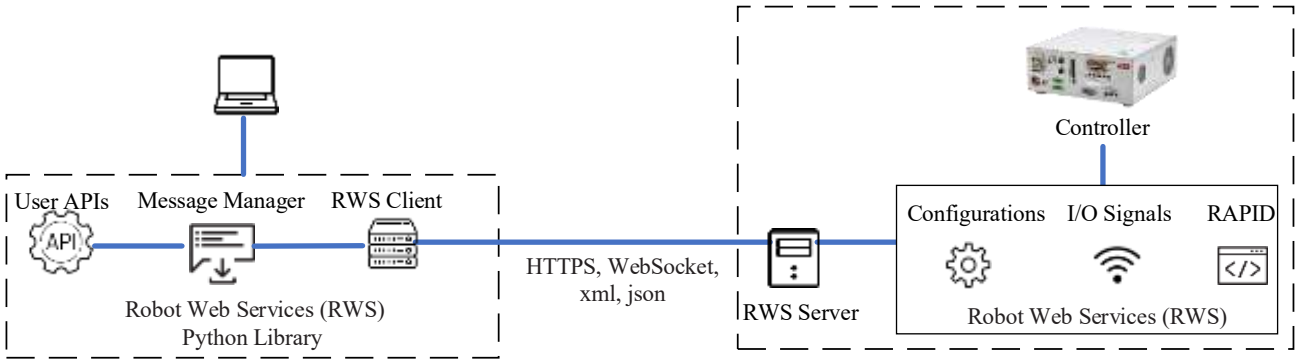
- GET: Lấy tài nguyên.
- PUT: Tạo hoặc cập nhật tài nguyên.
- POST: Cập nhập tài nguyên.
- DELETE: Xóa tài nguyên.
- OPTIONS: Được hỗ trợ phương thức bởi tài nguyên.

Để thao tác với các tài nguyên được xác định bằng URL. Dữ liệu được truyền dưới dạng XML hoặc JSON, cho phép các ứng dụng khách phân tích và xử lý thông tin một cách linh hoạt.

Ưu điểm của RWS và REST API:

- Giao tiếp qua mạng LAN/WiFi, không cần phần mềm riêng.
- Đơn giản, dễ tích hợp với hệ thống Python, JavaScript, Qt designer, ...

- Tương thích tốt với web, phần mềm điều khiển, và dịch vụ nền tảng đám mây.
- Không yêu cầu license riêng như RobotStudio Online.



Hình 1 8 Sơ đồ tổng quan về RWS

1.3.2. Điều kiện tiên quyết để bắt đầu với Robot Web Services

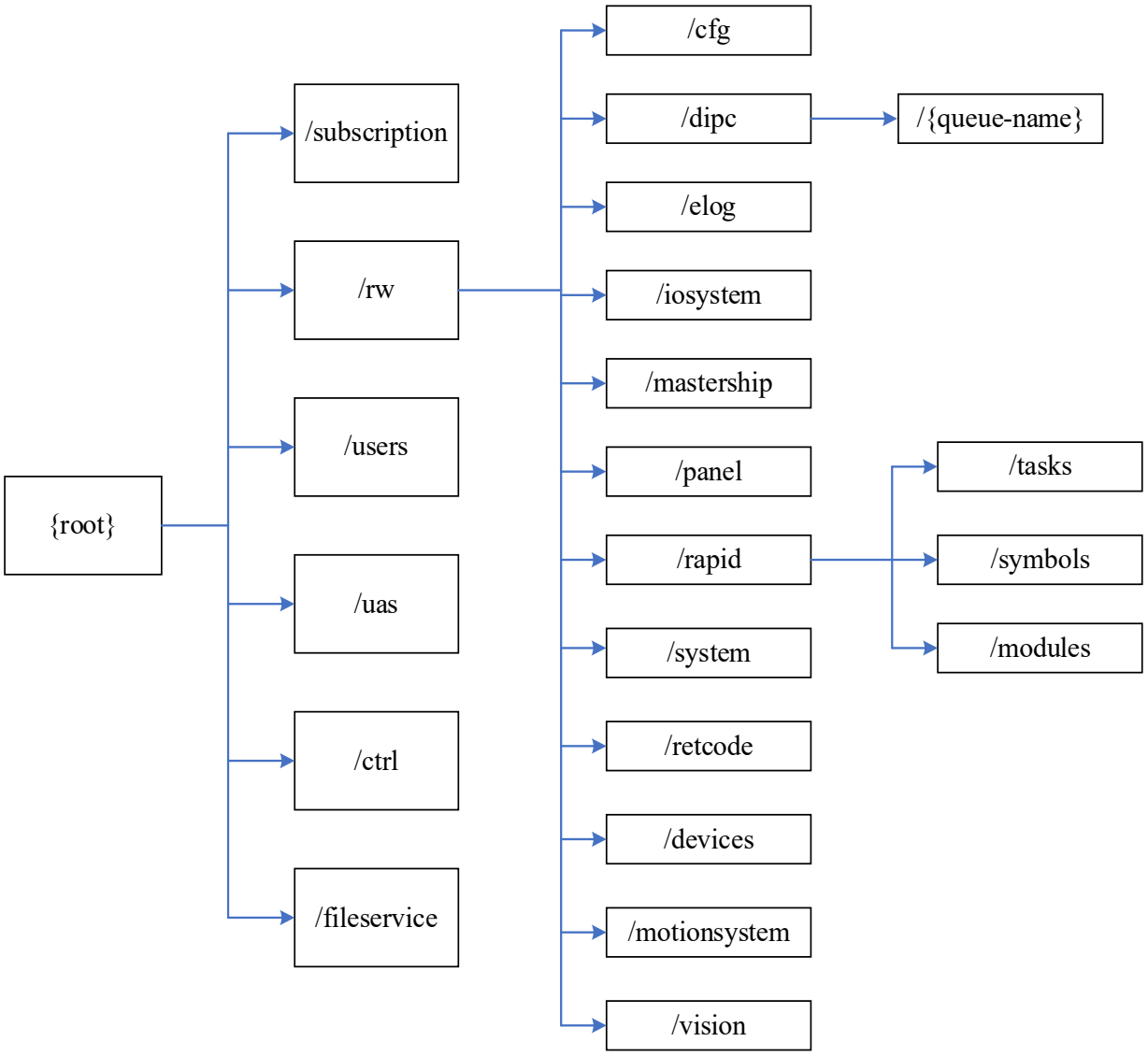
Yêu cầu người sử dụng [2]:

- Có hiểu biết về Hyper Text Transfer Protocol Secure (HTTPS).
- Có hiểu biết về XML hoặc JSON.
- Thư viện lập trình có thể khởi tạo HTTPS và phân tích cú pháp phản hồi.
- Một ứng dụng client, ví dụ như trình duyệt web thông thường.

1.3.3. Các dịch vụ

RWS cung cấp nhiều dịch vụ để quản lý và điều khiển robot, các dịch vụ chính của RWS [2]

- File Service: Cho phép truy cập và quản lý tệp tin trên bộ điều khiển từ xa, bao gồm tải lên, tải xuống, xóa và đổi tên tệp.
- Subscription Service: Hỗ trợ đăng ký theo dõi các thay đổi của tài nguyên và nhận sự kiện thông qua WebSockets, giúp cập nhật trạng thái theo thời gian thực.
- Ctrl Service: Quản lý các chức năng toàn cục của bộ điều khiển robot, như truy cập đồng hồ hệ thống, nhận diện bộ điều khiển và thực hiện khởi động lại.
- Users Service: Xử lý việc đăng ký và quản lý người dùng kết nối.
- RW (RobotWare) Services: Quản lý các dịch vụ của RobotWare như IO, RAPID, E-log và cấu hình hệ thống.



Hình 1 9 Sơ đồ các dịch vụ của Robot Web Services

1.3.4. Xác thực đăng nhập

Tất cả các máy khách sử dụng Robot Web Services đều phải đăng nhập bằng tên người dùng và mật khẩu. Tên người dùng và mật khẩu được định nghĩa trong hệ thống phân quyền người dùng (User Authorization System – UAS) của bộ điều khiển robot. Robot Web Services sử dụng phương thức digest làm phương thức xác thực mặc định, vì nó không gửi tên người dùng và mật khẩu dưới dạng văn bản thuần (clear text).

Khi một ứng dụng kết nối và gửi yêu cầu HTTP đầu tiên (chưa được xác thực), máy chủ HTTP trên bộ điều khiển robot sẽ phản hồi bằng lỗi HTTP "Unauthorized (401)". Sau đó, phía gửi yêu cầu phải gửi lại thông tin xác thực để được cấp quyền kết nối. Khi kết nối được chấp nhận, máy chủ sẽ gửi một cookie (tên là ABBCX) cho ứng

dụng client. Cookie ABBCX này được Robot Web Services sử dụng để duy trì phiên làm việc.

Ứng dụng client phải sử dụng một HTTP client hỗ trợ xác thực digest và có khả năng xử lý cookie để truy cập vào Robot Web Services. Các chi tiết này sẽ hiển thị cho lập trình viên khi sử dụng trình duyệt hoặc thư viện HTTP hỗ trợ digest và cookie. [2]

1.4. Giới thiệu chung về thị giác máy tính

Thị giác máy tính là một lĩnh vực của trí tuệ nhân tạo nhằm cho phép máy móc nắm bắt và diễn giải thông tin hình ảnh từ thế giới giống như con người. Thị giác máy tính nhằm mục đích tự động hóa hệ thống thị giác của con người để nhận dạng các vật thể, hiểu các cảnh và đưa ra phán đoán sau khi phân tích dữ liệu hình ảnh.

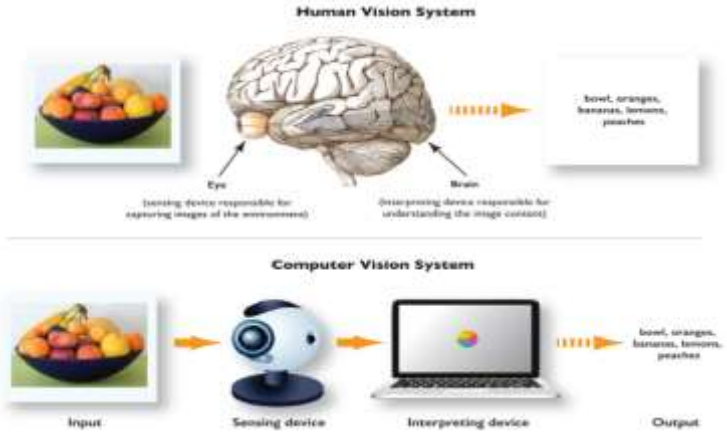
Hình 1 10 Ứng dụng thị giác máy tính trong cuộc sống

Trong hai thập kỷ qua, các ứng dụng thị giác máy tính đã tăng đáng kể nhờ những tiến bộ trong công nghệ camera, sự gia tăng của các nguồn tài nguyên tính toán và tính khả dụng của lượng dữ liệu lớn. Trong chăm sóc sức khỏe, thị giác máy tính hỗ trợ phát hiện sớm các bệnh gây tử vong như khối u não hoặc ung thư vú. Trong ô tô, nó cung cấp năng lượng cho xe tự lái để nhận dạng các vật thể và hiểu tình trạng đường sá. Sản xuất được hưởng lợi từ kiểm soát chất lượng tự động, trong khi bán lẻ sử dụng thị giác máy tính để quản lý hàng tồn kho và trải nghiệm mua sắm được cá nhân hóa. Các ứng dụng đa dạng của thị giác máy tính đang tăng hiệu quả và thúc đẩy sự đổi mới trong nhiều lĩnh vực khác nhau, mang lại lợi ích đáng kể cho thế giới.

1.4.1. Bản chất của Computer Vision

Computer Vision cung cấp cho máy tính khả năng của con người để nhìn, phân tích và diễn giải hình ảnh trong thế giới thực. Hãy tưởng tượng việc dạy máy tính nhận

dạng các vật thể, chẳng hạn như xác định xem một bức ảnh có chứa chó hay mèo hay nhận dạng từ video liệu môn thể thao đang chơi là khúc côn cầu hay cầu lông.



Hình 1 11 Hệ thống thị giác con người và hệ thống thị giác máy tính

Trong cuộc sống hàng ngày, chúng ta thường xuyên sử dụng các ứng dụng được hỗ trợ bởi thị giác máy tính. Một trong số đó là nhận dạng khuôn mặt trên điện thoại thông minh của chúng ta. Sau khi khuôn mặt của bạn được quét trong khi thiết lập ID khuôn mặt, thuật toán sẽ phát hiện các điểm chính trên khuôn mặt của bạn và lưu trữ thông tin để sử dụng trong tương lai. Một ví dụ khác là bộ lọc Snapchat cho phép bạn thêm tai chó hoặc mũ ngộ nghĩnh vào khuôn mặt của mình. Ngoài ra, bằng cách sử dụng tìm kiếm trực quan, người dùng có thể tải lên ảnh các sản phẩm họ thích khi mua sắm trực tuyến trên các trang web thương mại điện tử và hệ thống sẽ tìm thấy các sản phẩm tương tự. Do đó, thị giác máy tính đã đóng một vai trò không thể thiếu trong các nhiệm vụ thường ngày của chúng ta.

1.4.2. Cách thức hoạt động của Computer Vision

Có một số giai đoạn liên quan đến hoạt động cơ bản của thuật toán thị giác máy tính, như mô tả dưới đây.

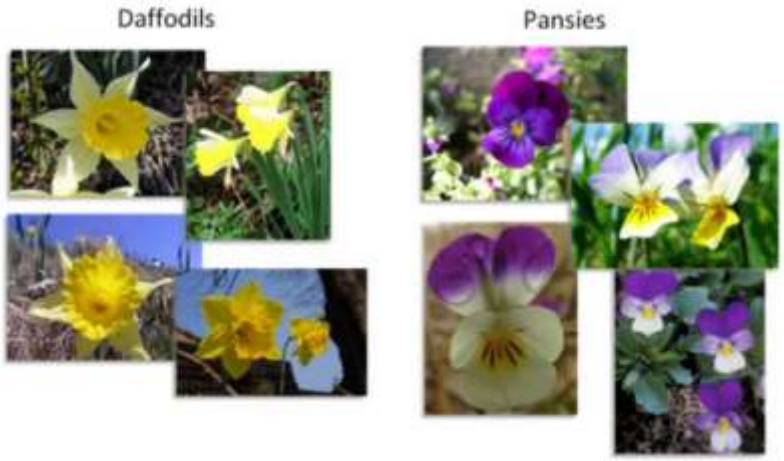
1. Thu thập hình ảnh: Bước đầu tiên là thu thập hình ảnh hoặc video bằng cách chụp bằng các thiết bị như máy ảnh hoặc cảm biến. Dữ liệu này là dữ liệu thô và đóng vai trò là dữ liệu đầu vào cho bước tiền xử lý tiếp theo.
2. Tiền xử lý dữ liệu: Dữ liệu thô được xử lý trước để loại bỏ nhiễu không mong muốn và nâng cao chất lượng hình ảnh, giúp dữ liệu phù hợp cho việc phân tích sâu hơn.
3. Trích xuất tính năng: Dữ liệu đã xử lý sau đó được chuyển đến các thuật toán thị giác máy tính để trích xuất các tính năng (mẫu) từ hình ảnh. Một số tính năng ban

đầu là các cạnh, hình dạng và kết cấu. Với các lần lặp lại tiếp theo của thuật toán, nó có thể phát hiện ra các tính năng nâng cao hơn.

4. Hiểu biết cấp cao: Giai đoạn cuối cùng của quá trình là hiểu biết cấp cao. Ở đây, các đặc điểm được trích xuất sẽ được suy luận để diễn giải hình ảnh. Đầu ra dựa trên nhiệm vụ cụ thể được thực hiện, chẳng hạn như phân loại, phát hiện hoặc nhiệm vụ khác.

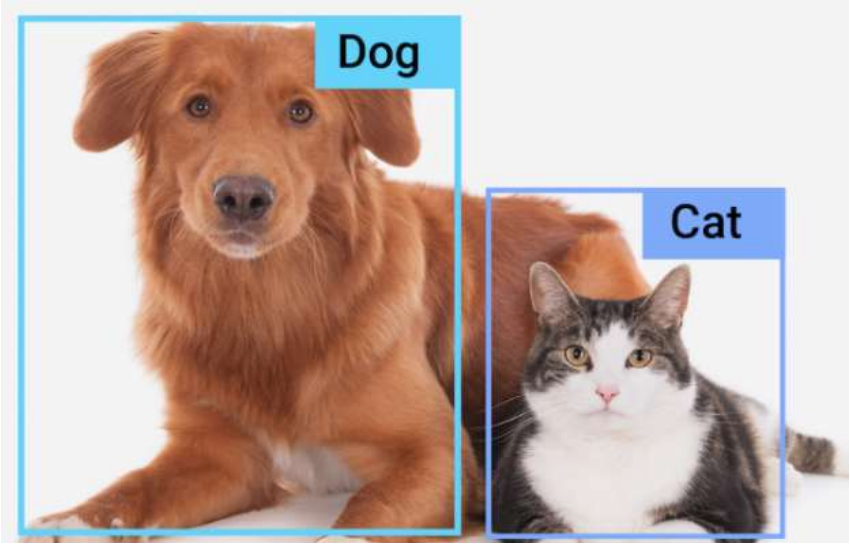
1.4.3. Nhiệm vụ chính của Computer Vision

+ Phân loại ảnh (Image Classification): Xác định nội dung tổng thể của ảnh, ví dụ: ảnh có chứa bàn tay hay không.



Hình 1 12 Phân loại hình ảnh bằng thị giác máy tính

+ Phát hiện đối tượng (Object Detection): Định vị đối tượng trong ảnh và xác định nhãn (label) tương ứng.

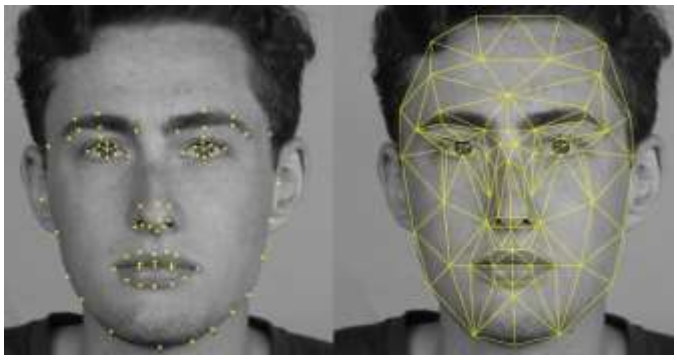


Hình 1 13 Phát hiện đối tượng bằng thị giác máy tính

+ Phân đoạn ảnh (Image Segmentation): Phân chia ảnh thành các vùng tương ứng với từng đối tượng cụ thể.

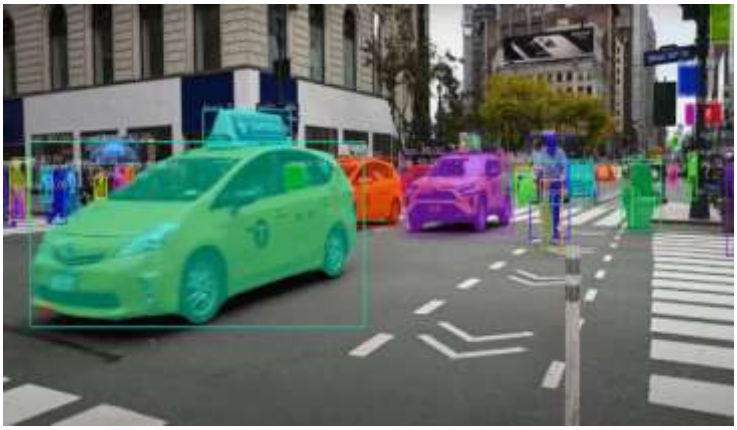
Hình 1 14 Phân đoạn hình ảnh bằng thị giác máy tính

+ Nhận diện khuôn mặt/cử chỉ: Nhận dạng biểu cảm, danh tính hoặc cử chỉ tay và hành động con người.



Hình 1 15 Nhận dạng khuôn mặt bằng thị giác máy tính

+ Theo dõi đối tượng (Tracking): Duy trì vị trí của đối tượng qua từng khung hình video liên tiếp.



Hình 1 16 Theo dõi đối tượng bằng thị giác máy tính

1.5. Giới thiệu chung về ngôn ngữ và công cụ lập trình (rút ngắn)

1.5.1. Tổng quan về ngôn ngữ lập trình Python

Python là một ngôn ngữ lập trình bậc cao, được phát triển bởi Guido van Rossum và ra mắt lần đầu vào năm 1991. Python được nhấn mạnh sự rõ ràng trong cú pháp và khả năng đọc hiểu mã nguồn, giúp người dùng lập trình nhanh gọn mà vẫn đảm bảo tính hiệu quả và dễ bảo trì. Python không yêu cầu người lập trình khai báo kiểu dữ liệu một cách cứng nhắc, từ đó tạo điều kiện thuận lợi cho cả người mới bắt đầu và các chuyên gia trong lĩnh vực công nghệ.

Hiện nay, Python được sử dụng rộng rãi trong nhiều lĩnh vực khác nhau như: phát triển phần mềm, thiết kế website, trí tuệ nhân tạo (AI), học máy (machine learning), phân tích dữ liệu lớn (big data), tự động hóa, tài chính, khoa học kỹ thuật và giáo dục. Hệ sinh thái phong phú của Python với hàng nghìn thư viện mạnh mẽ (standard library) giúp người dùng có thể dễ dàng xây dựng các ứng dụng từ đơn giản đến phức tạp.

1.5.2. Tổng quan về phần mềm Visual Studio

Visual Studio là một môi trường phát triển tích hợp (Integrated Development Environment - IDE) do Microsoft phát triển, được thiết kế để hỗ trợ lập trình viên xây dựng các ứng dụng đa dạng trên nhiều nền tảng khác nhau như Windows, web, di động, và đám mây. Một trong những điểm nổi bật của Visual Studio là công cụ kéo và thả (drag-and-drop) trực quan, lập trình viên dễ dàng thêm các thành phần giao diện người dùng như nút bấm, ô nhập liệu, hộp thoại, danh sách lựa chọn và nhiều điều khiển khác mà không cần viết mã thủ công phức tạp.

Visual Studio hỗ trợ nhiều ngôn ngữ lập trình phổ biến như C#, C++, Python, JavaScript và nhiều ngôn ngữ khác, đồng thời tích hợp mạnh mẽ với các framework và thư viện giúp mở rộng khả năng phát triển ứng dụng. Ngoài ra, công cụ này còn cung cấp hệ thống quản lý dự án, gỡ lỗi (debug), kiểm thử (testing), và triển khai ứng dụng, mang đến trải nghiệm phát triển phần mềm toàn diện cho lập trình viên.

1.5.3. Tổng quan về ngôn ngữ lập trình RAPID

RAPID là ngôn ngữ lập trình chính thức được phát triển bởi ABB dành cho các robot công nghiệp của hãng. Với cú pháp rõ ràng, linh hoạt và dễ tiếp cận, RAPID giúp lập trình viên dễ dàng xây dựng các chương trình tự động hóa phức tạp, kiểm soát chuyển động robot, xử lý dữ liệu cảm biến và tương tác với thiết bị ngoại vi. Ngôn ngữ này hỗ trợ các khái niệm lập trình hiện đại như điều khiển luồng, xử lý sự kiện và quản lý biến,

tương tự các ngôn ngữ bậc cao thông dụng. RAPID cũng được tích hợp chặt chẽ với phần mềm lập trình robot RobotStudio của ABB, cho phép lập trình viên mô phỏng, kiểm thử và tối ưu chương trình trên máy tính trước khi triển khai trên robot thực tế. Nhờ đó, việc phát triển và vận hành robot trở nên nhanh chóng và an toàn hơn.

1.5.4. Tổng quan về phần mềm RobotStudio

RobotStudio là phần mềm mô phỏng và lập trình robot do hãng ABB phát triển, được thiết kế nhằm hỗ trợ lập trình viên và kỹ sư tự động hóa trong việc phát triển, thử nghiệm và tối ưu hóa các ứng dụng robot công nghiệp mà không cần phải trực tiếp thao tác trên robot vật lý. RobotStudio có những tính năng chính sau:

1. Lập trình robot bằng ngôn ngữ RAPID: RobotStudio cho phép người dùng lập trình trực tiếp các robot ABB bằng ngôn ngữ RAPID. Tạo các lệnh điều khiển chuyển động, xử lý logic điều kiện, vòng lặp, và tương tác với thiết bị ngoại vi một cách chính xác và linh hoạt.
2. Tối ưu hóa đường đi và hiệu suất: RobotStudio tích hợp công cụ phân tích và tối ưu hóa đường đi (path optimization), giúp cải thiện hiệu suất vận hành. Người dùng có thể điều chỉnh tốc độ, giảm thời gian chu kỳ, tối ưu hoá các điểm dừng và góc di chuyển để nâng cao độ chính xác và hiệu quả sản xuất.
3. Mô phỏng thời gian thực (Virtual Controller): Tính năng Virtual Controller mô phỏng chính xác bộ điều khiển thực tế của robot ABB, cho phép chương trình hoạt động giống hệt như khi chạy trên robot thật. Điều này giúp giảm thiểu sự khác biệt giữa môi trường lập trình và môi trường sản xuất thực tế.
4. Hỗ trợ lập kế hoạch ngoại tuyến (Offline Programming): Người dùng có thể lập trình, mô phỏng và kiểm thử robot mà không cần dừng dây chuyền sản xuất hoặc có mặt tại nhà máy. Điều này giúp tiết kiệm thời gian và chi phí đáng kể trong quá trình phát triển.
5. Tích hợp thiết bị và hệ thống ngoại vi: RobotStudio hỗ trợ kết nối với nhiều loại thiết bị ngoại vi như cảm biến, băng chuyền, máy CNC, máy quét 3D, PLC và các hệ thống tự động hóa khác. Từ đó, robot có thể tương tác và phối hợp linh hoạt trong môi trường sản xuất thực tế.
6. Giao diện trực quan và dễ sử dụng: Phần mềm có giao diện đồ họa thân thiện, hỗ trợ thao tác kéo – thả, gắn kết mô hình CAD và các công cụ dựng hình, giúp kỹ

sư dễ dàng xây dựng mô hình làm việc nhanh chóng mà không cần phải viết mã hoàn toàn bằng tay.

1.6. Kết luận chương 1

Trong chương này, nhóm chúng tôi đã trình bày tổng quan các nền tảng lý thuyết và công nghệ cốt lõi phục vụ nghiên cứu xây dựng hệ thống tương tác giữa người và robot ABB CRB15000, với trọng tâm là thị giác máy tính và giao tiếp thời gian thực thông qua các giao thức EGM và RWS. Những nội dung trên đã cung cấp cái nhìn toàn diện và tạo nên cơ sở vững chắc cho việc xây dựng và phát triển hệ thống tương tác giữa người và robot ABB CRB15000. Trong các chương tiếp theo, nghiên cứu sẽ tập trung vào thiết kế hệ thống và triển khai giải pháp tích hợp các công nghệ đã trình bày vào ứng dụng điều khiển thực tế.

Chương 2: XÂY DỰNG HỆ THỐNG NHẬN DIỆN CỬ CHỈ TAY ĐỂ ĐIỀU KHIỂN ROBOT

2.1. OpenCV – Thư viện xử lý ảnh mã nguồn mở

OpenCV (Open-Source Computer Vision Library) là thư viện mã nguồn mở chuyên xử lý ảnh, video và thị giác máy tính, hỗ trợ đa nền tảng như Windows, Linux, macOS và các ngôn ngữ lập trình phổ biến như C++, Python, Java. Thư viện cung cấp hàng nghìn hàm xử lý ảnh, bao gồm chuyển đổi hệ màu, làm mịn, phát hiện cạnh, phân đoạn, biến đổi hình học, trích xuất đặc trưng, phát hiện chuyển động, đồng thời hỗ trợ giao tiếp thời gian thực với camera.

Các chức năng chính OpenCV:

+ Xử lý ảnh cơ bản: Chuyển đổi hệ màu, làm mịn, phát hiện cạnh, phân đoạn ảnh.

+ Phát hiện và nhận dạng đối tượng: Sử dụng các thuật toán như Haar Cascade, HOG, SVM.

+ Theo dõi đối tượng: Sử dụng các thuật toán như MeanShift, CamShift, KLT.

+ Xử lý video: Đọc, ghi, và xử lý video theo thời gian thực.

+ Tích hợp với các thư viện khác: Dễ dàng tích hợp với NumPy, SciPy, TensorFlow, …

Đối với các công nghệ hiện đại, OpenCV cũng là một yếu tố không thể thiếu. Tất cả những ứng dụng công nghệ như robot, xe tự lái, bảng cảm ứng thông minh… đều có sự góp mặt của OpenCV trong khâu xử lý hình ảnh. Ví dụ gần gũi nhất trong cuộc sống có thể kể đến hệ thống mở khóa điện thoại bằng cách nhận diện khuôn mặt người dùng…

2.2. MediaPipe – Nền tảng nhận diện cử chỉ thời gian thực

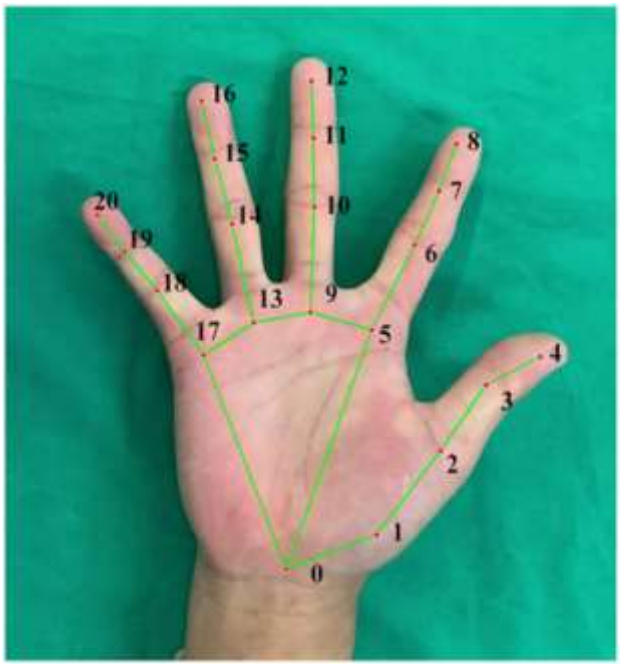
MediaPipe là một framework mã nguồn mở do Google phát triển, thiết kế để xây dựng các pipeline xử lý đa phương tiện như ảnh, video, âm thanh theo thời gian thực. MediaPipe đặc biệt nổi bật nhờ khả năng chạy hiệu quả trên cả thiết bị di động, máy tính để bàn và trong môi trường trình duyệt.

Các module nổi bật:

Bảng 2 1 Một số module nổi bật của thư viện Mediapipe

Module	Chức năng
Hand Tracking (Hands)	Phát hiện và theo dõi 21 điểm trên bàn tay
Pose	Nhận dạng tư thế toàn thân (33 điểm)
Face Mesh	Phát hiện lưới 3D trên khuôn mặt (~468 điểm)
Holistic	Kết hợp cả 3 module trên
Objectron	Phát hiện và định vị vật thể 3D

Một trong những mô-đun mạnh nhất của MediaPipe là MediaPipe Hands, có khả năng phát hiện vị trí lòng bàn tay bằng mô hình CNN. Dự đoán tọa độ 3D của 21 điểm xương trên bàn tay (gọi là landmarks). Theo dõi liên tục khung hình để giảm độ trễ.



Hình 2 1 Mô hình nhận diện các điểm đặc trưng trên bàn tay của MediaPipe

Cấu trúc pipeline MediaPipe Hands

+ Palm Detection: Phát hiện bàn tay.
+ Hand Landmark Model: Xác định 21 điểm xương tay.
+ Tracking: Duy trì vị trí tay liên tục trong video.

Ưu điểm

+ Hiệu suất cao, chạy real-time trên CPU
+ Cung cấp mô hình sẵn, dễ tích hợp vào các ứng dụng GUI và robot nhúng
+ Hỗ trợ Python, C++, Android, Web

2.3. Lập trình xử lý ảnh

2.3.1. Các bước nhận dạng cử chỉ tay để điều khiển robot

Bước 1: Phát hiện bàn tay

Sử dụng mô hình học sâu Mediapipe Hands, được huấn luyện trên bộ dữ liệu đa dạng với các kiểu tay và môi trường khác nhau. Vì đầu vào khung hình ở dạng BRG, cần chuyển sang định dạng RBG để phù hợp với mô hình nhận dạng bàn tay của thư viện Mediapipe. Ở bước này, có thể giới hạn số lượng bàn tay phát hiện bằng tham số max_num_hands.

Bước 2: Trích xuất các điểm đặc trưng của bàn tay

Từ mỗi bàn tay phát hiện, xác định 21 điểm đặc trưng (landmarks) đại diện cho các khớp ngón tay, đầu ngón tay và lòng bàn tay. Mỗi điểm sẽ có tọa độ tương đối (x, y) và sẽ được chuyển thành tọa độ tuyệt đối (cx, cy) dựa trên kích thước khung hình. Chúng tôi sử dụng mô hình Landmark của MediaPipe Hands. Mô hình này nhận đầu vào là ảnh đã cắt chứa bàn tay và trả về tọa độ 3D tương đối của 21 điểm, được lưu dưới dạng [id, cx, cy]. Sau đó, các điểm đặc trưng và các mối liên kết giữa chúng sẽ được vẽ lên khung hình gốc bằng module drawing_utils.

Bước 3: Nhận dạng cử chỉ

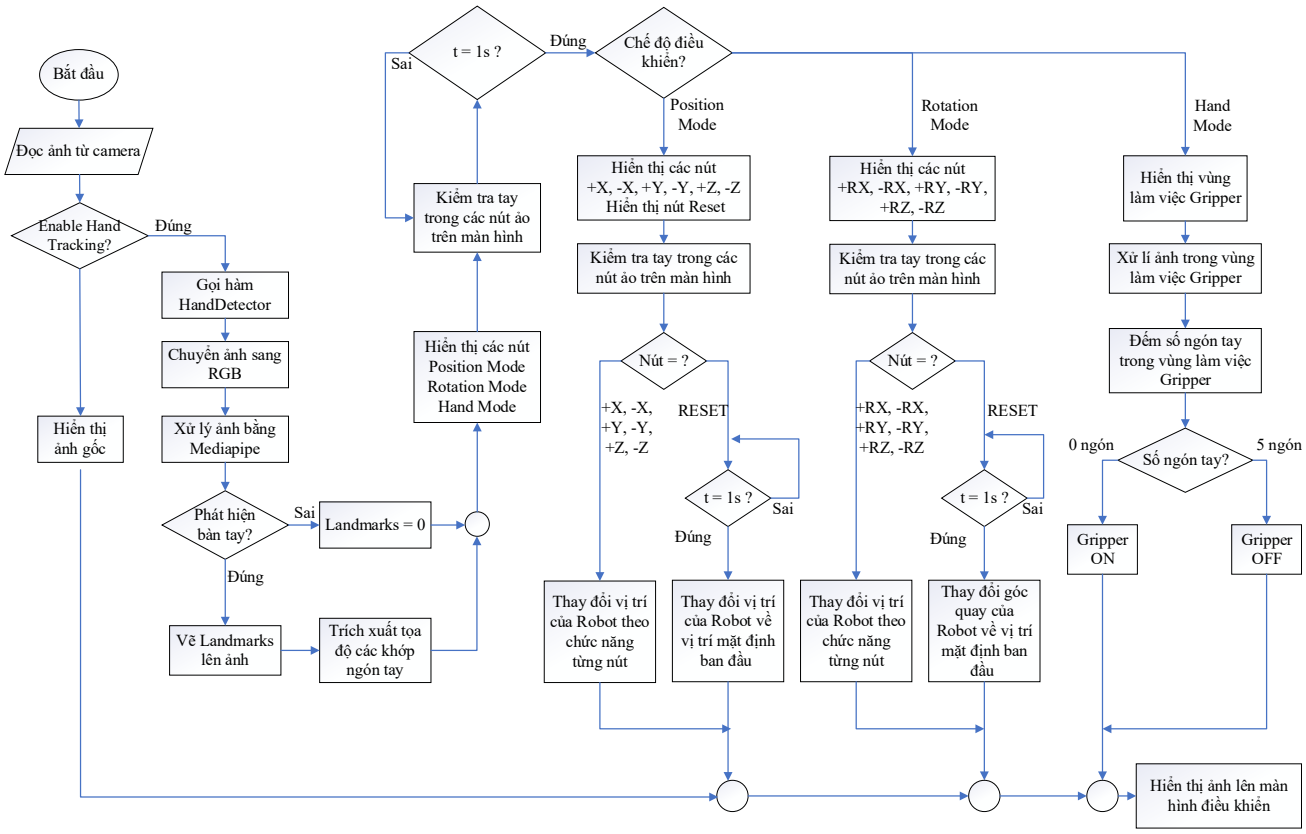
Dựa trên các điểm đặc trưng đã được trích xuất, phân loại cử chỉ tay thành các lệnh điều khiển gripper bao gồm nắm/mở bàn tay. Để làm điều này, chúng tôi sử dụng các điểm đặc trưng tương ứng với năm đầu ngón tay. Đầu tiên là ngón cái, do có sự khác biệt so với các ngón còn lại, trạng thái của nó sẽ được kiểm tra bằng cách so sánh tọa độ của đầu ngón cái và điểm nay dưới nó. Đối với bốn ngón còn lại sẽ so sánh tọa độ của đầu ngón tay với tọa độ điểm khớp ngay giữa ngón.

Bước 4: Xác định cử chỉ để điều khiển robot

+ Điều khiển Gripper: Tạo một khung chữ nhật kích thước cố định, nếu bàn tay được phát hiện trong khung chữ nhật, hệ thống sẽ bắt đầu nhận diện cử chỉ nắm/mở tương ứng để điều khiển đóng/mở Gripper của robot.

+ Điều khiển vị trí/góc quay của robot bằng các nút nhấn ảo: Vẽ các nút ảo lên khung hình, nếu tay nằm trong vùng nút và giữ lâu hơn 1 giây thì hệ thống sẽ xác định lệnh tương ứng với từng nút, từ đó điều khiển vị trí/góc quay của robot.

2.3.2. Lưu đồ thuật toán



Hình 2 2 Lưu đồ thuật toán quy trình xử lý ảnh, nhận diện cử chỉ tay

2.4. Kết quả đạt được

Hệ thống nhận diện cử chỉ tay được xây dựng trên nền tảng OpenCV và MediaPipe đã đạt được một số chức năng chính như sau:

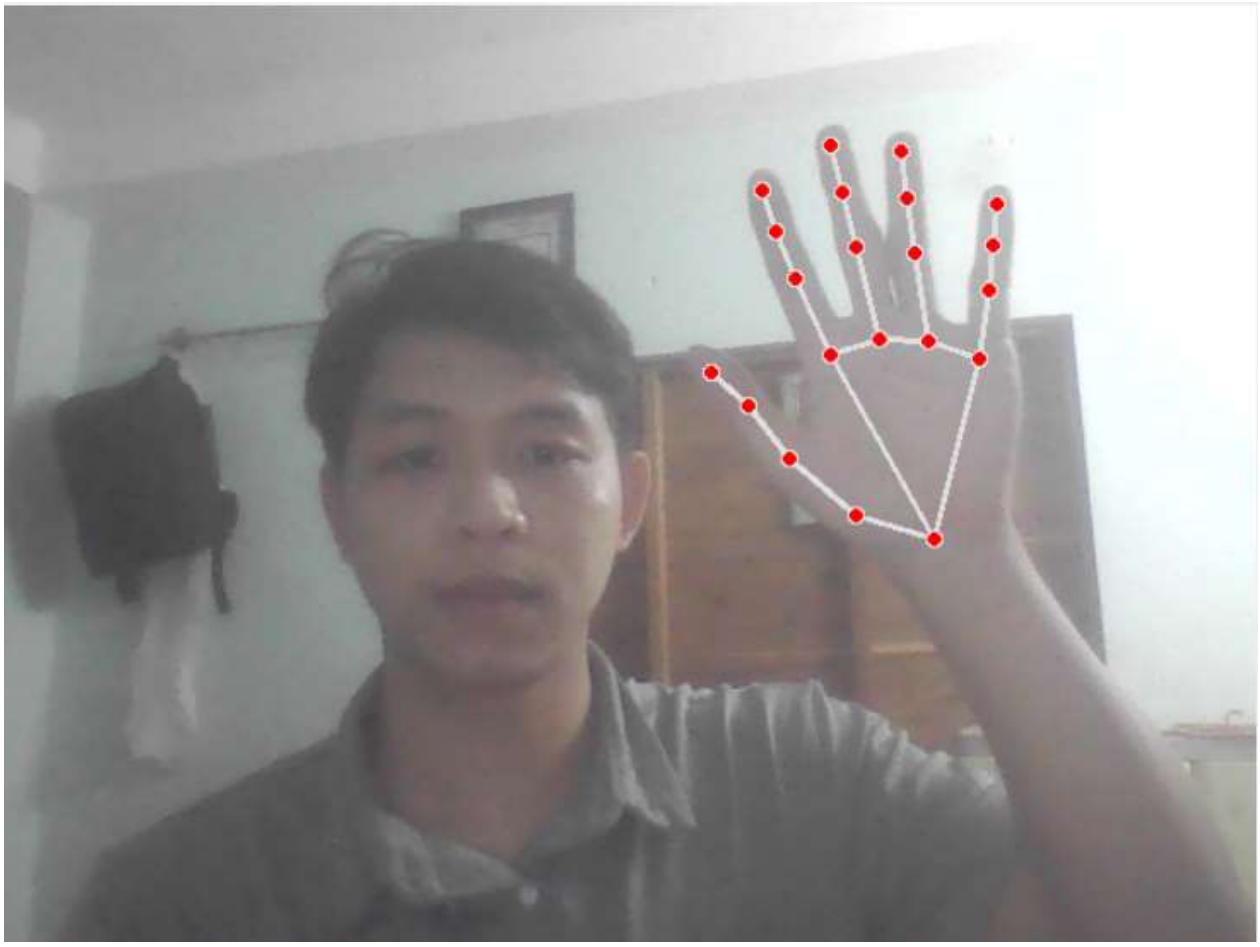
1. Nhận diện bàn tay và cử chỉ thời gian thực

Hệ thống sử dụng module MediaPipe Hands để phát hiện bàn tay và xác định 21 điểm đặc trưng (landmarks) trên mỗi bàn tay với độ chính xác cao. Việc xử lý được thực hiện theo thời gian thực với độ trễ thấp, đảm bảo cho phản hồi điều khiển robot mượt mà và liên tục. Bao gồm các chức năng sau:

+ Phát hiện bàn tay trong khung hình video từ webcam.

+ Theo dõi cử động tay liên tục thông qua tọa độ landmark.

+ Xác định cử chỉ dựa trên hình học và mối quan hệ không gian giữa các điểm đặc trưng.



Hình 2 3 Phát hiện, theo dõi và nhận dạng cử chỉ tay

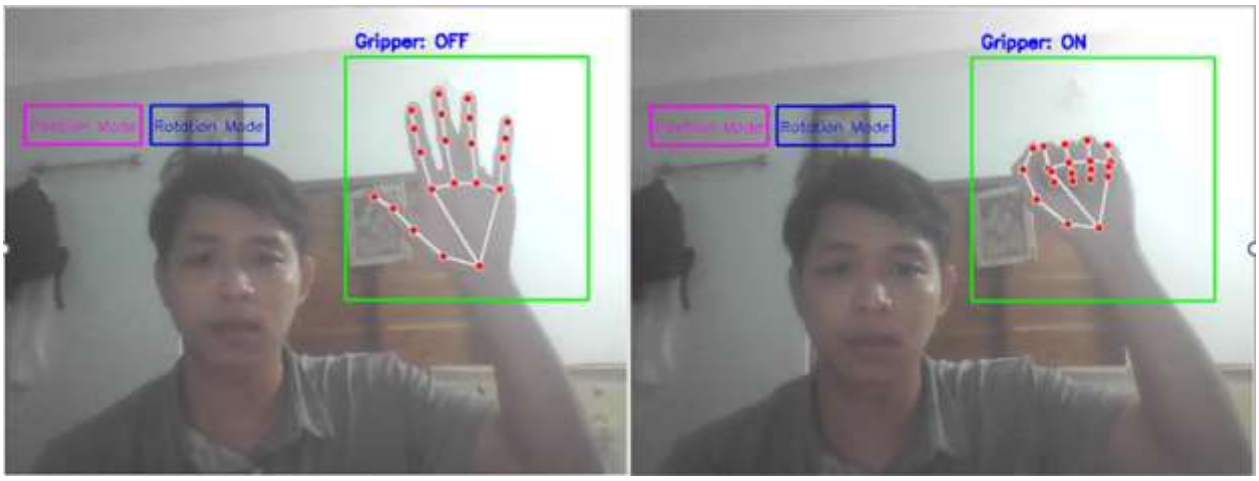
2. Điều khiển đóng/mở gripper robot bằng cử chỉ

Một trong những chức năng quan trọng là điều khiển gripper của robot thông qua nhận dạng cử chỉ tay cụ thể:

+ Cử chỉ bàn tay mở → Lệnh mở gripper.

+ Cử chỉ nắm tay → Lệnh đóng gripper.

Các cử chỉ này được phân loại bằng cách phân tích khoảng cách giữa các ngón tay và trạng thái đóng/mở của chúng thông qua landmark bàn tay.



Hình 2 4 Đóng/Mở Gripper của robot bằng cử chỉ nắm/thả tay

3. Điều khiển vị trí/góc quay robot bằng các nút ảo trên giao diện camera

Hệ thống tích hợp các nút điều khiển ảo hiển thị trực tiếp trên khung hình camera, cho phép người dùng điều khiển vị trí của robot bằng cách chạm và đưa tay vào các vùng định sẵn trên màn hình. Các chức năng bao gồm:

+ Các nút ảo để điều chỉnh tăng giảm các thông số vị trí (x, y, z) và góc quay (rx, ry, rz) của robot.

+ Các nút ảo để tùy chọn chế độ điều khiển như chế độ điều khiển gripper, chế độ điều khiển vị trí, chế độ điều khiển góc quay.

+ Giao diện trực quan hỗ trợ phản hồi trực tiếp khi tay chạm vào nút ảo.

Cơ chế này giúp nâng cao khả năng tương tác tự nhiên, loại bỏ sự cần thiết của các thiết bị ngoại vi truyền thống như bàn phím hay chuột.

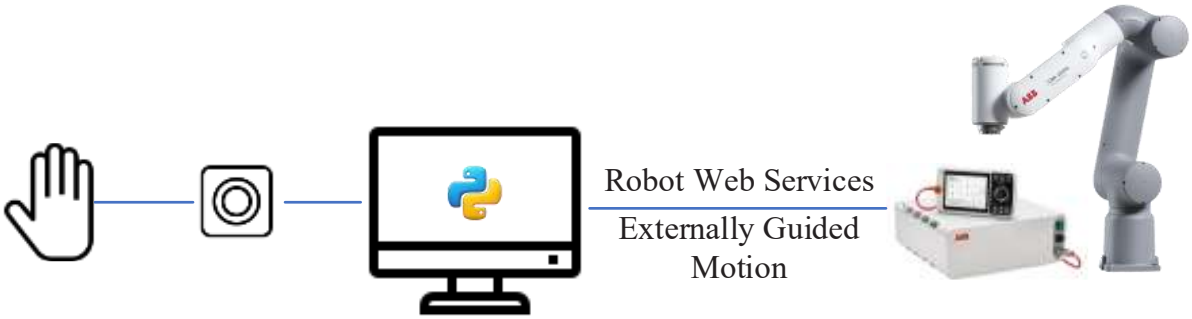
Hình 2 5 Điều khiển vị trí và góc quay của robot bằng các nút nhấn ảo

2.5. Kết luận chương 2

Trong chương này, chúng tôi đã hoàn thành việc lập trình xử lý ảnh, nhận diện cử chỉ tay để điều khiển vị trí, góc quay và đóng mở gripper của robot. Hệ thống kết hợp giữa MediaPipe và OpenCV đã cho thấy tính ứng dụng thực tiễn cao trong lĩnh vực điều khiển robot dựa trên thị giác máy tính và nhận dạng cử chỉ. Các chức năng như nhận diện bàn tay, phân loại cử chỉ và tương tác với nút nhấn ảo đã hoạt động ổn định trong môi trường thử nghiệm, mở ra tiềm năng ứng dụng trong điều khiển robot công nghiệp, dịch vụ hoặc hỗ trợ người khuyết tật…

Chương 3: XÂY DỰNG HỆ THỐNG GIAO TIẾP GIỮA CAMERA-PC - ROBOT

3.1. Tổng quát toàn bộ hệ thống



Hình 3 1 Sơ đồ tổng quát toàn bộ hệ thống

Hệ thống tương tác người – máy trong đề tài này được xây dựng theo mô hình giao tiếp phân tầng giữa ba khối chính: Camera – PC – Robot ABB CRB15000. Dưới đây là mô tả chi tiết các thành phần chính và chức năng của từng phần trong hệ thống:

Bảng 3 1 Các thành phần chính trong hệ thống

Thành phần	Vai trò chính
Camera	Thu nhận ảnh đầu vào
OpenCv	Tiền xử lý ảnh
MediaPipe	Nhận diện và trích xuất cử chỉ tay
Thuật toán điều khiển	Phân tích cử chỉ và nhận lệnh cho robot
PyQt5	Giao diện người dùng, hiển thị video và điều khiển
EGM/RWS	Giao tiếp giữa PC và Robot
Bộ điều khiển Robot	Nhận lệnh và điều phối hoạt động robot
Robot ABB CRB15000	Thực thi hành động vật lí

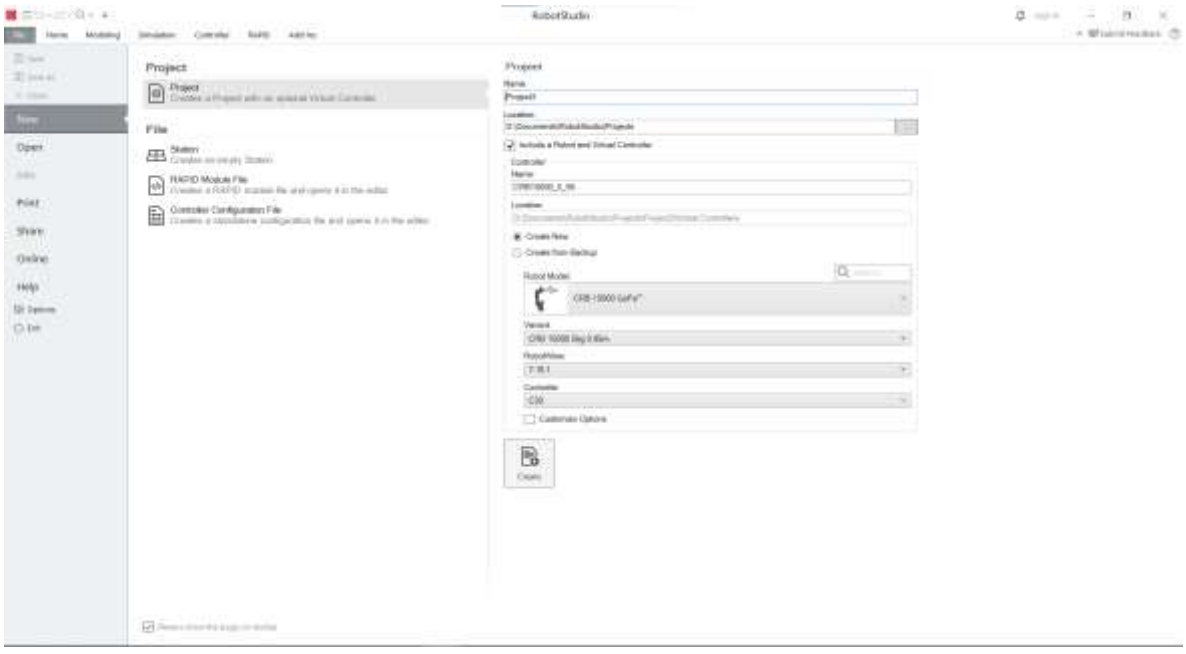
Hệ thống điều khiển robot ABB CRB15000 bằng cử chỉ tay thời gian thực bao gồm ba thành phần chính: camera, máy tính xử lý trung tâm và robot ABB CRB15000. Camera có nhiệm vụ thu nhận hình ảnh liên tục của tay người dùng với tốc độ khoảng 30 khung hình/giây và truyền dữ liệu hình ảnh này tới máy tính. Tại máy tính, ảnh được xử lý sơ bộ bằng thư viện OpenCV để chuyển đổi không gian màu, sau đó được đưa vào

mô hình MediaPipe Hands để phát hiện bàn tay và trích xuất 21 điểm đặc trưng (landmarks). Dựa trên các đặc trưng hình học này, hệ thống xác định cử chỉ tương ứng và đưa ra các lệnh điều khiển robot (Ví dụ: di chuyển, xoay, mở hoặc đóng gripper). Các lệnh điều khiển vị trí/góc quay được mã hóa và gửi tới bộ điều khiển Omnicore C30 của robot ABB thông qua giao thức EGM (Externally Guided Motion) sử dụng UDP, với tốc độ cập nhật khoảng 250Hz, cho phép robot phản hồi gần như tức thời theo tay người điều khiển. Đồng thời, các lệnh điều khiển hệ thống như bật/tắt động cơ, chọn chế độ hoạt động (Auto/Man), chạy/dừng/reset chương trình hoặc đóng/mở gripper được gửi qua giao thức RWS (Robot Web Services) sử dụng HTTPS. Tất cả quá trình xử lý, truyền lệnh và phản hồi đều được giám sát trên giao diện người dùng được lập trình bằng thư viện PyQt5, nơi hiển thị video thời gian thực và các nút ảo hỗ trợ người vận hành điều khiển robot bằng cử chỉ một cách trực quan và hiệu quả. Toàn bộ hệ thống vận hành liên tục, đảm bảo độ trễ thấp và hoạt động ổn định trong môi trường công nghiệp.

3.2. Cấu hình EGM và lập trình Robot trên RobotStudio

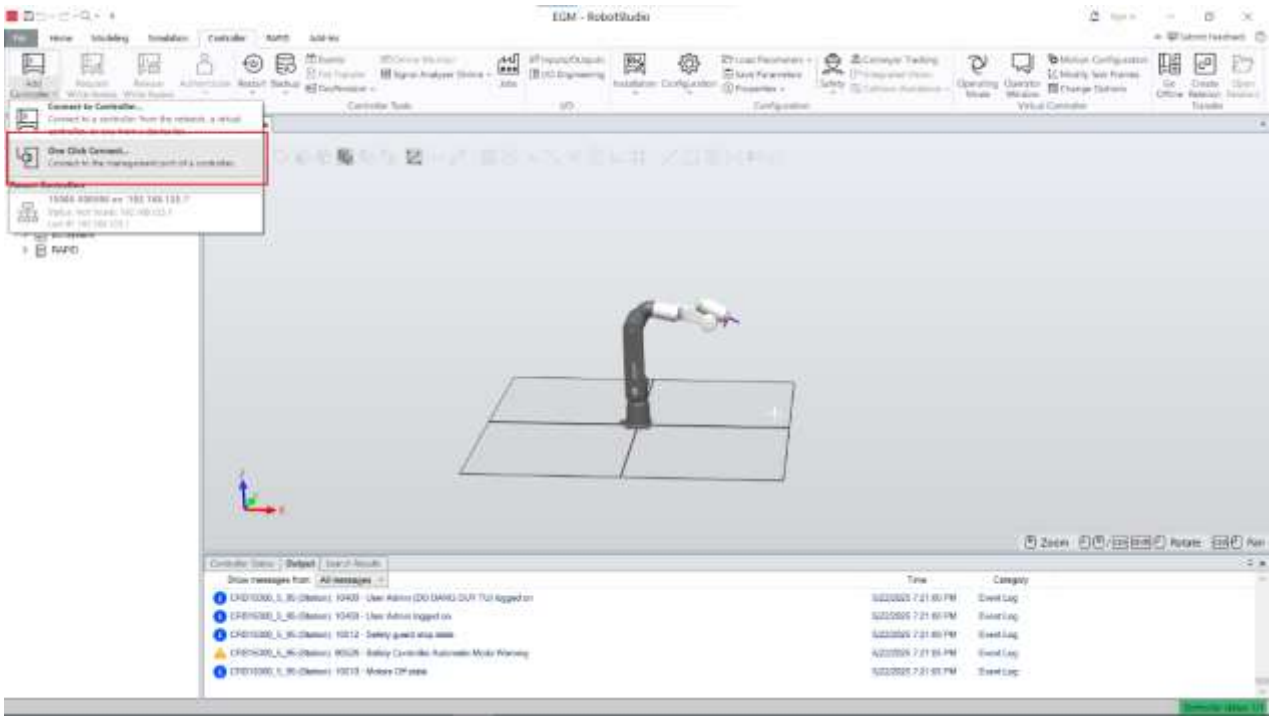
3.2.1. Các cấu hình cần thiết cho EGM

Bước 1: Tạo mới một Solution. Tùy chọn bao gồm Robot và Bộ điều khiển ảo nếu sử dụng mô phỏng.



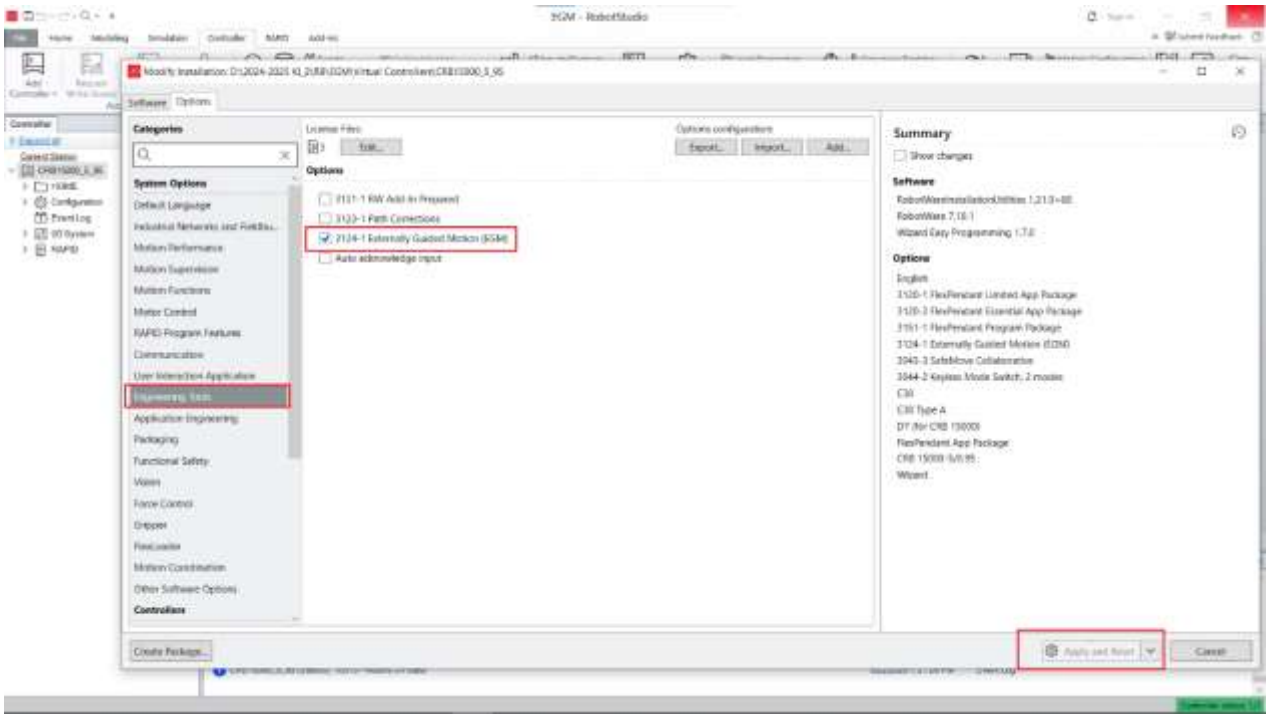
Hình 3 2 Tạo mới Solution trên RobotStudio

Nếu sử dụng Robot và Bộ điều khiển thật, sau khi tạo Solution, ta tiến hành thêm bộ điều khiển bằng cách chọn vào tab Controller > Add Controller > One Click Connect.



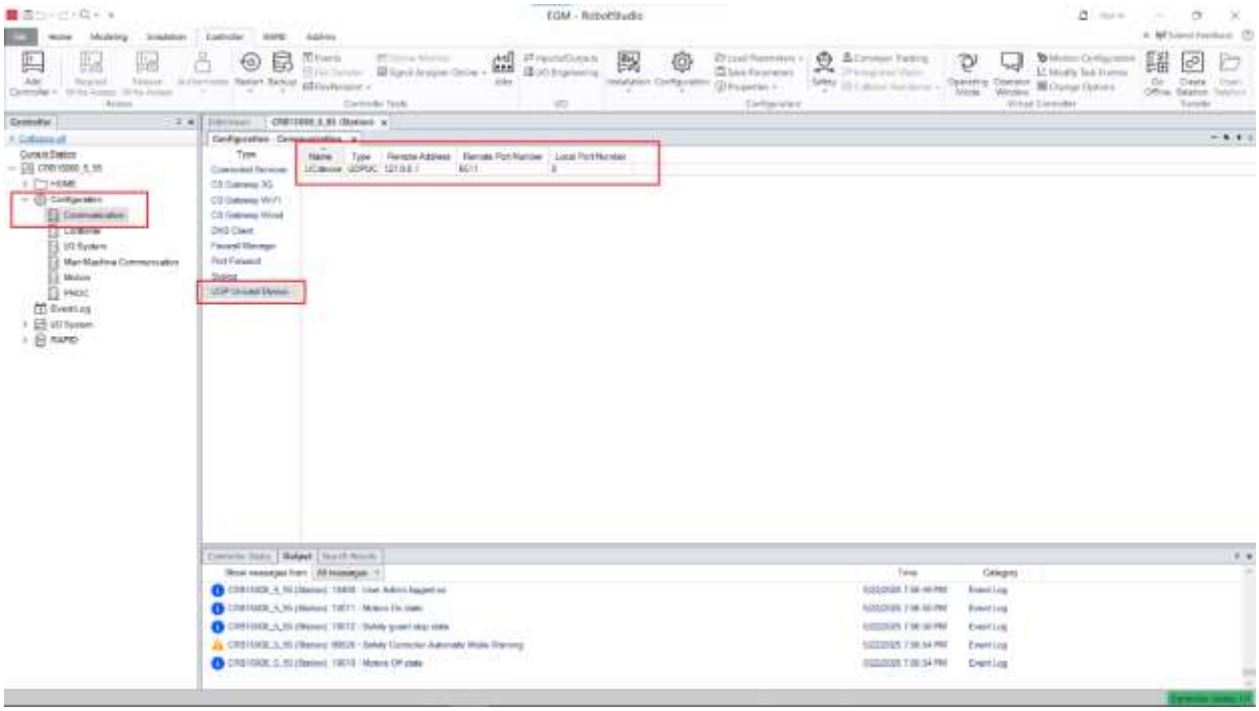
Hình 3 3 Kết nối bộ điều khiển Robot

Bước 2: Vì tùy chọn EGM không được kích hoạt mặc định trong bộ điều khiển nên ta sẽ phải kích hoạt trong tùy chọn bộ điều khiển. Sau khi Bộ điều khiển được kết nối, nhấp chuột phải vào Controller và chọn Change Options. Ở tab Options, chọn Engineering Tools và tích chọn EGM. Sau đó Apply và Reset bộ điều khiển.



Hình 3 4 Kích hoạt tùy chọn EGM trên bộ điều khiển Robot

Bước 3: Thiết lập cổng và địa chỉ cho Thiết bị UDP. Ở Tab Controller, chọn Configuration > Communication > UDP Unicast Device để thiết lập địa chỉ IP và Cổng. Sau khi thiết lập hoàn tất, tiến hành Reset bộ điều khiển.



Hình 3 5 Thiết lập cấu hình cho thiết bị UDP

3.2.2. Lập trình Rapid cho Robot

3.2.2.1. Các hàm EGM chính

a) EGMActPose

Mục đích:

EGMActPose kích hoạt một tiến trình EGM cụ thể điều khiển Robot đến một mục tiêu tư thế. Đây là bước chuẩn bị trước khi gọi hàm EGMRunPose.

Cú pháp cơ bản:

EGMActPose EGMid [\Tool] [\WObj] [\TLoad] [\DOtoSensor] [\DataToSensor] [\DIfromSensor] [\DataFromSensor] CorrFrame CorrFrType SensorFrame SensorFrType [\x] [\y] [\z] [\rx] [\ry] [\rz] [\LpFilter] [\SampleRate] [\MaxPosDeviation] [\MaxSpeedDeviation]

Các tham số chính

+ EGMid: Biến định danh cho kết nối EGM. Kiểu dữ liệu egmident.

+ \Tool: Công cụ được sử dụng cho các chuyển động được thực hiện với hướng dẫn EGMRunPose. Mặc định là tool0.

+ \WObj: WorkObject - hệ tọa độ làm việc. Nếu không khai báo, sử dụng hệ tọa độ toàn cục (world).

+ \TLoad: Tải được sử dụng cho các chuyển động được thực hiện với lệnh EGMRunPose. Nếu không khai báo, lấy từ tooldata.

+ \DotoSensor: Tín hiệu I/O được gửi từ bộ điều khiển Robot đến thiết bị bên ngoài.

+ \DataToSensor: Một mảng có tối đa 40 giá trị dnum được gửi từ bộ điều khiển Robot đến giao diện bên ngoài chẳng hạn như cảm biến.

+ \DifromSensor: Tín hiệu I/O được gửi từ giao diện bên ngoài như cảm biến đến bộ điều khiển Robot.

+ \DataFromSensor: Một mảng có tối đa 40 giá trị dnum được gửi từ giao diện bên ngoài chẳng hạn như cảm biến đến bộ điều khiển Robot.

+ CorrFrame: Khung hiệu chỉnh.

+ CorrFrType: Loại khung của khung hiệu chỉnh.

+ SensorFrame: Khung cảm biến.

+ SensorFrType: Loại khung của khung cảm biến.

+ \x, \y, \z: Tiêu chí hội tụ cho x, y và z tính bằng milimét. Giá trị mặc định là ±1,0 mm. Dữ liệu tiêu chí hội tụ được sử dụng để quyết định xem Robot đã đạt đến vị trí đã ra lệnh theo hướng trục đã chỉ định hay chưa. Nếu chênh lệch giữa vị trí đã ra lệnh và vị trí thực tế nằm trong khoảng egm_minmax.min và egm_minmax.max, thì Robot được coi là đã đến vị trí đã ra lệnh.

+ \rx, \ry, \rz: Tiêu chí hội tụ để xoay x, y và z theo độ. Giá trị mặc định là $\pm 0.5°$. Dữ liệu về tiêu chí hội tụ được sử dụng để quyết định xem Robot đã đạt đến hướng đã ra lệnh dọc theo trục đã chỉ định hay chưa. Nếu sự khác biệt giữa hướng theo thứ tự và hướng thực tế nằm trong phạm vi egm_minmax.min và egm_minmax.max, thì Robot được coi là đã đạt đến hướng đã ra lệnh.

+ \LpFilter: Bộ lọc thông thấp (Hz) giúp giảm nhiễu từ cảm biến. Mặc định lấy từ EGMSetup.

+ \SampleRate: Tần suất lấy mẫu dữ liệu từ cảm biến. Đơn vị là bội số của 4ms.

+ \MaxPosDeviation: Giới hạn lệch vị trí tối đa (°).

+ \MaxSpeedDeviation: Giới hạn thay đổi vận tốc tối đa (°/s) trên các khớp.

b) EGMRunPose

Mục đích:

EGMRunPose thực hiện một chuyển động có hướng dẫn từ cảm biến đến một vị trí đích bắt đầu từ một điểm chính xác cho một tiến trình EGM cụ thể, xác định các hướng và tư thế nào có thể được thay đổi.

Cú pháp cơ bản:

EGMRunPose EGMid, Mode [\NoWaitCond] [\x] [\y] [\z] [\rx] [\ry] [\rz] [\CondTime] [\RampInTime] [\RampOutTime] [\Offset] [\PosCorrGain]

Các tham số chính

+ EGMid: Biến định danh cho kết nối EGM. Kiểu dữ liệu egmident.

+ Mode: Kiểu dữ liệu egmstopmode. Xác định cách kết thúc chuyển động.

+ \NoWaitCond: Cho phép RAPID tiếp tục mà không chờ đến khi chuyển động hoàn tất. Phải gọi EGMWaitCond sau đó để đồng bộ.

+ \x, \y, \z: Chuyển động theo hướng x, y và z.

+ \rx, \ry, \rz: Định hướng lại xung quanh các trục x, y và z.

+ \CondTime: Cho phép RAPID tiếp tục mà không chờ đến khi chuyển động hoàn tất. Phải gọi EGMWaitCond sau đó để đồng bộ. Giá trị mặc định là 1 giây.

+ \RampInTime: Xác định tốc độ bắt đầu của chuyển động trong vài giây.

+ \RampOutTime: Xác định trong vài giây tốc độ giảm tốc của EGM sẽ được thực hiện.

+ Offset: Một pose tĩnh để cộng thêm vào dữ liệu cảm biến.

+ \PosCorrGain: Hệ số hiệu chỉnh vị trí. Giá trị từ 0 đến 1. Mặc định là 1.

c) EGMGetId

Mục đích:

EGMGetId được sử dụng để đặt trước một định danh EGM (EGMid). Định danh này sau đó sẽ được sử dụng trong tất cả các lệnh và hàm RAPID EGM khác để xác định một

tiến trình EGM cụ thể đang được kết nối với tác vụ chuyển động RAPID mà từ đó nó được sử dụng. Một egmident được nhận diện bằng tên của nó, nghĩa là khi gọi EGMGetId lần thứ hai hoặc thứ ba với cùng một egmident, sẽ không tạo ra một tiến trình EGM mới cũng như không thay đổi nội dung của nó.

Để giải phóng một egmident nhằm cho phép các tiến trình EGM khác sử dụng, cần sử dụng lệnh RAPID EGMReset.

Cú pháp cơ bản:

EGMGetId EGMid

Các tham số chính

+ EGMid: Biến định danh cho kết nối EGM. Kiểu dữ liệu egmident.

d) EGMSetupUC

Mục đích:

EGMSetupUC được sử dụng để thiết lập giao thức UdpUc cho một tiến trình EGM cụ thể (EGMid) làm nguồn cung cấp giá trị vị trí đích, mà robot và tối đa 6 trục phụ sẽ được dẫn hướng đến.

Cú pháp cơ bản:

EGMSetupUC MecUnit, EGMid, ExtConfigName, UCDevice [\Joint] | [\Pose] | [\PathCorr] [\APTR | \LATR] [\CommTimeout>]

Các tham số chính

+ MecUnit: Tên đơn vị cơ khí.
+ EGMid: Biến định danh cho kết nối EGM. Kiểu dữ liệu egmident.
+ ExtConfigName: Tên cấu hình giao tiếp định nghĩa trong system parameters.
+ UCDevice: Tên thiết bị UdpUc.
+ \Joint: Chọn chuyển động khớp để hướng dẫn vị trí.
+ \Pose: Chọn chuyển động tư thế để hướng dẫn vị trí.
+ \PathCorr: Chọn hiệu chỉnh đường dẫn.
+ \APTR: Thiết lập loại cảm biến At-point tracker để hiệu chỉnh đường dẫn.
+ \LATR: Thiết lập loại cảm biến Look-ahead tracker để hiệu chỉnh đường dẫn.

+ \CommTimeout: Thời gian timeout (giây) nếu không nhận được dữ liệu UDP.

e) EGMGetState

Mục đích:

EGMGetState trong RAPID cho phép bạn kiểm tra trạng thái hiện tại của một quá trình EGM (được xác định bằng egmident). Thuộc kiểu dữ liệu egmstate.

Cú pháp cơ bản:

EGMGetState (EGMid)

Các tham số chính

+ EGMid: Biến định danh cho kết nối EGM. Kiểu dữ liệu egmident.

3.2.2.2. Các kiểu dữ liệu EGM đặc trưng

a) Egmstate

Cách sử dụng:

Egmstate được sử dụng để xác định trạng thái của các hiệu chỉnh và đo đạc cảm biến trong EGM.

Mô tả:

Egmstate là giá trị trả về của hàm EGMGetState.

Các giá trị:

Bảng 3 2 Các giá trị trạng thái của một chu trình EGM

Giá trị	Mô tả
EGM_STATE_DISCONNECTED	Trạng thái EGM của tiến trình chưa được xác định. Chưa có thiết lập nào.
EGM_STATE_CONNECTED	Tiến trình EGM đã được thiết lập nhưng chưa hoạt động.
EGM_STATE_RUNNING	Tiến trình EGM đang hoạt động. Robot đang được điều khiển.

b) Egmident

Cách sử dụng:

Egmident dùng để xác định một tiến trình EGM cụ thể.

Mô tả:

Một egmident được đặt trước bằng lệnh EGMGetId. Sau đó, nó được dùng để liên kết và xác định chuỗi các lệnh như: EGMSetupXX, EGMActX, EGMRunX, và EGMReset cùng thuộc về một tiến trình EGM duy nhất.

Một egmident được xác định bằng tên biến của nó. Việc gọi EGMGetId nhiều lần với cùng một tên egmident sẽ không tạo tiến trình mới và không thay đổi nội dung của tiến trình hiện tại.

Chỉ có lệnh EGMReset mới có thể giải phóng một egmident để tiến trình khác có thể sử dụng lại.

c) Egmmin_max

Cách sử dụng:

Egm_minmax được dùng để xác định tiêu chí hội tụ (convergence criteria) cho việc dừng quá trình EGM. Nó giúp robot biết khi nào sai số vị trí nằm trong giới hạn cho phép, và có thể coi là đã đạt đến mục tiêu.

Mô tả:

Egm_minmax thường được sử dụng trong các lệnh như EGMActJoint và EGMActPose.

Thành phần:

Bảng 3 3 Các giá trị thành phần của Egm_minmax

Thành phần	Kiểu dữ liệu	Mô tả
Min	num	Sai số nhỏ nhất được cho phép.
Max	num	Sai số lớn nhất được cho phép.

d) Egmstopmode

Cách sử dụng:

Egmstopmode được dùng để xác định chế độ dừng cho các lệnh điều khiển hiệu chỉnh và dữ liệu đo từ cảm biến trong EGM.

Mô tả:

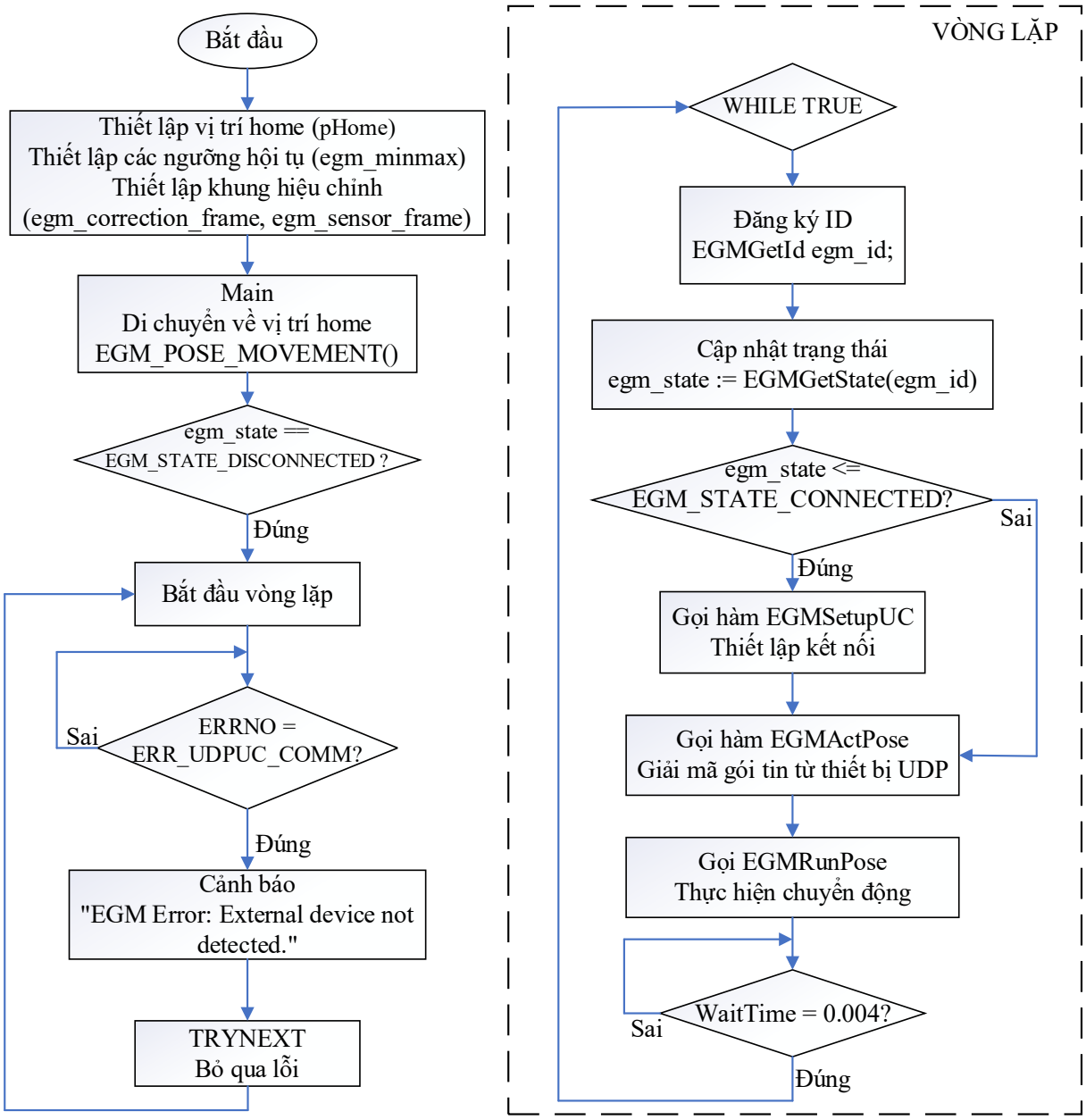
Egmstopmode thường được sử dụng trong các lệnh như EGMRunJoint, EGMRunPose và EGMStop.

Thành phần:

Bảng 3 4 Các giá trị được xác định của kiểu dữ liệu EGMstopmode

Giá trị	Mô tả
EGM_STOP_HOLD	Giữ nguyên vị trí cuối cùng khi dừng EGM. Robot không quay về vị trí ban đầu.
EGM_STOP_RAMP_DOWN	Trở về vị trí bắt đầu của EGM một cách từ từ.

3.2.2.3. Lưu đồ thuật toán



Hình 3 6 Lưu đồ thuật toán chương trình Rapid

3.3. Lập trình giao tiếp giữa PC và Robot

3.3.1. Lập trình giao thức EGM

3.3.1.1. Thiết lập ban đầu

Để thực hiện giao tiếp bằng giao thức EGM, ABB đã cung cấp một tệp có tệp "egm.proto". Đây là một tệp định nghĩa giao thức, dùng để định nghĩa cấu trúc của các message truyền qua UDP giữa Robot và PC. Để sử dụng, cần biên dịch bằng trình biên dịch Protocol Buffers, tạo ra mã nguồn python tương ứng.

Các cấu trúc message chính được sử dụng trong hệ thống

Bảng 3 5 Các tham số message EGM chính được sử dụng

Tên message	Chức năng
EgmRobot	Dữ liệu phản hồi gửi từ robot đến máy tính.
EgmSensor	Dữ liệu điều khiển gửi từ máy tính đến robot.
EgmHeader	Thông tin tiêu đề gói tin (số thứ tự, loại message, thời gian).
EgmPlanned	Vị trí dự kiến (Có thể là joint hoặc cartrsian).
EgmCartesian	Tọa độ X, Y, Z.
EgmEuler	Góc quay Euler: Rx, Ry, Rz.
EgmPose	Bao gồm EgmCartesian, EgmEuler.
EgmFeedBack	Vị trí thực tế phản hồi từ robot.
EgmMCIState	Trạng thái chu trình EGM.

3.3.1.2. Chức năng chính

Hệ thống sử dụng ngôn ngữ Python, thông qua giao thức EGM sẽ thực hiện hai chức năng chính:

+ Nhận phản hồi các giá trị vị trí/góc quay/trạng thái được gửi về từ Robot.
+ Gửi lệnh điều khiển vị trí/góc quay đến Robot với tần số 250Hz

3.3.1.3. Các lớp chính trong chương trình EGM

a) Lớp Tạo gói tin (CreateSensorMessage)

Chức năng: Tạo gói tin dạng EGMSensor để gửi đến Robot.

Các tham số chính:

+ Seq_num: Số thứ tự của gói tin

+ Pos: Vị trí mục tiêu (x, y, z)

+ Euler: Góc quay (rx, ry, rz)

b) Lớp Nhận gói tin từ Robot (ReceiveThread)

Chức năng: Nhận gói tin từ Robot sau đó xử lý dữ liệu phản hồi như vị trí, góc quay và trạng thái.

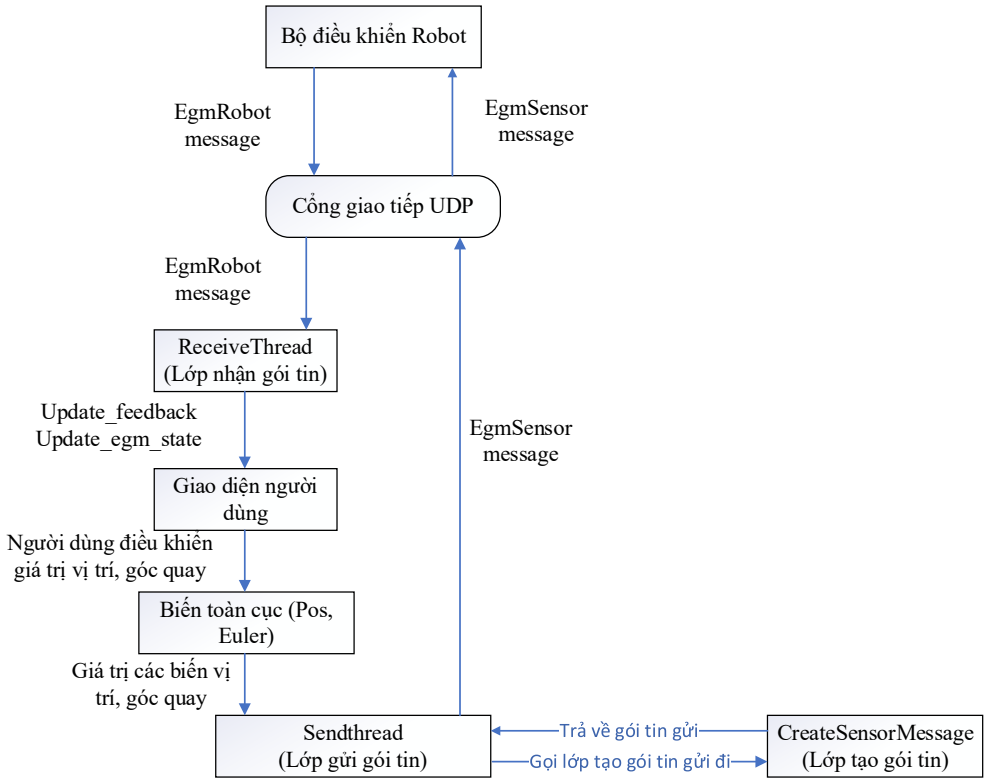
Các thành phần chính:

+ Update_feedback: Bao gồm các thông số vị trí (x, y, z) và góc quay (rx, ry, rz) thực tế được phản hồi từ Robot.

+ Update_egm_state: Chứa thông số trạng thái chu trình EGM.

c) Lớp Gửi gói tin đến Robot (SendThread)

Chức năng: Gửi gói tin chứa các thông số điều khiển tới Robot.



Hình 3 7 Sơ đồ luồng dữ liệu EGM

3.3.2. Lập trình RWS

3.3.2.1. Kiến trúc tổng quan

Mục đích chính của RWS là thông qua các phương thức GET/POST của giao thức HTTPS để nhận, gửi các lệnh điều khiển các tài nguyên của robot.

Định dạng dữ liệu: Trả về XML hoặc JSON (điều chỉnh qua header Accept)**.** Gửi dữ liệu, ví dụ thay đổi biến RAPID hay I/O, thường dùng XML trong phần body.

Các nhóm dịch vụ, tài nguyên chính được sử dụng trong hệ thống gồm:

+ /users: Đăng ký quyền người dùng.

+ /subscription: Đăng ký các thay đổi tài nguyên.

+ /ctrl/identity: Lấy thông tin của bộ điều khiển.

+ /rw/rapid: Quản lý chương trình RAPID.

+ /rw/panel: Truy xuất trạng thái giao diện điều khiển.

+ /rw/iosystem: Giám sát và điều khiển các tín hiệu I/O.

Định dạng dữ liệu: Trả về XML hoặc JSON (điều chỉnh qua header Accept)**.** Gửi dữ liệu, ví dụ thay đổi biến RAPID hay I/O, thường dùng XML trong phần body.

Cấu trúc cơ bản của một lệnh RWS sẽ bao gồm:

+ URL: Base URL + Endpoint cụ thể

+ Phương thức HTTP: Tùy vào mục đích của lệnh. Có thể dùng GET để đọc giá trị từ robot, POST để ghi giá trị đến robot.

+ Headers: Xác định cách mã hóa dữ liệu gửi và nhận, cùng thông tin xác thực. Bao gồm Content-Type, Accept.

+ Authorization: Chứa thông tin xác thực. Gồm Username và Password.

+ Data: Thường có khi sử dụng phương thức POST. Dữ liệu gửi đi dưới dạng form urlencoded.

3.3.2.2. Các chức năng chính được sử dụng trong hệ thống

a) Điều khiển và giám sát trạng thái bộ điều khiển

Bảng 3 6 Các hàm chức năng điều khiển và giám sát trạng thái bộ điều khiển

Hàm	Chức năng	Phương thức HTTP	API Endpoint
set_ctrl_state(state)	Thiết lập trạng thái của bộ điều khiển (motoron/motoroff).	POST	/rw/panel/ctrl-state
get_ctrl_state()	Cập nhập trạng thái bộ của điều khiển.	GET	/rw/panel/ctrl-state

b) Điều khiển và giám sát chế độ vận hành

Bảng 3 7 Các hàm chức năng điều khiển và giám sát chế độ vận hành

Hàm	Chức năng	Phương thức HTTP	API Endpoint
set_opmode(mode)	Thiết lập chế độ vận hành (Auto/Man). Yêu cầu quyền người dùng cục bộ.	POST	/rw/panel/opmode
opmode_ack()	Xác nhận chuyển chế độ (Bắt buộc khi sang chế độ auto)	POST	/rw/panel/opmode /acknowledge
get_opmode()	Cập nhập trạng thái vận hành (Auto/Man)	GET	/rw/panel/opmode

c) Điều khiển và giám tốc độ robot

Bảng 3 8 Các hàm chức năng điều khiển và giám sát tốc độ robot

Hàm	Chức năng	Phương thức HTTP	API Endpoint
set_speed(ratio)	Đặt tỷ lệ tốc độ robot (0-100%).	POST	/rw/panel/speedratio?mastership =implicit
get_speed()	Cập nhập tốc độ của robot.	GET	/rw/panel/speedratio

d) Điều khiển và giám sát trạng thực thi chương trình Rapid

Bảng 3 9 Các hàm chức năng điều khiển và giám sát trạng thái thực thi chương trình RAPID

Hàm	Chức năng	Phương thức HTTP	API Endpoint
start_rapid()	Bắt đầu thực thi chương trình RAPID.	POST	/rw/rapid/execution/start
stop_rapid ()	Dừng thực thi chương trình RAPID.	POST	/rw/rapid/execution/stop ?mastership=implicit

reset_rapid()	Reset chương trình Rapid.	POST	/rw/rapid/execution/resetpp ?mastership=implicit
get_rapid_state()	Cập nhập trạng thái thực thi chương trình RAPID (Start/Stop).	GET	/rw/rapid/execution

e) Điều khiển và giám sát trạng thái I/O

Bảng 3 10 Các hàm chức năng điều khiển và giám sát trạng thái I/O

Hàm	Chức năng	Phương thức HTTP	API Endpoint
set_IO(lvalue)	Thiết lập trạng thái I/O (ON/OFF gripper)	POST	/rw/iosystem/signals/EtherNetIP /ABB_Scalable_IO1 /ABB_Scalable_IO1_0_DO2 /set-value
get_IO_state()	Cập nhập trạng thái I/O hiện tại.	GET	/rw/iosystem/signals/EtherNetIP /ABB_Scalable_IO1 /ABB_Scalable_IO1_0_DO2

f) Đăng ký quyền người dùng cục bộ

Bảng 3 11 Hàm đăng ký người dùng cục bộ

Hàm	Chức năng	Phương thức HTTP	API Endpoint
register_user_local()	Đăng ký người dùng cục bộ, dùng để điều khiển chế độ vận hành.	POST	/users/register/local

g) Nhận thông tin của bộ điều khiển

Bảng 3 12 Hàm chức năng nhận thông tin của bộ điều khiển

Hàm	Chức năng	Phương thức HTTP	API Endpoint

get_identity()	Cập nhập thông tin của bộ điều khiển (tên, loại thiết bị)	POST	/ctrl/identity

h) Đăng ký tài nguyên

Bảng 3 13 Hàm chức năng đăng ký các thay đổi tài nguyên

Hàm	Chức năng	Phương thức HTTP	API Endpoint
subscribe(parent)	Đăng ký các thay đổi trên các tài nguyên khác nhau. Cập nhập thay đổi liên tục thông qua Websocket.	POST	/subscription

3.3.3. Thiết kế giao diện

3.3.3.1. Giới thiệu về thư viện PyQt5

Qt Desiger là một bộ framework phát triển phần mềm giao diện người dùng (GUI – Graphical User Interface) đa nền tảng, mã nguồn mở, được phát triển ban đầu bởi công ty Trolltech và hiện được duy trì bởi Qt Desiger Company. Qt Desiger hỗ trợ phát triển ứng dụng trên nhiều hệ điều hành như Windows, Linux, macOS, Android, và iOS, cho phép các lập trình viên xây dựng ứng dụng có giao diện nhất quán và giàu tính năng trên nhiều thiết bị.

Qt5 Designer là phiên bản thứ năm của framework này, được phát hành chính thức từ năm 2012. Với nhiều cải tiến về hiệu năng và khả năng mở rộng, Qt5 Desiger hỗ trợ không chỉ phát triển phần mềm desktop mà còn cả các hệ thống nhúng, thiết bị IoT và giao diện công nghiệp thông minh.

a) PyQt5 – Bộ công cụ giao diện Python dựa trên Qt5

PyQt5 là một thư viện wrapper của Python cho Qt5, do Riverbank Computing phát triển. Thư viện này cung cấp hơn 620 lớp (class) bao trùm toàn bộ tính năng của Qt5 Desiger, từ các thành phần giao diện cơ bản như nút bấm, menu, hộp thoại... đến các hệ thống phức tạp như xử lý sự kiện, đồ họa 2D/3D, hiển thị ảnh/video, đa luồng và truyền thông mạng (networking). PyQt5 cho phép lập trình viên viết ứng dụng GUI chuyên nghiệp bằng ngôn ngữ Python – ngôn ngữ được ưa chuộng trong nghiên cứu AI, xử lý ảnh và điều khiển robot – mà vẫn tận dụng được sức mạnh của Qt Desiger.

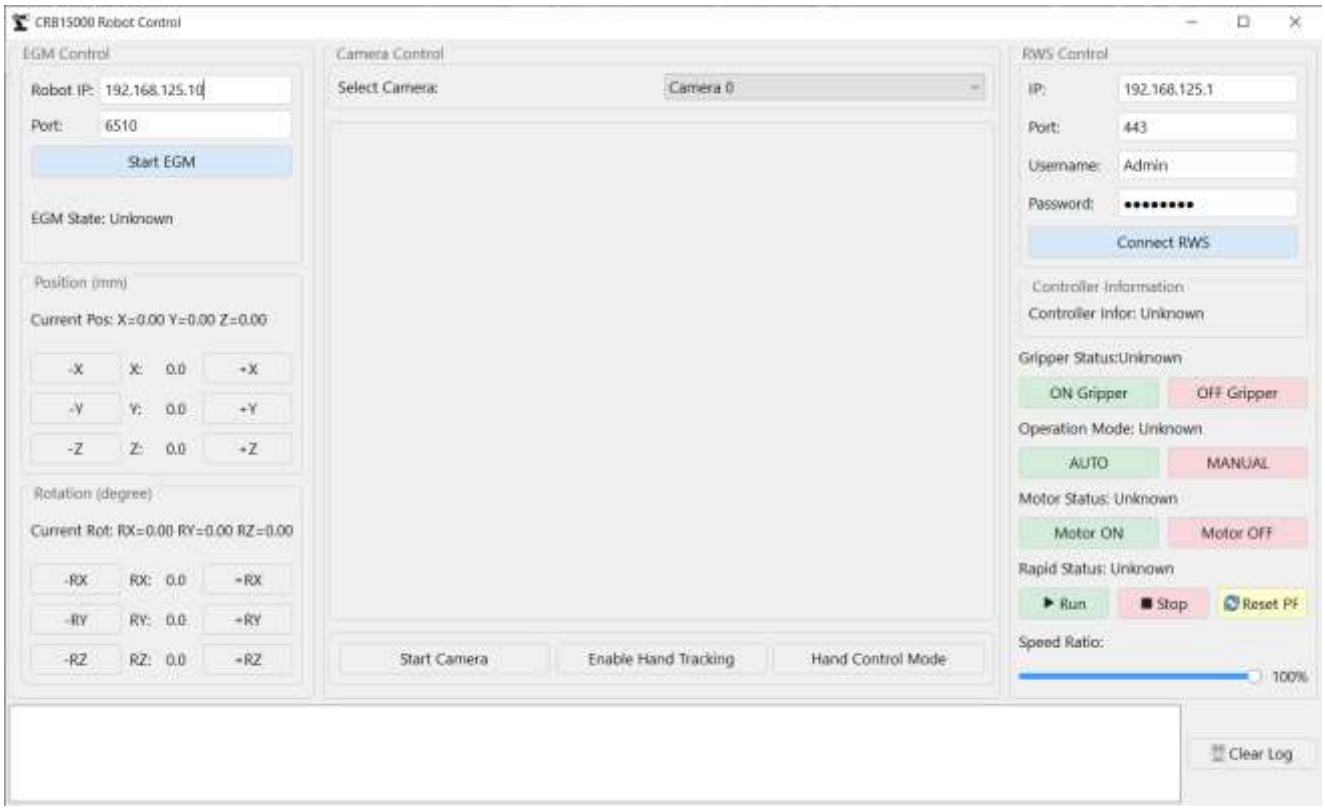
b) Các tính năng nổi bật của PyQt5

Bảng 3 14 Các chức năng nổi bật của thư viện PyQt5

Chức năng	Mô tả chi tiết
Thiết kế giao diện (Widgets)	Cung cấp hơn 50 widget giao diện như QPushButton, QLabel, QLineEdit, QComboBox, QTableView...
Trình quản lý layout	Hỗ trợ layout tự động theo chiều ngang, dọc, lưới, cho phép bố trí giao diện dễ dàng, thích nghi đa độ phân giải.
Signal-Slot	Cơ chế xử lý sự kiện mạnh mẽ, cho phép kết nối hành vi người dùng với logic xử lý một cách rõ ràng, tách biệt.
Đa luồng (Multithreading)	Hỗ trợ các tác vụ chạy song song như xử lý hình ảnh, thu dữ liệu camera, điều khiển robot... mà không làm treo giao diện.
Xử lý ảnh và video	Hỗ trợ hiển thị ảnh từ OpenCV (QImage, QPixmap), kết hợp với QTimer và QThread để cập nhật khung hình theo thời gian thực.
Tích hợp với Qt Designer	Giao diện kéo-thả GUI trực quan, tiết kiệm thời gian viết mã. Các file .ui được biên dịch thành mã Python dễ dàng.
Kết nối mạng	Có thể tích hợp HTTP, socket, WebSocket… phục vụ cho REST API hoặc robot giao tiếp thời gian thực.

3.3.3.2. Cấu trúc của giao diện

Giao diện điều khiển Robot ABB CRB15000 được chia làm 4 nhóm chính: EGM Control, Camera Control, RWS Control và System Log



Hình 3 8 Giao diện điều khiển Robot CRB15000

a) EGM Control

Phần điều khiển EGM nằm bên trái của giao diện, bao gồm các thành phần chính:

Bảng 3 15 Các thành phần chính của nhóm EGM Control trên giao diện

Thành phần	Chức năng
Các ô text	Nhập địa chỉ IP/Port để giao tiếp UDP.
Nút nhấn "Connect EGM"	Tạo kết nối EGM đến robot để bắt đầu gửi/nhận vị trí/góc quay.
EGM State	Hiển thị trạng thái của quy trình EGM.
Các nút nhấn tăng/giảm X, Y, Z	Điều khiển các giá trị vị trí của Robot.
Các label hiển thị giá trị X, Y, Z	Hiển thị giá trị vị trí được gửi đi và giá trị vị trí thực tế của robot phản hồi.
Các nút nhấn tăng/giảm RX, RY, RZ	Điều khiển giá trị góc quay của Robot
Các label hiển thị giá trị RX, RY, RZ	Hiển thị giá trị góc quay được gửi đi và giá trị góc quay thực tế của robot phản hồi.

b) Camera Control

Phần Camera nằm ở chính giữa giao diện, bao gồm các thành phần chính:

Bảng 3 16 Các thành phần chính của nhóm Camera Control trên giao diện

Thành phần	Chức năng
Ô lựa chọn camera	Lựa chọn thiết bị camera để bắt đầu xử lý ảnh.
Nút nhấn "Start Camera"	Bắt đầu kết nối đến camera.
Nút nhấn "Enable Hand Tracking"	Cho phép bật chế độ phát hiện bàn tay.
Nút nhấn "Hand Control Mode"	Bật chế độ xử lý cử chỉ tay.
Phần hiển thị camera	Hiển thị video trực tiếp từ camera. Phục vụ cho việc nhận dạng, xử lý cử chỉ tay cũng như các nút nhấn ảo để điều khiển robot.

c) RWS Control

Phần điều khiển RWS nằm bên phải của giao diện, bao gồm các thành phần:

Bảng 3 17 Các thành phần chính của nhóm RWS Control trên giao diện

Thành phần	Chức năng
Các ô nhập liệu	Nhập địa chỉ IP/Port/Username/Password để kết nối đến bộ điều khiển của robot.
Nút nhấn "Connect RWS"	Tạo kết nối RWS đến bộ điều khiển robot.
Controller Information	Hiển thị các thông tin của bộ điều khiển như Tên/Loại bộ điều khiển.
Các nút nhấn ON/OFF Gripper	Điều khiển đóng mở công cụ Gripper của robot.
Các nút nhấn AUTO/MANUAL	Điều khiển chế độ vận hành của bộ điều khiển.
Các nút nhấn Motor ON/Motor OFF	Điều khiển bật/tắt động cơ.
Các nút nhấn START RAPID/STOP RAPID/RESET	Điều khiển chạy/dừng/reset chương trình chính.

Các nhãn hiển thị thông tin	Hiển thị các trạng thái của Gripper, chế độ vận hành, trạng thái motor, trạng thái thực thi chương trình, tốc độ.
Thanh trượt	Điều chỉnh tốc độ của bộ điều khiển.

d) System Log

Nằm phía dưới cùng của giao diện, gồm 2 thành phần chính

Bảng 3 18 Các thành phần chính của nhóm System Log trên giao diện

Thành phần	Chức năng
Phần text log	Hiển thị các thay đổi của hệ thống.
Nút nhấn “Clear Log”	Xóa toàn bộ các thông tin có trong Log.

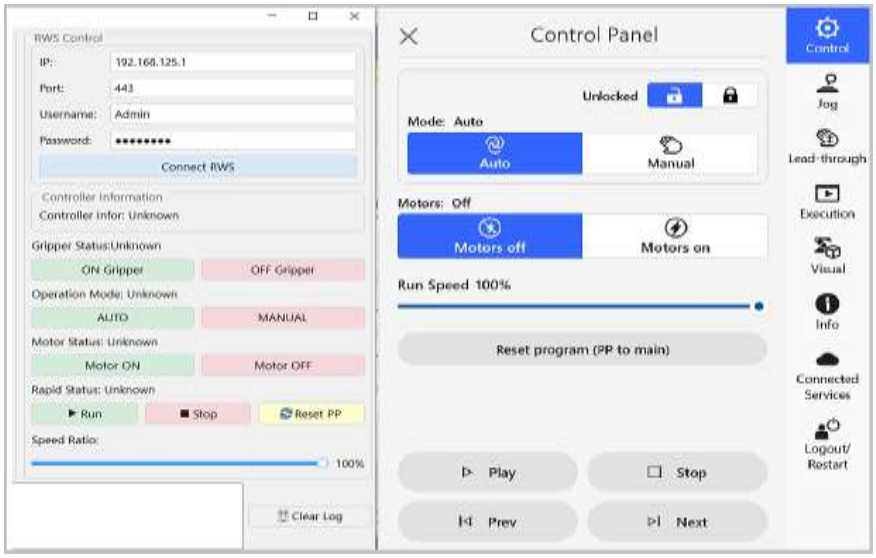
3.4. Kết luận chương 3

Trong chương này, chúng tôi đã trình bày tổng quan toàn bộ hệ thống tương tác giữa người và robot CRB15000. Bên cạnh đó, nội dung chương cũng đã đi sâu vào quá trình thiết kế hệ thống bằng ngôn ngữ Python với các giao thức EGM, RWS và xây dựng một giao diện trực quan giúp cho việc tương tác được hiệu quả. Sau khi hoàn thiện việc lập trình cho toàn bộ hệ thống, trong chương tiếp theo chúng tôi sẽ tiến hành kiểm nghiệm bằng mô phỏng và thực nghiệm trên robot thật, từ đó đánh giá hiệu quả hoạt động và mức độ đáp ứng của hệ thống.

Chương 4: KIỂM NGHIỆM VÀ ĐÁNH GIÁ KẾT QUẢ

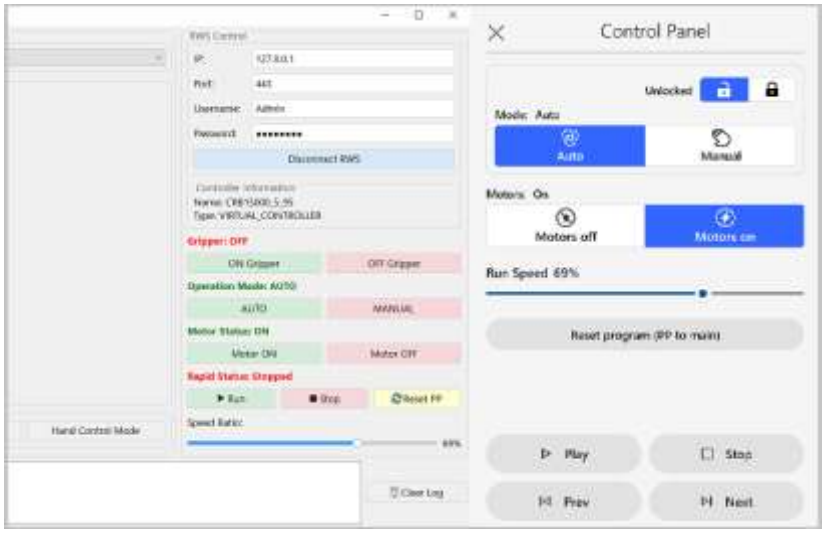
4.1. Mô phỏng trên RobotStudio

4.1.1. Kiểm tra chức năng RWS



Hình 4 1 Giao diện RWS Control khi chưa kết nối với robot

Đây là giao diện RWS Control khi chưa kết nối đến bộ điều khiển của robot. Lúc này, các thông tin như trạng thái của gripper, chế độ vận hành, trạng thái motor… vẫn chưa được cập nhập và chưa thể sử dụng các nút để điều khiển.

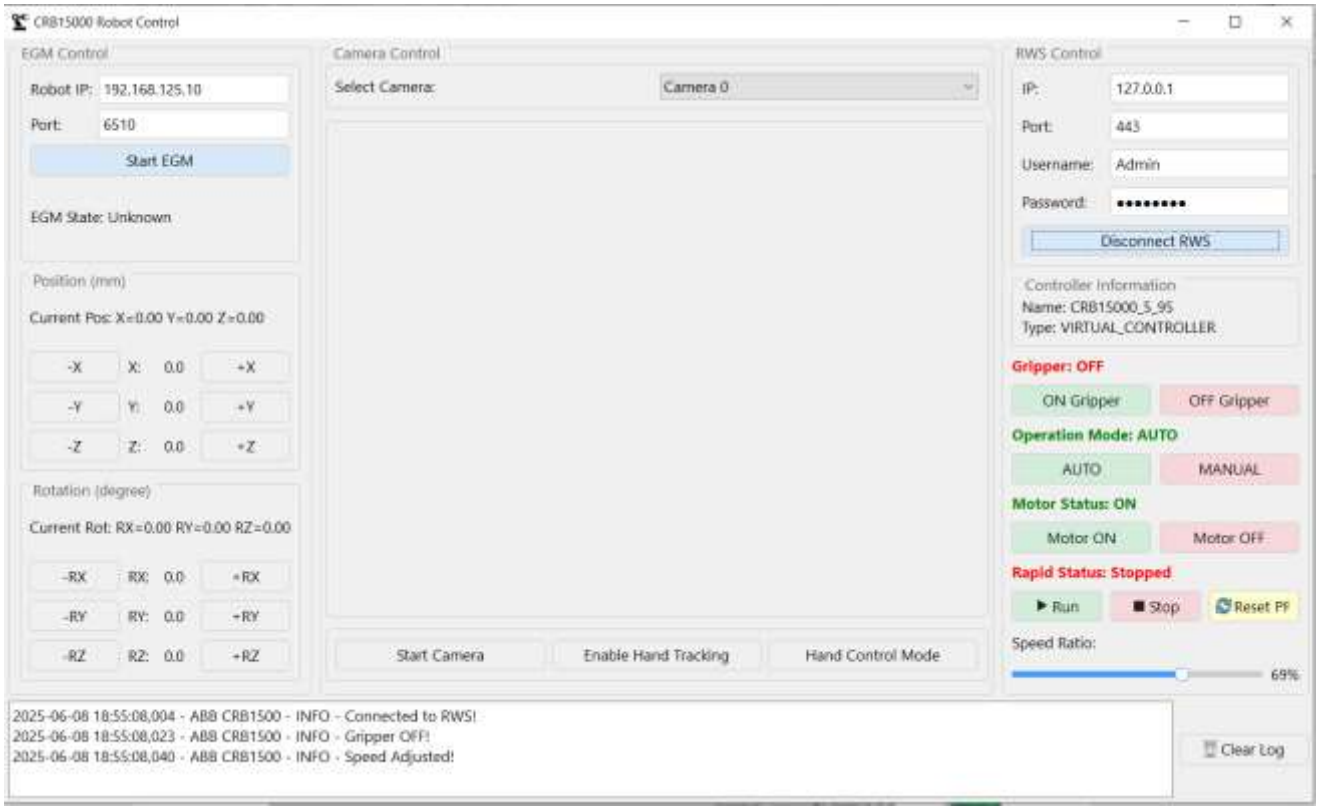


Hình 4 2 Giao diện RWS Control kết nối robot, điều khiển và hiển thị trạng thái thời gian thực

Sau khi kết nối RWS, có thể thấy Giao diện RWS Control đã cập nhập đầy đủ các trạng thái vận hành, trạng thái motor, trạng thái thực thi chương trình cũng như tốc độ. Đồng thời có thể sử dụng các nút nhấn, thanh trượt để điều khiển.

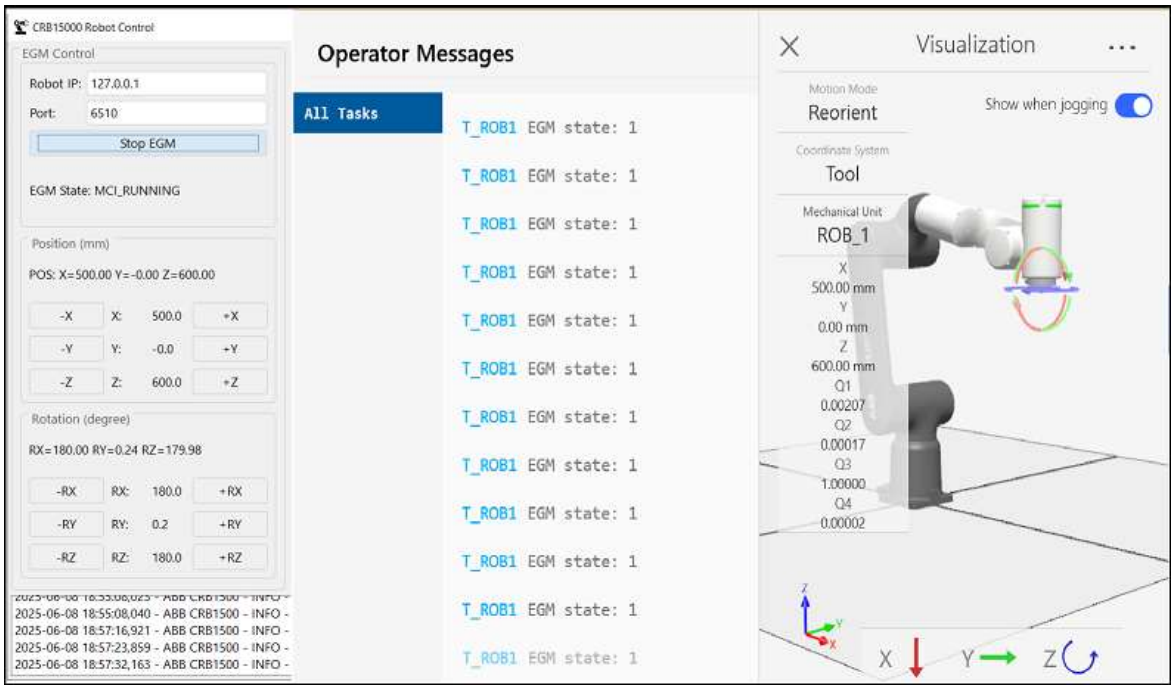
Kết quả đạt được: Giao diện đã có thể kết nối với robot, có thể điều khiển cũng như giám sát trạng thái của các đối tượng cơ bản trên bộ điều khiển robot với thời gian phản hồi nhanh chóng.

4.1.2. Kiểm tra chức năng điều khiển vị trí robot



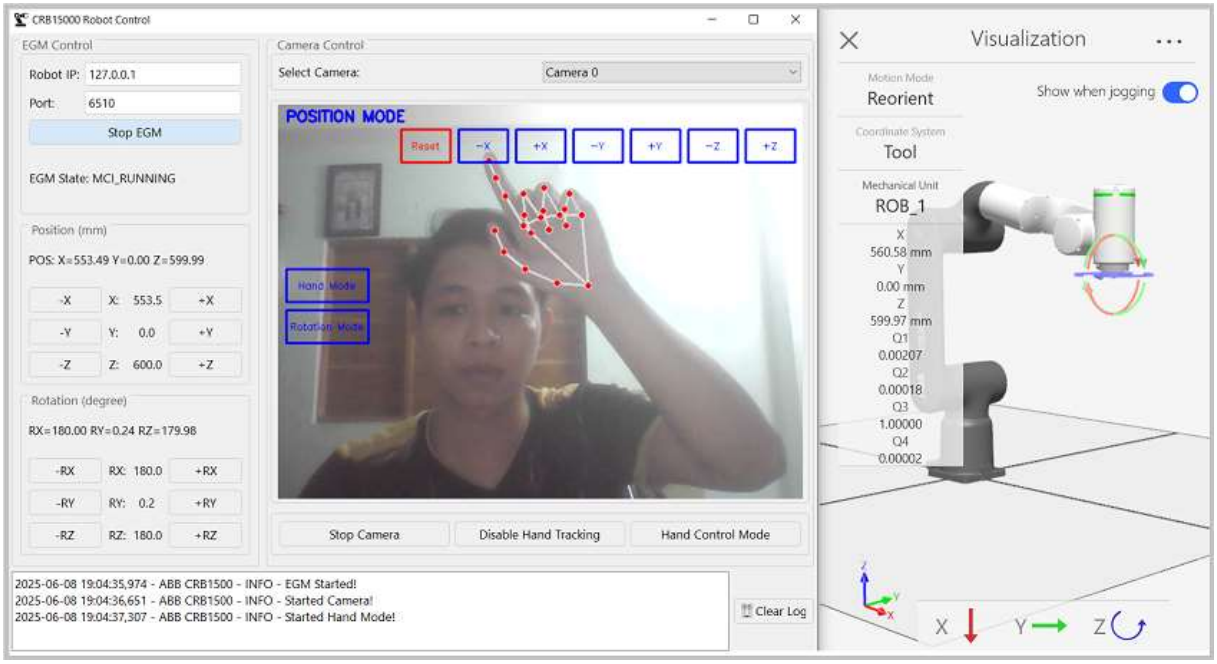
Hình 4 3 Giao diện người dùng khi chưa kết nối EGM với robot

Có thể thấy, khi chưa bắt đầu giao thức EGM thì trên giao diện vẫn chưa cập nhập được trạng thái hoạt động của chu trình EGM, chưa thể điều khiển cũng như giám sát được vị trí/góc quay của robot.



Hình 4 4 Giao diện EGM Control sau khi kết nối EGM với robot

Sau khi bắt đầu EGM, có thể thấy trên giao diện đã cập nhập được trạng thái của chu trình EGM, đồng thời có thể điều khiển và giám sát vị trí/góc quay của robot.



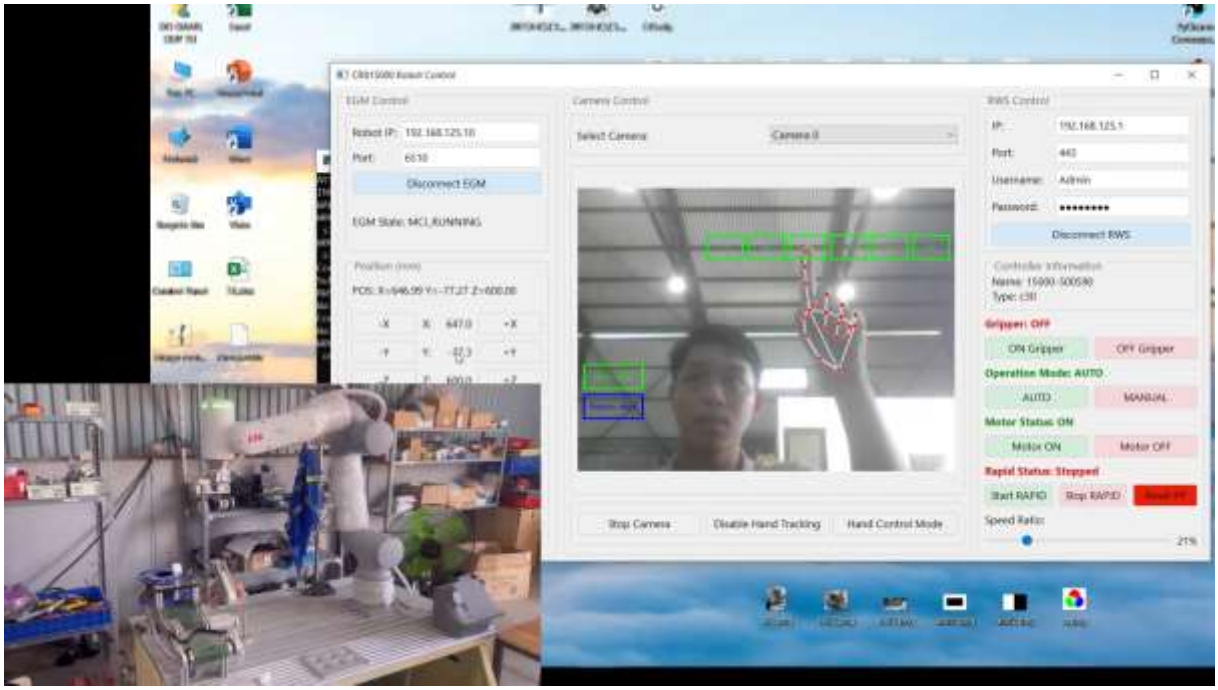
Hình 4 5 Điều khiển vị trí/góc quay của robot bằng xử lý ảnh

Bên cạnh việc điều khiển vị trí/góc quay của robot bằng các nút nhấn trên giao diện, người dùng cũng có thể điều khiển vị trí/góc quay của robot bằng các nút nhấn ảo thông qua xử lý ảnh, nhận diện cử chỉ tay.

4.2. Thực nghiệm trên Robot CRB15000

Hình 4 6 Điều khiển và giám sát trạng thái robot thông qua giao diện

Sau khi kết nối với bộ điều khiển của robot, giao diện đã cập nhập chính xác và nhanh chóng thông tin bộ điều khiển, trạng thái động cơ, chế độ vận hành và tốc độ của robot.



Hình 4 7 Điều khiển vị trí/góc quay của robot

Kiểm tra hoạt động điều khiển vị trí/góc quay của robot thông qua các nút nhấn trên giao diện và các nút nhấn ảo bằng xử lý ảnh. Hệ thống hoạt động trơn tru, chính xác.

Hình 4 8 Điều khiển đóng/mở gripper

Điều khiển đóng, mở gripper bằng cử chỉ nắm/mở bàn tay thông qua xử lý ảnh.

4.3. Đánh giá hệ thống

4.3.1. Độ trễ của hệ thống

Hệ thống tương tác người máy với robot ABB CRB15000 ứng dụng xử lý ảnh, nhận dạng cử chỉ tay và các nút nhấn ảo để điều khiển vị trí của robot thông qua giao thức EGM và quản lý trạng thái, chế độ robot bằng giao thức RWS. Trong quá trình thử nghiệm thực tế, độ trễ của hệ thống được đo đạc bằng cách xác định thời gian từ lúc người điều khiển bắt đầu chuyển động đến khi robot hoàn tất hành vi tương ứng. Độ trễ tổng thể của hệ thống bao gồm các thành phần chính:

+ Thời gian xử lý ảnh: Bao gồm thời gian thu hình từ camera, phát hiện bàn tay và xử lý để nhận diện cử chỉ tay. Tổng thời gian này thường giao động từ 80-120ms tùy thuộc vào camera, cấu hình máy, ánh sáng.

+ Thời gian gửi lệnh điều khiển thông qua giao thức EGM: Giao thức EGM thường có độ trễ điều khiển rơi vào khoảng 10-20ms tùy loại robot.

+ Thời gian phản hồi và di chuyển của robot: Robot thường mất khoảng 20ms để phản hồi vị trí mới.

Nhận xét: Độ trễ tổng thể của hệ thống 100-150 ms, đủ để đáp ứng yêu cầu điều khiển thời gian thực trong các ứng dụng tương tác cử chỉ với robot thông thường. Tuy nhiên để đạt hiệu quả cao nhất, cần tối ưu phần cứng xử lý ảnh (Camera, GPU…) hoặc sử dụng các thuật toán nhận dạng tối ưu hơn.

4.3.2. Độ chính xác

Hệ thống tương tác người máy với robot ABB CRB15000 ứng dụng xử lý ảnh, nhận dạng cử chỉ tay và các nút nhấn ảo để điều khiển vị trí của robot thông qua giao thức EGM và quản lý trạng thái, chế độ robot bằng giao thức RWS. Độ chính xác của hệ thống sẽ được đánh giá dựa trên hai thành phần chính

+ Độ chính xác về vị trí: Độ chính xác vị trí của robot sẽ được đánh giá dựa trên việc so sánh giá trị đặt và giá trị thực tế của robot. Kết quả thực nghiệm cho thấy vị trí thực tế của robot gần như chính xác tuyệt đối với vị trí đặt nhờ vào giao thức EGM với giao thức UDP tốc độ cao (250Hz).

+ Độ chính xác của quá trình nhận dạng, xử lý cử chỉ tay: Hệ thống hoạt động nhận dạng và xử lý cử chỉ tay tương đối chính xác trong điều kiện ổn định, tuy nhiên vẫn còn một số yếu tố ảnh hưởng đến chất lượng nhận diện như: ánh sáng, nhiễu, tốc độ cử chỉ của người dùng, góc quay bàn tay và chất lượng camera

Nhận xét: Nhìn chung, hệ thống đáp ứng tốt yêu cầu về độ chính xác phù hợp cho các ứng dụng điều khiển thời gian thực đơn giản. Tuy nhiên, để đạt độ tin cậy cao hơn trong môi trường công nghiệp thì cần cải thiện thêm một số yếu tố như bổ sung các thuật toán lọc nhiễu, tăng chất lượng camera…

4.4. Kết luận chương 4

Trong chương này, nhóm chúng tôi đã hoàn thành việc kiểm nghiệm bằng mô phỏng trên phần mềm RobotStudio và thực nghiệm trên hệ thống robot thật tại Công ty TNHH Tự động hóa Nhật Tri. Kết quả của quá trình kiểm nghiệm đã cho thấy hệ thống hoạt động ổn định, tốc độ cao và độ trễ thấp, ngoài ra giao thức EGM Position Guidance đã cho thấy độ chính xác tuyệt đối, đảm bảo robot đạt được vị trí mục tiêu như được chỉ định.

KẾT LUẬN

Đồ án "Phát triển hệ thống tương tác người – máy cho robot ABB CRB15000" tập trung vào nghiên cứu và triển khai giải pháp điều khiển và giám sát robot cộng tác CRB15000 thông qua máy tính, nhằm xây dựng một hệ thống giao tiếp người – máy trực quan, linh hoạt và hiệu quả hơn trong môi trường công nghiệp hiện đại.

Trong suốt quá trình thực hiện, nhóm đã hoàn thành các nội dung chính sau:

+ Về xử lý ảnh và nhận diện cử chỉ: Nhóm sử dụng thư viện MediaPipe kết hợp OpenCV để xây dựng hệ thống nhận diện cử chỉ tay theo thời gian thực. Hệ thống cho phép nhận diện bàn tay, phân tích 21 điểm đặc trưng (landmark) và điều khiển robot thực hiện các thao tác như đóng/mở gripper hay điều chỉnh vị trí/góc quay thông qua các nút nhấn ảo trên giao diện camera.

+ Về điều khiển robot: Áp dụng giao thức Externally Guided Motion (EGM) để điều khiển vị trí của robot theo thời gian thực, đồng thời tích hợp Robot Web Services (RWS) để giám sát và gửi lệnh vận hành robot thông qua giao diện.

+ Về giao diện người dùng: Giao diện điều khiển được thiết kế bằng ngôn ngữ Python và thư viện PyQt5, tích hợp đồng thời ba thành phần chính: EGM Control, RWS Control và Camera Control. Nhờ đó, người vận hành có thể điều khiển và giám sát trạng thái, vị trí của robot một cách trực quan, dễ hiểu.

+ Về thực nghiệm: Hệ thống đã được kiểm chứng bằng mô phỏng trên phần mềm RobotStudio, đồng thời triển khai thực nghiệm trên robot ABB CRB15000 thật tại Công ty TNHH Tự Động Hóa Nhật Tri. Các thử nghiệm cho thấy hệ thống hoạt động ổn định, có độ phản hồi tốt, đáp ứng yêu cầu điều khiển và giám sát thực tế.

Hiện tại hệ thống đã đáp ứng được nhu cầu điều khiển và giám sát robot thông qua giao diện tương tác được lập trình kết hợp với nhận dạng cử chỉ tay, tuy nhiên vẫn còn một số vấn đề tồn tại như chất lượng nhận dạng cử chỉ tay còn phụ thuộc vào ánh sáng môi trường xung quanh, chất lượng camera và thuật toán nhân dạng chưa tối ưu…Dựa vào những thiếu sót trên, nhóm đưa ra một số phương án cải thiện như: Tối ưu phần cứng xử lý ảnh (Camera, GPU…) hoặc sử dụng các thuật toán nhận dạng tối ưu hơn.

Bên cạnh đó, nhóm chúng tôi cũng có định hướng phát triển trong tương lai vì còn tồn tại các hạn chế của hệ thống. Một trong những bất tiện lớn của hệ thống hiện tại là

người dùng buộc phải đứng gần máy tính để thao tác điều khiển robot, gây hạn chế về tính linh hoạt và tính cơ động trong môi trường làm việc thực tế. Vì vậy, nhóm định hướng phát triển hệ thống theo hướng tích hợp với các thiết bị đeo thông minh như kính Google Glass. Với khả năng hiển thị thông tin trực tiếp trong tầm mắt và hỗ trợ nhập lệnh bằng giọng nói hoặc cử chỉ, kính thông minh sẽ giúp người vận hành có thể giám sát và điều khiển robot từ xa, hoàn toàn rảnh tay, đồng thời di chuyển linh hoạt trong không gian sản xuất mà không bị phụ thuộc vào vị trí của máy tính. Thêm vào đó việc tích hợp hệ thống vào mắt kính có thể giúp chúng tôi phát triển hệ thống theo những hướng khác như có thể biết được thông tin máy móc, tình trạng hoạt động chỉ bằng cách nhìn vào thiết bị đó và vô số tiện ích khác mà hệ thống có thể mang lại.

Cuối cùng, nhóm chúng tôi xin gửi lời cảm ơn chân thành đến TS. Giáp Quang Huy và cảm ơn công ty TNHH Tự Động Hoá Nhật Tri, đặc biệt là KS. Lê Phước Sinh đã đồng hành và tạo điều kiện tốt nhất về kiến thức cũng như cơ sở vật chất, thiết bị để nhóm có thể hoàn thành đồ án tốt nghiệp. Trong suốt quá trình làm việc, nhóm chúng tôi đã nhận được rất nhiều sự giúp đỡ, đóng góp ý kiến và sự hướng dẫn từ phía giảng viên hướng dẫn và phía công ty. Đây chính là nguồn động lực và nền tảng quan trọng giúp chúng tôi hoàn thiện đồ án một cách hiệu quả và đúng tiến độ.

TÀI LIỆU THAM KHẢO

[1] A. Robotics, "Application Manual – Externally Guided Motion," ABB, 2024.

[2] A. Robotics, "Robot Web Services API Documentation," [Online]. Available: https://developercenter.robotstudio.com/api/RWS.

[3] ABB, Product Manual – OmniCore C30, 2025.

[4] ABB, Operating manual RobotStudio, 2024.

[5] Y. Shastri, "What is Computer Vision?," 28 Aug 2024. [Online]. Available: https://zilliz.com/learn/what-is-computer-vision. [Accessed 28 May 2025].

[6] OpenCV, "About," 04 Nov 2020. [Online]. Available: https://opencv.org/about/. [Accessed 28 05 2025].

[7] J. Beachy, "GitHub - macs-lab/abb-egm-with-python: A script that allows for interfacing with ABB's Externally Guided Motion (EGM) option in python," 2023. [Online]. Available: https://github.com/macs-lab/abb-egm-with-python.

[8] F. Fronchetti, "GitHub - lsurobotics/egm-for-abb-robots: :books: Tutorial on ABB robot manipulation using Externally Guided Motion (EGM)," 2022. [Online]. Available: https://github.com/lsurobotics/egm-for-abb-robots.

[9] S. Jason Van, "Python and REST APIs: Interacting With Web Services," 28 July 2021. [Online]. Available: https://realpython.com/api-integration-in-python/#rest-architecture.

PHỤ LỤC

PHỤ LỤC 1: MÃ NGUỒN CHƯƠNG TRÌNH RAPID GIAO TIẾP EGM

```
MODULE EGMCommunication
    VAR egmident egm_id;
    VAR egmstate egm_state;
    PERS robtarget pHome:=[[500.0,0.0,600.0],[0.0,0.0,1.0,0.0],
                    [0,0,-0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    ! Ngưỡng hội tụ
    CONST egm_minmax egm_minmax_translation := [-1, 1];
    CONST egm_minmax egm_minmax_rotation := [-1, 1];
    ! Khung hiệu chỉnh và cảm biến
    LOCAL CONST pose egm_correction_frame := [[0, 0, 0], [1, 0, 0, 0]];
    LOCAL CONST pose egm_sensor_frame := [[0, 0, 0], [1, 0, 0, 0]];

    PROC main()
        MoveJ pHome,v100,fine,tool0;
        EGM_POSE_MOVEMENT;
    ENDPROC

    PROC EGM_POSE_MOVEMENT()
        !Kiểm tra có thiết lập EGM nào đang hoạt động không
        IF egm_state = EGM_STATE_DISCONNECTED THEN
            TPWrite "EGM State: Preparing controller for EGM communication.";
        ENDIF

        WHILE TRUE DO
        ! Đăng ký ID EGM mới
        EGMGetId egm_id;
        ! Nhận trạng thái hiện tại của chu trình EGM
        egm_state := EGMGetState(egm_id);
        !Thiết lập giao tiếp EGM
        IF egm_state <= EGM_STATE_CONNECTED THEN
            TPWrite "EGM State: Setting up communication.";
            EGMSetupUC ROB_1, egm_id, "default", "UCdevice", \Pose;
```

```
      ENDIF
      !Giải mã gói tin từ thiết bị bên ngoài
        EGMActPose egm_id\Tool:=tool0,
            egm_correction_frame,
            EGM_FRAME_BASE,
            egm_sensor_frame,
            EGM_FRAME_BASE
            \x:=egm_minmax_translation
            \y:=egm_minmax_translation
            \z:=egm_minmax_translation
            \rx:=egm_minmax_rotation
            \ry:=egm_minmax_rotation
            \rz:=egm_minmax_rotation
            \LpFilter:=16
            \MaxSpeedDeviation:=30;

        ! Thực hiện chuyển động dựa trên mục tiêu tư thế được thiết bị bên ngoài gửi đến
        EGMRunPose egm_id, EGM_STOP_HOLD \x \y \z \Rx \Ry \Rz \CondTime:=1
\RampInTime:=0;
        WaitTime 0.004;  ! Tần số cập nhật 250Hz
      ENDWHILE

      ERROR
      IF ERRNO = ERR_UDPUC_COMM THEN
        TPWrite "EGM Error: External device not detected.";
        TRYNEXT;
      ENDIF
    ENDPROC
ENDMODULE
```

PHỤ LỤC 2: MÃ NGUỒN CHƯƠNG TRÌNH PYTHON GIAO TIẾP VỚI ROBOT

```
import sys
import socket
from datetime import datetime
import threading
import time
from tracemalloc import reset_peak
import requests
from requests.auth import HTTPBasicAuth
import urllib3
import egm_pb2 as egm
from lxml import etree
import cv2
import mediapipe as mp
from PyQt5.QtCore import pyqtSignal, QThread, QTimer
from PyQt5.QtWidgets import (
    QApplication, QWidget, QVBoxLayout, QLabel, QLineEdit, QPushButton,QMessageBox,
    QHBoxLayout, QSlider, QGroupBox, QGridLayout, QComboBox, QCompleter, QStyle,
QTextEdit,QTabWidget)
from PyQt5.QtGui import QImage, QPixmap, QIcon
from PyQt5.QtCore import Qt,QTimer,pyqtSignal
from collections import deque
import numpy as np
import pyqtgraph as pg
from ws4py.client.threadedclient import WebSocketClient
import xml.etree.ElementTree as ET
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

# ==================== Global Variables ====================
Pos = [500.0, 0.0, 600.0]
Euler = [0.0, 0.0, 0.0]
num = 0
lock = threading.Lock()

# ==================== EGM Handler ====================
```

```
class ReceiveThread(QThread):
    update_feedback = pyqtSignal(float, float, float,float, float, float)
    update_egm_state = pyqtSignal(str)

    def __init__(self, sock):
        super().__init__()
        self.sock = sock
        self.running = True

    def run(self):
        global num
        while self.running:
            try:
                data, addr = self.sock.recvfrom(2048)
                msg = egm.EgmRobot()
                msg.ParseFromString(data)
                num = msg.header.seqno + 1
                self.update_feedback.emit(msg.feedBack.cartesian.pos.x,
                                          msg.feedBack.cartesian.pos.y,
                                          msg.feedBack.cartesian.pos.z,
                                          msg.feedBack.cartesian.euler.x,
                                          msg.feedBack.cartesian.euler.y,
                                          msg.feedBack.cartesian.euler.z)

self.update_egm_state.emit(egm.EgmMCIState.MCIStateType.Name(msg.mciState.state))
            except Exception as e:
                print("EGM receive error:", e)
                time.sleep(0.01)

def CreateSensorMessage(seq_num, pos, euler):
    egmSensor = egm.EgmSensor()
    egmSensor.header.seqno = seq_num
    egmSensor.header.mtype = egm.EgmHeader.MessageType.MSGTYPE_CORRECTION
    pose = egmSensor.planned.cartesian
    pose.pos.x, pose.pos.y, pose.pos.z = pos
    pose.euler.x, pose.euler.y, pose.euler.z = euler
    return egmSensor

class SendThread(QThread):
```

```
    def __init__(self, sock, addr):
        super().__init__()
        self.sock = sock
        self.addr = addr
        self.running = True

    def run(self):
        global num, Pos, Euler
        while self.running:
            with lock:
                pos = Pos.copy()
                euler = Euler.copy()
            msg = CreateSensorMessage(num, pos, euler)
            self.sock.sendto(msg.SerializeToString(), self.addr)
            num += 1
            time.sleep(0.004)

# ==================== RWS Handler ====================
class RwsWebSocketClient(WebSocketClient):
    def __init__(self, url, headers, parent=None):
        super().__init__(url, protocols=['rws_subscription'], headers=headers)
        self.parent = parent
    def opened(self):
        print("RWS WebSocket connection established")
    def closed(self, code, reason=None):
        print(f"RWS WebSocket closed: {code} - {reason}")
    def received_message(self, message):
        if message.is_text:
            try:
                self.parent.process_rws_event(message.data.decode('utf-8'))
            except Exception as e:
                print(f'Error processing message: {e}")

class RwsClient:
    def __init__(self, ip, port=443, username="Admin", password="robotics"):
        self.session = requests.Session()
        self.ip = ip
        self.port = port
        self.base_url = f"https://{ip}:{port}/rw/"
```

```
        self.auth = HTTPBasicAuth(username, password)
        self.headers = {"Accept": "application/xhtml+xml;v=2.0"}
        self.post_headers = {
            "Content-Type": "application/x-www-form-urlencoded;v=2.0",
            "Accept": "application/xhtml+xml;v=2.0"
        }
        self.reg_url = f"https://{ip}:{port}/users/register/local"
        self.identity_url = f"https://{ip}:{port}/ctrl/identity"
        self.subscription_url = f"https://{ip}:{port}/subscription"
        self.ws_client = None
        self.subscription_thread = None
        self.running = False

    def register_user_local(self):
        try:
            url = self.reg_url
            data = {
                "username": "abc",
                "application": "RobotStudio",
                "location" : "Labbr",
                "local-key" : "123456"
            }
            r = self.session.post(url, data=data,
                                  headers=self.post_headers,
                                  auth=self.auth, verify=False)
            return r.status_code
        except Exception as e:
            return str(e)

    def get_identity(self):
        try:
            response = self.session.get(
                self.identity_url,
                headers=self.headers,
                auth=self.auth,
                verify=False
            )

            if response.status_code != 200:
```

```
            return {"error": f"HTTP Error: {response.status_code}"}

        root = etree.fromstring(response.content)
        ns = {'xhtml': 'http://www.w3.org/1999/xhtml'}

        name = root.xpath(
            './/xhtml:li[@class="ctrl-identity-info"]/xhtml:span[@class="ctrl-name"]/text()',
            namespaces=ns
        )
        ctrl_type = root.xpath(
            './/xhtml:li[@class="ctrl-identity-info"]/xhtml:span[@class="ctrl-type"]/text()',
            namespaces=ns
        )

        return {
            "name": name[0].strip() if name else "N/A",
            "type": ctrl_type[0].strip() if ctrl_type else "N/A"
        }

    except Exception as e:
        return {"error": f"Request failed: {str(e)}"}

def set_opmode(self, mode):
    try:
        r = self.session.post(self.base_url + "panel/opmode", data={"opmode": mode},
                              headers=self.post_headers, auth=self.auth, verify=False)

        return r.status_code
    except Exception as e:
        return str(e)

def opmode_ack(self):
    try:
        r = self.session.post(self.base_url + "/panel/opmode/acknowledge", data={"opmode":
"auto"},
                              headers=self.post_headers, auth=self.auth, verify=False)

        return r.status_code
    except Exception as e:
```

```
            return str(e)

    def get_opmde(self):
        try:
            response = self.session.get(
                self.base_url + "/panel/opmode",
                headers=self.headers,
                auth=self.auth,
                verify=False,
                timeout=2
            )
            if response.status_code != 200:
                return f"HTTP Error: {response.status_code}"
            root = etree.fromstring(response.content)
            ns = {'xhtml': 'http://www.w3.org/1999/xhtml'}
            state = root.xpath(
                './/xhtml:li[@class="pnl-opmode"]/xhtml:span[@class="opmode"]/text()',
                namespaces=ns
            )
            return state[0].strip().lower() if state else "unknown"
        except Exception as e:
            return f"Error: {str(e)}"

    def set_ctrl_state(self, state):
        try:
            r = self.session.post(self.base_url + "panel/ctrl-state", data={"ctrl-state": state},
                        headers=self.post_headers, auth=self.auth, verify=False)
            return r.status_code
        except Exception as e:
            return str(e)

    def get_ctrl_state(self):
        try:
            response = self.session.get(
                self.base_url + "panel/ctrl-state",
                headers=self.headers,
                auth=self.auth,
                verify=False,
                timeout=2
```

```
            )
            if response.status_code != 200:
                return f"HTTP Error: {response.status_code}"
            root = etree.fromstring(response.content)
            ns = {'xhtml': 'http://www.w3.org/1999/xhtml'}
            state = root.xpath(
                './/xhtml:li[@class="pnl-ctrlstate"]/xhtml:span[@class="ctrlstate"]/text()',
                namespaces=ns
            )
            return state[0].strip().lower() if state else "unknown"
        except Exception as e:
            return f"Error: {str(e)}"

    def start_rapid(self):
        try:
            url = self.base_url + "rapid/execution/start"
            data = {
                "regain": "clear",
                "execmode": "continue",
                "cycle": "forever",
                "condition": "none",
                "stopatbp": "disabled",
                "alltaskbytsp": "true"
            }
            r = self.session.post(url, data=data,
                                  headers=self.post_headers,
                                  auth=self.auth, verify=False)
            return r.status_code
        except Exception as e:
            return str(e)

    def stop_rapid(self):
        try:
            url = self.base_url + "rapid/execution/stop?mastership=implicit"
            data = {
                "stopmode": "stop",
                "usetsp": "true"
            }
            r = self.session.post(url, data=data,
```

```
                    headers=self.post_headers,
                    auth=self.auth, verify=False)
        return r.status_code
    except Exception as e:
        return str(e)

def reset_rapid(self):
    try:
        url = self.base_url + "rapid/execution/resetpp?mastership=implicit"
        data = {
        }
        r = self.session.post(url, data=data,
                    headers=self.post_headers,
                    auth=self.auth, verify=False)
        return r.status_code
    except Exception as e:
        return str(e)

def get_rapid_state(self):
    try:
        response = self.session.get(
            self.base_url + "/rapid/execution",
            headers=self.headers,
            auth=self.auth,
            verify=False,
            timeout=2
        )
        if response.status_code != 200:
            return f"HTTP Error: {response.status_code}"
        root = etree.fromstring(response.content)
        ns = {'xhtml': 'http://www.w3.org/1999/xhtml'}
        state = root.xpath(
            './/xhtml:li[@class="rap-execution"]/xhtml:span[@class="ctrlexecstate"]/text()',
            namespaces=ns
        )
        return state[0].strip().lower() if state else "unknown"
    except Exception as e:
        return f"Error: {str(e)}"
```

```
    def set_IO(self, lvalue):
        try:
            r = self.session.post(self.base_url + "/iosystem/signals/EtherNetIP/ABB_Scalable_IO1/ABB_Scalable_IO1_0_DO2/set-value",
                                  data={"lvalue": lvalue},
                                  headers=self.post_headers, auth=self.auth, verify=False)
            return r.status_code
        except Exception as e:
            return str(e)
    def get_IO_state(self):
        try:
            response = self.session.get(
                self.base_url + "/iosystem/signals/EtherNetIP/ABB_Scalable_IO1/ABB_Scalable_IO1_0_DO2",
                headers=self.headers,
                auth=self.auth,
                verify=False,
                timeout=2
            )
            if response.status_code != 200:
                return f"HTTP Error: {response.status_code}"
            root = etree.fromstring(response.content)
            ns = {'xhtml': 'http://www.w3.org/1999/xhtml'}
            state = root.xpath(
                './/xhtml:li[@class="ios-signal-li"]/xhtml:span[@class="lvalue"]/text()',
                namespaces=ns
            )
            return state[0].strip().lower() if state else "unknown"
        except Exception as e:
            return f"Error: {str(e)}"

    def set_speed(self, ratio):
        try:
            r = self.session.post(self.base_url + "panel/speedratio?mastership=implicit",
                                  data={"speed-ratio": str(ratio)},
                                  headers=self.post_headers, auth=self.auth, verify=False)
            return r.status_code
        except Exception as e:
            return str(e)
```

```
def get_speed(self):
    try:
        response = self.session.get(
            self.base_url + "panel/speedratio",
            headers=self.headers,
            auth=self.auth,
            verify=False,
            timeout=2
        )
        if response.status_code != 200:
            return f"HTTP Error: {response.status_code}"
        root = etree.fromstring(response.content)
        ns = {'xhtml': 'http://www.w3.org/1999/xhtml'}
        speed_1 = root.xpath(
            './/xhtml:li[@class="pnl-speedratio"]/xhtml:span[@class="speedratio"]/text()',
            namespaces=ns
        )
        return int(speed_1[0].strip().lower()) if speed_1 else "unknown"
    except Exception as e:
        return f"Error: {str(e)}"

def subscribe(self, parent):
    try:
        payload = {
            'resources': ['1','2','3'],
            '1': '/rw/panel/speedratio',
            '1-p': '1',
            '2': '/rw/panel/ctrl-state',
            '2-p': '1',
            '3': '/rw/panel/opmode',
            '3-p': '1'
        }

        resp = self.session.post(self.subscription_url,
                                 auth=self.auth,
                                 headers=self.post_headers,
                                 data=payload,
                                 verify=False)
```

```
            if resp.status_code == 201:
                location = resp.headers['Location']

                cookies = []
                for cookie in self.session.cookies:
                    cookies.append(f"{cookie.name}={cookie.value}")
                cookie_str = "; ".join(cookies)

                auth_header = HTTPBasicAuth(self.auth.username, self.auth.password)
                auth_value = auth_header(self.session).headers['Authorization']

                headers = [
                    ('Cookie', cookie_str),
                    ('Authorization', auth_value)
                ]
                ws_url = location.replace('https://', 'wss://')

                self.ws_client = RwsWebSocketClient(ws_url, headers, parent)
                self.subscription_thread = threading.Thread(target=self.ws_client.connect)
                self.subscription_thread.daemon = True
                self.subscription_thread.start()
                return True
            else:
                print(f"Failed to subscribe: {resp.status_code}")
                return False
        except Exception as e:
            print(f"Subscription error: {e}")
            return False

#================================CAMERA
Hanlder==================================
class HandDetector:
    def __init__(self):
        self.mp_hands = mp.solutions.hands
        self.hands = self.mp_hands.Hands(max_num_hands=1)
        self.mp_draw = mp.solutions.drawing_utils

    def find_hands(self, frame):
        img_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
```

```
        results = self.hands.process(img_rgb)
        hand_landmarks = []

        if results.multi_hand_landmarks:
            for hand_landmark in results.multi_hand_landmarks:
                self.mp_draw.draw_landmarks(frame, hand_landmark,
self.mp_hands.HAND_CONNECTIONS)

                h, w, _ = frame.shape
                landmarks = []
                for id, lm in enumerate(hand_landmark.landmark):
                    cx, cy = int(lm.x * w), int(lm.y * h)
                    landmarks.append([id, cx, cy])
                if landmarks:
                    hand_landmarks.append(landmarks)

        return frame, hand_landmarks

    def count_fingers(self, landmarks):
        if not landmarks:
            return -1

        finger_indices = [4, 8, 12, 16, 20]
        count = 0

        if landmarks[finger_indices[0]][1] < landmarks[finger_indices[0] - 1][1]:
            count += 1

        for idx in finger_indices[1:]:
            if landmarks[idx][2] < landmarks[idx - 2][2]:
                count += 1

        return count

def is_hand_in_area(x, y, w, h, landmarks):
    if not landmarks:
        return False
    for hand in landmarks:
        for _, lx, ly in hand:
```

```
            if x < lx < x + w and y < ly < y + h:
                return True
    return False

def draw_button(frame, x, y, w, h, text, color=None, font_scale=0.5, thickness=1, bg_color=(200, 200, 200)):
    if color is None:
        color = (0, 0, 0)
    cv2.rectangle(frame, (x, y), (x + w, y + h), bg_color, 1)
    cv2.rectangle(frame, (x, y), (x + w, y + h), color, 2)
    (tw, th), _ = cv2.getTextSize(text, cv2.FONT_HERSHEY_SIMPLEX, font_scale, thickness)
    tx = x + (w - tw) // 2
    ty = y + (h + th) // 2
    cv2.putText(frame, text, (tx, ty), cv2.FONT_HERSHEY_SIMPLEX, font_scale, color, thickness)

class CameraThread(QThread):
    frame_processed = pyqtSignal(QImage, list)
    button_pressed = pyqtSignal(str)
    gripper_changed = pyqtSignal(bool)
    reset_xyz_requested = pyqtSignal()
    reset_rxyz_requested = pyqtSignal()
    def __init__(self):
        super().__init__()
        self.detector = HandDetector()
        self.running = False
        self.process_hands = False
        self.camera_index = 0
        self.states = {
            'hand_mode': False,
            'current_mode': None,
            'gripper': False,
            'xyz': {'X': 500, 'Y': 0, 'Z': 600},
            'rxyz': {'RX': 180, 'RY': 0, 'RZ': 180},
            'xyz_timers': {'X+': 0, 'X-': 0, 'Y+': 0, 'Y-': 0, 'Z+': 0, 'Z-': 0},
            'rxyz_timers': {'RX+': 0, 'RX-': 0, 'RY+': 0, 'RY-': 0, 'RZ+': 0, 'RZ-': 0},
            'mode_timers': {}}

    def run(self):
```

```
        cap = cv2.VideoCapture(self.camera_index)
        while self.running:
            ret, frame = cap.read()
            if not ret:
                print("Failed to capture frame")
                continue

            frame = cv2.flip(frame, 1)

            if self.process_hands:
                frame, landmarks = self.detector.find_hands(frame)
                self.process_controls(frame, landmarks)
            else:
                landmarks = []

            rgb_image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            h, w, ch = rgb_image.shape
            qt_image = QImage(rgb_image.data, w, h, ch * w, QImage.Format_RGB888)
            self.frame_processed.emit(qt_image, landmarks)

        cap.release()
        print("Camera thread stopped")

    def process_controls(self, frame, landmarks):
        current_time = time.time()
        # Gripper Control Mode
        if self.states['hand_mode']:

            draw_button(frame, 20, 100, 120, 40, "Position Mode", (255,0,255))
            if is_hand_in_area(20, 100, 120, 40, landmarks):
                if 'position' not in self.states['mode_timers']:
                    self.states['mode_timers']['position'] = current_time
                elif current_time - self.states['mode_timers']['position'] >= 1.0:
                    self.states['hand_mode'] = False
                    self.states['current_mode'] = 'xyz'
                    del self.states['mode_timers']['position']
            else:
                if 'position' in self.states['mode_timers']:
                    del self.states['mode_timers']['position']
```

```
        draw_button(frame, 150, 100, 120, 40, "Rotation Mode", (255, 0, 0))
        if is_hand_in_area(150, 100, 120, 40, landmarks):
            if 'rotation' not in self.states['mode_timers']:
                self.states['mode_timers']['rotation'] = current_time
            elif current_time - self.states['mode_timers']['rotation'] >= 1.0:
                self.states['hand_mode'] = False
                self.states['current_mode'] = 'rxyz'
                del self.states['mode_timers']['rotation']
        else:
            if 'rotation' in self.states['mode_timers']:
                del self.states['mode_timers']['rotation']

        area_x, area_y = 350, 50
        area_w, area_h = 250, 250
        cv2.rectangle(frame, (area_x, area_y), (area_x + area_w, area_y + area_h), (0, 255, 0),
2)

        if is_hand_in_area(area_x, area_y, area_w, area_h, landmarks):
            fingers = self.detector.count_fingers(landmarks[0] if landmarks else [])
            if fingers in [0, 5]:
                new_state = (fingers == 0)

                if new_state != self.states['gripper']:
                    if 'gripper_hold' not in self.states['mode_timers']:
                        self.states['mode_timers']['gripper_hold'] = current_time
                    elif current_time - self.states['mode_timers']['gripper_hold'] >= 1.0:
                        self.states['gripper'] = new_state
                        self.gripper_changed.emit(new_state)
                        del self.states['mode_timers']['gripper_hold']
                else:
                    if 'gripper_hold' in self.states['mode_timers']:
                        del self.states['mode_timers']['gripper_hold']
            else:
                if 'gripper_hold' in self.states['mode_timers']:
                    del self.states['mode_timers']['gripper_hold']
        else:

            if 'gripper_hold' in self.states['mode_timers']:
```

```
                del self.states['mode_timers']['gripper_hold']

            cv2.putText(frame, f"Gripper: {'ON' if self.states['gripper'] else 'OFF'}",
                        (area_x + 10, area_y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 0, 0),
2)

        # XYZ Position Control Mode
        elif self.states['current_mode'] == 'xyz':
            xyz_labels = ['-X', '+X', '-Y', '+Y', '-Z', '+Z']

            draw_button(frame, 10, 200, 120, 40, "Hand Mode", (0,255,255))
            if is_hand_in_area(10, 200, 120, 40, landmarks):
                if 'hand_xyz' not in self.states['mode_timers']:
                    self.states['mode_timers']['hand_xyz'] = current_time
                elif current_time - self.states['mode_timers']['hand_xyz'] >= 1.0:
                    self.states['hand_mode'] = True
                    del self.states['mode_timers']['hand_xyz']
            else:
                if 'hand_xyz' in self.states['mode_timers']:
                    del self.states['mode_timers']['hand_xyz']

            draw_button(frame, 10, 250, 120, 40, "Rotation Mode", (255, 0, 0))
            if is_hand_in_area(10, 250, 120, 40, landmarks):
                if 'rotation_xyz' not in self.states['mode_timers']:
                    self.states['mode_timers']['rotation_xyz'] = current_time
                elif current_time - self.states['mode_timers']['rotation_xyz'] >= 1.0:
                    self.states['current_mode'] = 'rxyz'
                    del self.states['mode_timers']['rotation_xyz']
            else:
                if 'rotation_xyz' in self.states['mode_timers']:
                    del self.states['mode_timers']['rotation_xyz']

            for i, label in enumerate(xyz_labels):
                btn_x = 220 + i * 70
                draw_button(frame, btn_x, 30, 60, 40, label, (255,0,255))
                axis = label[1]
                delta = -3 if label[0] == '-' else 3
                key = f'{axis}{label[0]}'
                current_time = time.time()
```

```
            if is_hand_in_area(btn_x, 30, 60, 40, landmarks):
                last_time = self.states['xyz_timers'][key]
                if last_time == 0 or current_time - last_time > 0.2:
                    self.states['xyz'][axis] += delta
                    self.states['xyz_timers'][key] = current_time
                    self.button_pressed.emit(label)
            else:
                self.states['xyz_timers'][key] = 0

        draw_button(frame, 150, 30, 60, 40, "Reset", (0, 0, 255))
        if is_hand_in_area(150, 30, 60, 40, landmarks):
            if 'reset_timer' not in self.states['mode_timers']:
                self.states['mode_timers']['reset_timer'] = current_time
            elif current_time - self.states['mode_timers']['reset_timer'] >= 1.0:
                self.states['xyz'] = {'X': 500, 'Y': 0, 'Z': 600}
                self.reset_xyz_requested.emit()
                del self.states['mode_timers']['reset_timer']
        else:
            if 'reset_timer' in self.states['mode_timers']:
                del self.states['mode_timers']['reset_timer']

        cv2.putText(frame, f"POSITION MODE",(10,20), cv2.FONT_HERSHEY_SIMPLEX,
0.6, (255,0,255), 2)
    # RXYZ Rotation Control Mode
    elif self.states['current_mode'] == 'rxyz':
        rxyz_labels = ['-RX', '+RX', '-RY', '+RY', '-RZ', '+RZ']

        draw_button(frame,10, 200, 120, 40, "Hand Mode", (0,255,255))
        if is_hand_in_area(10, 200, 120, 40, landmarks):
            if 'hand_rxyz' not in self.states['mode_timers']:
                self.states['mode_timers']['hand_rxyz'] = current_time
            elif current_time - self.states['mode_timers']['hand_rxyz'] >= 1.0:
                self.states['hand_mode'] = True
                del self.states['mode_timers']['hand_rxyz']
        else:
            if 'hand_rxyz' in self.states['mode_timers']:
                del self.states['mode_timers']['hand_rxyz']

        draw_button(frame, 10, 250, 120, 40, "Position Mode", (255,0,255))
```

```
        if is_hand_in_area(10, 250, 120, 40, landmarks):
            if 'position_rxyz' not in self.states['mode_timers']:
                self.states['mode_timers']['position_rxyz'] = current_time
            elif current_time - self.states['mode_timers']['position_rxyz'] >= 1.0:
                self.states['current_mode'] = 'xyz'
                del self.states['mode_timers']['position_rxyz']
        else:
            if 'position_rxyz' in self.states['mode_timers']:
                del self.states['mode_timers']['position_rxyz']

        for i, label in enumerate(rxyz_labels):
            btn_x = 220 + i * 70
            draw_button(frame, btn_x, 30, 60, 40, label, (255, 0, 0))
            axis = label[1:]
            delta = -0.1 if label[0] == '-' else 0.1
            current_time = time.time()
            if is_hand_in_area(btn_x, 30, 60, 40, landmarks):
                last_time = self.states['rxyz_timers'].get(axis, 0)
                if last_time == 0 or current_time - last_time > 0.4:
                    self.states['rxyz'][axis] += delta
                    self.states['rxyz_timers'][axis] = current_time
                    self.button_pressed.emit(label)
            else:
                self.states['rxyz_timers'][axis] = 0

        draw_button(frame, 150, 30, 60, 40, "Reset", (0, 0, 255))
        if is_hand_in_area(150, 30, 60, 40, landmarks):
            if 'reset_timer' not in self.states['mode_timers']:
                self.states['mode_timers']['reset_timer'] = current_time
            elif current_time - self.states['mode_timers']['reset_timer'] >= 1.0:
                self.states['rxyz'] = {'RX': 180, 'RY': 0, 'RZ': 180}
                self.reset_rxyz_requested.emit()
                del self.states['mode_timers']['reset_timer']
        else:
            if 'reset_timer' in self.states['mode_timers']:
                del self.states['mode_timers']['reset_timer']

        cv2.putText(frame, f"ROTATION MODE",(10,20),
cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 0, 0), 2)
```

```
#====================HOLDBUTTON
Hanlder============================
class HoldButton(QPushButton):
    pressed = pyqtSignal()
    released = pyqtSignal()

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self._timer = QTimer(self)
        self._timer.timeout.connect(self._handle_repeat)
        self._initial_delay = 300
        self._repeat_interval = 100

    def mousePressEvent(self, event):
        self.pressed.emit()
        self._timer.start(self._initial_delay)
        super().mousePressEvent(event)

    def mouseReleaseEvent(self, event):
        self.released.emit()
        self._timer.stop()
        super().mouseReleaseEvent(event)

    def _handle_repeat(self):
        self._timer.setInterval(self._repeat_interval)
        self.clicked.emit()

# ==================== GUI ====================
class MainWindow(QWidget):
    speed_changed = pyqtSignal(int)
    opmode_changed = pyqtSignal(str)
    motor_state_changed = pyqtSignal(bool)

    def __init__(self):
        super().__init__()
        self.rws_connected = False
        self.robot_socket = None
        self.initial_pose_received = False
        self.setWindowTitle("CRB15000 Robot Control")
```

```
        self.setWindowIcon(QIcon("D:/2024-2025
KI_2/PythonApplication1/PythonApplication1/icon.png"))

        self.setStyleSheet("""
            QWidget {
                font-family: Segoe UI;
                font-size: 10pt;
            }
            QGroupBox {
                border: 1px solid #cccccc;
                border-radius: 5px;
                margin-top: 1ex;
            }
            QGroupBox::title {
                subcontrol-origin: margin;
                left: 10px;
                padding: 0 3px;
                color: #555555;
            }
            QPushButton {
                background-color: #f0f0f0;
                border: 1px solid #cccccc;
                border-radius: 4px;
                padding: 5px 10px;
                min-width: 80px;
            }
            QPushButton:hover {
                background-color: #e0e0e0;
            }
            QLineEdit {
                border: 1px solid #cccccc;
                border-radius: 4px;
                padding: 3px;
            }
            QSlider::groove:horizontal {
                border: 1px solid #bbb;
                background: #ddd;
                height: 5px;
                border-radius: 3px;
```

```
    }

    QSlider::sub-page:horizontal {
        background: qlineargradient(
            x1:0, y1:0, x2:1, y2:0,
            stop:0 #409EFF,
            stop:1 #409EFF
        );
        border: 1px;
        height: 5px;
        border-radius: 2px;
    }

    QSlider::add-page:horizontal {
        background: #ccc;
        border: none;
        height: 5px;
        border-radius: 2px;
    }

    QSlider::handle:horizontal {
        background: white;
        border: 1px solid #409EFF;
        width: 14px;
        margin: -5px 0;
        border-radius: 6px;
    }

    QSlider::handle:horizontal:hover {
        background: #66baff;
        border: 1px solid #0077cc;
    }
""")

#
self.speed_changed.connect(self.update_speed_ui)
self.opmode_changed.connect(self.update_opmode_ui)
self.motor_state_changed.connect(self.update_motor_ui)
```

```
# ===== Main Layout =====
main_layout = QVBoxLayout()
main_layout.setContentsMargins(5, 5, 5, 5)



# ===== Control Layout =====
main_control_layout = QHBoxLayout()
main_control_layout.setContentsMargins(0, 0, 0, 0)
main_control_layout.setSpacing(10)

    # +++++++ Left Panel (EGM) +++++++
left_panel = QWidget()
left_layout = QVBoxLayout(left_panel)
left_layout.setContentsMargins(0, 0, 0, 0)
left_layout.setSpacing(10)
egm_group = self.create_egm_group()
left_layout.addWidget(egm_group)
main_control_layout.addWidget(left_panel, 2)

    # +++++++ Center Panel (CAMERA) +++++++
center_panel = QWidget()
center_layout = QVBoxLayout(center_panel)
center_layout.setContentsMargins(0, 0, 0, 0)
cam_group = self.init_camera_ui()
center_layout.addWidget(cam_group)
main_control_layout.addWidget(center_panel, 2)

    # +++++++ Right Panel (RWS) +++++++
right_panel = QWidget()
right_layout = QVBoxLayout(right_panel)
right_layout.setContentsMargins(0, 0, 0, 0)
right_layout.setSpacing(10)
rws_group = self.create_rws_group()
right_layout.addWidget(rws_group)
main_control_layout.addWidget(right_panel, 2)

# ===== System Log =====
```

```
        main_log_layout = QHBoxLayout()
        main_log_layout.setContentsMargins(0, 0, 0, 0)
        self.log_output = QTextEdit()
        self.log_output.setReadOnly(True)
        self.log_output.setFixedHeight(110)
        self.btn_clear_log = QPushButton("Clear Log")
        self.btn_clear_log.setIcon(QIcon("D:/2024-2025
KI_2/PythonApplication1/PythonApplication1/trash.png"))
        self.btn_clear_log.clicked.connect(self.clear_log)
        main_log_layout.addWidget(self.log_output, 9)
        main_log_layout.addWidget(self.btn_clear_log, 1)

        #=======================================
        main_layout.addLayout(main_control_layout)
        main_layout.addLayout(main_log_layout)
        self.setLayout(main_layout)



        # Khoi tao cameraThread
        self.camera_thread = CameraThread()
        self.camera_thread.frame_processed.connect(self.update_camera_preview)
        self.camera_thread.button_pressed.connect(self.handle_camera_button)
        self.camera_thread.gripper_changed.connect(self.handle_gripper_change)
        self.camera_thread.reset_xyz_requested.connect(self.reset_xyz_position)
        self.camera_thread.reset_rxyz_requested.connect(self.reset_rxyz_rotation)
        # Thread and connection variables
        self.recv_thread = None
        self.send_thread = None
        self.connected = False
        self.addr = None
        self.rws = RwsClient("127.0.0.1")

#********Camera Group************
    def init_camera_ui(self):
        camera_group = QGroupBox("Camera Control")
        camera_layout = QVBoxLayout(camera_group)
        camera_layout.setContentsMargins(5, 15, 5, 15)
        camera_layout.setSpacing(10)
```

```
        #Camera Selection
        select_group = QGroupBox()
        select_layout = QHBoxLayout(select_group)
        select_layout.setContentsMargins(10, 10, 10, 10)
        select_layout.addWidget(QLabel("Select Camera:"))
        self.camera_combo = QComboBox()
        self.camera_combo.addItem("Camera 0", 0)
        self.camera_combo.addItem("Camera 1", 1)
        select_layout.addWidget(self.camera_combo)
        camera_layout.addWidget(select_group)

        #Camera Stream
        stream_group = QGroupBox()
        stream_layout = QVBoxLayout(stream_group)
        stream_layout.setContentsMargins(0, 5, 0, 5)
        self.camera_label = QLabel()
        self.camera_label.setFixedSize(750, 550)
        self.camera_label.setAlignment(Qt.AlignCenter)
        stream_layout.addWidget(self.camera_label)
        camera_layout.addWidget(stream_group)
        #Camera Controls
        control_group = QGroupBox()
        control_layout = QGridLayout(control_group)
        #control_layout.setContentsMargins(10, 15, 10, 15)
        self.btn_camera = QPushButton("Start Camera")
        self.btn_camera.clicked.connect(self.toggle_camera)
        self.btn_hand_tracking = QPushButton("Enable Hand Tracking")
        self.btn_hand_tracking.clicked.connect(self.toggle_hand_tracking)
        self.btn_hand_control = QPushButton("Hand Control Mode")
        self.btn_hand_control.clicked.connect(self.toggle_hand_control)
        control_layout.addWidget(self.btn_camera,0,0)
        control_layout.addWidget(self.btn_hand_tracking,0,1)
        control_layout.addWidget(self.btn_hand_control,0,2)
        control_group.setLayout(control_layout)
        camera_layout.addWidget(control_group)

        return camera_group

#********Egm Group*************
```

```
def create_egm_group(self):
    egm_group = QGroupBox("EGM Control")
    egm_layout = QVBoxLayout()
    egm_layout.setContentsMargins(12, 15, 12, 15)
    egm_layout.setSpacing(10)

    # Egm Connect
    connect_group = QGroupBox()
    conn_layout = QGridLayout()
    self.egm_ip = QLineEdit("192.168.125.10")
    ip_suggest = [
    "192.168.125.10",
    "127.0.0.1"]
    completer = QCompleter(ip_suggest)
    completer.setFilterMode(Qt.MatchContains)
    completer.setCaseSensitivity(Qt.CaseInsensitive)
    self.egm_ip.setCompleter(completer)
    self.egm_port = QLineEdit("6510")
    self.btn_connect = QPushButton("Start EGM")
    self.btn_connect.clicked.connect(self.toggle_egm)
    self.btn_connect.setStyleSheet("background-color: #d7e9f9;")
    self.lbl_egm_state = QLabel("EGM State: Unknown")
    conn_layout.addWidget(QLabel("Robot IP:"), 0, 0)
    conn_layout.addWidget(self.egm_ip, 0, 1)
    conn_layout.addWidget(QLabel("Port:"), 1, 0)
    conn_layout.addWidget(self.egm_port, 1, 1)
    conn_layout.addWidget(self.btn_connect, 2, 0, 1, 2)
    conn_layout.addWidget(self.lbl_egm_state,3,0,1,2)
    connect_group.setLayout(conn_layout)
    egm_layout.addWidget(connect_group)

    # Position controls
    pos_group = QGroupBox("Position (mm)")
    pos_layout = QVBoxLayout()
    pos_layout.setContentsMargins(10, 20, 10, 10)
    pos_layout.setSpacing(10)
    self.lbl_feedback1 = QLabel("Current Pos: X=0.00 Y=0.00 Z=0.00")
    pos_layout.addWidget(self.lbl_feedback1)
    # X axis controls
```

```
x_layout = QHBoxLayout()
self.btn_x_minus = HoldButton("-X")
self.btn_x_minus.pressed.connect(lambda: self.start_adjust(0, -1))
self.btn_x_minus.released.connect(self.stop_adjust)
self.lbl_x = QLabel("0.0")
x_layout.addWidget(self.btn_x_minus)
x_layout.addWidget(QLabel("X:"))
x_layout.addWidget(self.lbl_x)
self.btn_x_plus = HoldButton("+X")
self.btn_x_plus.pressed.connect(lambda: self.start_adjust(0, 1))
self.btn_x_plus.released.connect(self.stop_adjust)
x_layout.addWidget(self.btn_x_plus)
pos_layout.addLayout(x_layout)

# Y axis controls
y_layout = QHBoxLayout()
self.btn_y_minus = HoldButton("-Y")
self.btn_y_minus.pressed.connect(lambda: self.start_adjust(1, -1))
self.btn_y_minus.released.connect(self.stop_adjust)
self.lbl_y = QLabel("0.0")
y_layout.addWidget(self.btn_y_minus)
y_layout.addWidget(QLabel("Y:"))
y_layout.addWidget(self.lbl_y)
self.btn_y_plus = HoldButton("+Y")
self.btn_y_plus.pressed.connect(lambda: self.start_adjust(1, 1))
self.btn_y_plus.released.connect(self.stop_adjust)
y_layout.addWidget(self.btn_y_plus)
pos_layout.addLayout(y_layout)

# Z axis controls
z_layout = QHBoxLayout()
self.btn_z_minus = HoldButton("-Z")
self.btn_z_minus.pressed.connect(lambda: self.start_adjust(2, -1))
self.btn_z_minus.released.connect(self.stop_adjust)
self.lbl_z = QLabel("0.0")
z_layout.addWidget(self.btn_z_minus)
z_layout.addWidget(QLabel("Z:"))
z_layout.addWidget(self.lbl_z)
self.btn_z_plus = HoldButton("+Z")
```

```
self.btn_z_plus.pressed.connect(lambda: self.start_adjust(2, 1))
self.btn_z_plus.released.connect(self.stop_adjust)
z_layout.addWidget(self.btn_z_plus)
pos_layout.addLayout(z_layout)
pos_group.setLayout(pos_layout)

# Rotation controls
rot_group = QGroupBox("Rotation (degree)")
rot_layout = QVBoxLayout()
rot_layout.setContentsMargins(10, 20, 10, 10)
rot_layout.setSpacing(10)
self.lbl_feedback2 = QLabel("Current Rot: RX=0.00 RY=0.00 RZ=0.00")
rot_layout.addWidget(self.lbl_feedback2)

# RX axis controls
rx_layout = QHBoxLayout()
self.btn_rx_minus = HoldButton("-RX")
self.btn_rx_minus.pressed.connect(lambda: self.start_rotate(0, -1))
self.btn_rx_minus.released.connect(self.stop_rotate)
self.lbl_rx = QLabel("0.0")
rx_layout.addWidget(self.btn_rx_minus)
rx_layout.addWidget(QLabel("RX:"))
rx_layout.addWidget(self.lbl_rx)
self.btn_rx_plus = HoldButton("+RX")
self.btn_rx_plus.pressed.connect(lambda: self.start_rotate(0, 1))
self.btn_rx_plus.released.connect(self.stop_rotate)
rx_layout.addWidget(self.btn_rx_plus)
rot_layout.addLayout(rx_layout)

# RY axis controls
ry_layout = QHBoxLayout()
self.btn_ry_minus = HoldButton("-RY")
self.btn_ry_minus.pressed.connect(lambda: self.start_rotate(1, -1))
self.btn_ry_minus.released.connect(self.stop_rotate)
self.lbl_ry = QLabel("0.0")
ry_layout.addWidget(self.btn_ry_minus)
ry_layout.addWidget(QLabel("RY:"))
ry_layout.addWidget(self.lbl_ry)
self.btn_ry_plus = HoldButton("+RY")
```

```
        self.btn_ry_plus.pressed.connect(lambda: self.start_rotate(1, 1))
        self.btn_ry_plus.released.connect(self.stop_rotate)
        ry_layout.addWidget(self.btn_ry_plus)
        rot_layout.addLayout(ry_layout)

        # RZ axis controls
        rz_layout = QHBoxLayout()
        self.btn_rz_minus = HoldButton("-RZ")
        self.btn_rz_minus.pressed.connect(lambda: self.start_rotate(2, -1))
        self.btn_rz_minus.released.connect(self.stop_rotate)
        self.lbl_rz = QLabel("0.0")
        rz_layout.addWidget(self.btn_rz_minus)
        rz_layout.addWidget(QLabel("RZ:"))
        rz_layout.addWidget(self.lbl_rz)
        self.btn_rz_plus = HoldButton("+RZ")
        self.btn_rz_plus.pressed.connect(lambda: self.start_rotate(2, 1))
        self.btn_rz_plus.released.connect(self.stop_rotate)
        rz_layout.addWidget(self.btn_rz_plus)
        rot_layout.addLayout(rz_layout)
        rot_group.setLayout(rot_layout)

        egm_layout.addWidget(pos_group)
        egm_layout.addWidget(rot_group)
        egm_group.setLayout(egm_layout)

        return egm_group
#********Robot Web Services Group*************
    def create_rws_group(self):
        rws_group = QGroupBox("RWS Control")
        rws_layout = QVBoxLayout()
        rws_layout.setContentsMargins(10, 15, 10,10)
        rws_layout.setSpacing(10)

        # RWS Connect
        connect_group = QGroupBox()
        connect_layout = QGridLayout()
        connect_layout.setContentsMargins(10, 10, 10, 10)
        connect_layout.setVerticalSpacing(10)
        connect_layout.setHorizontalSpacing(20)
```

```
self.rws_ip = QLineEdit("192.168.125.1")
ip_suggestions = [
"192.168.125.1",
"127.0.0.1"]
completer = QCompleter(ip_suggestions)
completer.setFilterMode(Qt.MatchContains)
completer.setCaseSensitivity(Qt.CaseInsensitive)
self.rws_ip.setCompleter(completer)
self.rws_port = QLineEdit("443")
self.rws_user = QLineEdit("Admin")
user_suggestions = [
"Admin",
"Default User"]
completer = QCompleter(user_suggestions)
completer.setFilterMode(Qt.MatchContains)
completer.setCaseSensitivity(Qt.CaseInsensitive)
self.rws_user.setCompleter(completer)
self.rws_pass = QLineEdit("robotics")
self.rws_pass.setEchoMode(QLineEdit.Password)
self.btn_rws_connect = QPushButton("Connect RWS")
self.btn_rws_connect.setStyleSheet("background-color: #d7e9f9;")

connect_layout.addWidget(QLabel("IP:"), 0, 0)
connect_layout.addWidget(self.rws_ip, 0, 1)
connect_layout.addWidget(QLabel("Port:"), 1, 0)
connect_layout.addWidget(self.rws_port, 1, 1)
connect_layout.addWidget(QLabel("Username:"), 2, 0)
connect_layout.addWidget(self.rws_user, 2, 1)
connect_layout.addWidget(QLabel("Password:"), 3, 0)
connect_layout.addWidget(self.rws_pass, 3, 1)
connect_layout.addWidget(self.btn_rws_connect, 4, 0, 1, 2)
connect_group.setLayout(connect_layout)
self.btn_rws_connect.clicked.connect(self.toggle_rws_connection)
rws_layout.addWidget(connect_group)

# Identity section
iden_group = QGroupBox("Controller Information")
iden_layout = QVBoxLayout()
iden_layout.setContentsMargins(10, 20, 10, 10)
```

```
iden_layout.setSpacing(10)

identity_layout = QHBoxLayout()
self.lbl_identity = QLabel("Controller Infor: Unknown")
self.lbl_identity.setWordWrap(True)
identity_layout.addWidget(self.lbl_identity, 1)
iden_layout.addLayout(identity_layout)
iden_group.setLayout(iden_layout)
rws_layout.addWidget(iden_group)

#Gripper Control
IO_layout = QVBoxLayout()
IO_button_layout = QHBoxLayout()
self.lbl_IO_status = QLabel("Gripper Status:Unknown")
self.btn_on_io = QPushButton("ON Gripper")
self.btn_on_io.setStyleSheet("background-color: #d4edda;")
self.btn_on_io.clicked.connect(self.ON_IO)
self.btn_off_io = QPushButton("OFF Gripper")
self.btn_off_io.setStyleSheet("background-color: #f8d7da;")
self.btn_off_io.clicked.connect(self.OFF_IO)
IO_button_layout.addWidget(self.btn_on_io)
IO_button_layout.addWidget(self.btn_off_io)
IO_layout.addWidget(self.lbl_IO_status)
IO_layout.addLayout(IO_button_layout)
rws_layout.addLayout(IO_layout)

#Operation Mode Control
Mode_layout = QVBoxLayout()
Mode_button_layout = QHBoxLayout()
self.lbl_opmode = QLabel("Operation Mode: Unknown")
self.btn_mode_Auto = QPushButton("AUTO")
self.btn_mode_Auto.setStyleSheet("background-color: #d4edda;")
self.btn_mode_Auto.clicked.connect(self.mode_auto)
self.btn_mode_Man = QPushButton("MANUAL")
self.btn_mode_Man.setStyleSheet("background-color: #f8d7da;")
self.btn_mode_Man.clicked.connect(self.mode_man)
Mode_layout.addWidget(self.lbl_opmode)
Mode_button_layout.addWidget(self.btn_mode_Auto)
Mode_button_layout.addWidget(self.btn_mode_Man)
```

```
        Mode_layout.addLayout(Mode_button_layout)
        rws_layout.addLayout(Mode_layout)

        # Motor controls
        motor_layout = QVBoxLayout()
        motor_button_layout = QHBoxLayout()
        self.lbl_motor_status = QLabel("Motor Status: Unknown")
        self.btn_motor_on = QPushButton("Motor ON")
        self.btn_motor_on.setStyleSheet("background-color: #d4edda;")
        self.btn_motor_on.clicked.connect(self.motor_on)
        self.btn_motor_off = QPushButton("Motor OFF")
        self.btn_motor_off.setStyleSheet("background-color: #f8d7da;")
        self.btn_motor_off.clicked.connect(self.motor_off)
        motor_layout.addWidget(self.lbl_motor_status)
        motor_button_layout.addWidget(self.btn_motor_on)
        motor_button_layout.addWidget(self.btn_motor_off)
        motor_layout.addLayout(motor_button_layout)
        rws_layout.addLayout(motor_layout)

        # RAPID controls
        rapid_layout = QVBoxLayout()
        rapid_button_layout = QHBoxLayout()
        self.lbl_rapid_status = QLabel("Rapid Status: Unknown")
        self.btn_start_rapid = QPushButton("  Run")
        self.btn_start_rapid.setIcon(QIcon("D:/2024-2025
KI_2/PythonApplication1/PythonApplication1/play.png"))
        self.btn_start_rapid.setStyleSheet("background-color: #d4edda;")
        self.btn_start_rapid.clicked.connect(self.start_rapid)
        self.btn_start_rapid.clicked.connect(self.update_rapid_status)
        self.btn_stop_rapid = QPushButton("  Stop")
        self.btn_stop_rapid.setIcon(QIcon("D:/2024-2025
KI_2/PythonApplication1/PythonApplication1/stop.png"))
        self.btn_stop_rapid.setStyleSheet("background-color: #f8d7da;")
        self.btn_stop_rapid.clicked.connect(self.stop_rapid)
        self.btn_stop_rapid.clicked.connect(self.update_rapid_status)
        self.btn_reset_rapid = QPushButton("Reset PP")
        self.btn_reset_rapid.setIcon(self.style().standardIcon(QStyle.SP_BrowserReload))
        self.btn_reset_rapid.setIcon(QIcon("D:/2024-2025
KI_2/PythonApplication1/PythonApplication1/reset.png"))
```

```
    self.btn_reset_rapid.setStyleSheet("background-color: #FFFFCC;")
    self.btn_reset_rapid.clicked.connect(self.reset_rapid)
    rapid_layout.addWidget(self.lbl_rapid_status)
    rapid_button_layout.addWidget(self.btn_start_rapid)
    rapid_button_layout.addWidget(self.btn_stop_rapid)
    rapid_button_layout.addWidget(self.btn_reset_rapid)
    rapid_layout.addLayout(rapid_button_layout)
    rws_layout.addLayout(rapid_layout)

    # Speed control
    speed_layout = QVBoxLayout()
    speed_layout.addWidget(QLabel("Speed Ratio:"))
    slider_layout = QHBoxLayout()
    self.slider = QSlider(Qt.Horizontal)
    self.slider.setMinimum(0)
    self.slider.setMaximum(100)
    self.slider.setValue(100)
    self.slider.valueChanged.connect(self.set_speed)
    self.lbl_speed = QLabel("100%")
    slider_layout.addWidget(self.slider)
    slider_layout.addWidget(self.lbl_speed)
    speed_layout.addLayout(slider_layout)
    rws_layout.addLayout(speed_layout)
    rws_group.setLayout(rws_layout)

    return rws_group

# ==================== EGM Functions ====================
def toggle_egm(self):
    if not self.connected:
        try:
            # Validate inputs
            computer_ip = self.egm_ip.text()
            robot_port = int(self.egm_port.text())

            # Create new socket
            self.robot_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
            self.robot_socket.bind((computer_ip, robot_port))
```

```
            print(f"Waiting for EGM connection on {computer_ip}:{robot_port}...")
            data, addr = self.robot_socket.recvfrom(2048)
            print(f"Connected to robot at {addr}")

            self.addr = addr
            self.recv_thread = ReceiveThread(self.robot_socket)
            self.recv_thread.update_feedback.connect(self.update_feedback)
            self.recv_thread.update_egm_state.connect(self.update_egm_state)
            self.recv_thread.start()
            self.send_thread = SendThread(self.robot_socket, self.addr)
            self.send_thread.start()
            self.connected = True
            self.btn_connect.setText("Stop EGM")
            QMessageBox.information(self, "Success", "EGM Started!")
            self.add_log("EGM Started!")
            self.initial_pose_received = False

        except ValueError as ve:
            QMessageBox.critical(self, "Input Error", str(ve))
        except Exception as e:
            QMessageBox.critical(self, "Connection Error", str(e))
            if self.robot_socket:
                self.robot_socket.close()
    else:
        if self.recv_thread:
            self.recv_thread.running = False
            self.recv_thread.wait()
        if self.send_thread:
            self.send_thread.running = False
            self.send_thread.wait()
        if self.robot_socket:
            self.robot_socket.close()
        self.connected = False
        self.btn_connect.setText("Start EGM")
        QMessageBox.information(self, "Info", "EGM Stopped!")
        self.add_log("EGM Stopped!")
        self.lbl_egm_state.setText("EGM State: Disconnected")

def start_adjust(self, axis, direction):
```

```
        self.adjust_timer = QTimer()
        self.adjust_timer.timeout.connect(lambda: self.adjust_position(axis, direction))
        self.adjust_timer.start(100)
        self.adjust_position(axis, direction)

    def stop_adjust(self):
        if hasattr(self, 'adjust_timer'):
            self.adjust_timer.stop()
            del self.adjust_timer

    def adjust_position(self, axis, direction):
        global Pos, lock
        step = 3 * direction
        with lock:
            Pos[axis] += step
            self.camera_thread.states['xyz'][['X','Y','Z'][axis]] = Pos[axis]
            if axis == 0:
                self.lbl_x.setText(f"{Pos[axis]:.1f}")
            elif axis == 1:
                self.lbl_y.setText(f"{Pos[axis]:.1f}")
            elif axis == 2:
                self.lbl_z.setText(f"{Pos[axis]:.1f}")

    def update_feedback(self, x, y, z, rx, ry, rz):
        global Pos, Euler, lock
        if not self.initial_pose_received:
            with lock:
                Pos = [x, y, z]
                Euler = [rx,ry,rz]
        self.lbl_x.setText(f"{x:.1f}")
        self.lbl_y.setText(f"{y:.1f}")
        self.lbl_z.setText(f"{z:.1f}")
        self.lbl_rx.setText(f"{Euler[0]:.1f}")
        self.lbl_ry.setText(f"{Euler[1]:.1f}")
        self.lbl_rz.setText(f"{Euler[2]:.1f}")

        self.initial_pose_received = True
        self.lbl_feedback1.setText(
            f"POS: X={x:.2f} Y={y:.2f} Z={z:.2f} "
```

```
        )
        self.lbl_feedback2.setText(
            f"RX={rx:.2f} RY={ry:.2f} RZ={rz:.2f}"
        )

    def start_rotate(self, axis, direction):
        self.rotate_timer = QTimer()
        self.rotate_timer.timeout.connect(lambda: self.adjust_rotation(axis, direction))
        self.rotate_timer.start(100)
        self.adjust_rotation(axis, direction)

    def stop_rotate(self):
        if hasattr(self, 'rotate_timer'):
            self.rotate_timer.stop()
            del self.rotate_timer

    def adjust_rotation(self, axis, direction):
        global Euler, lock
        step = 0.1 * direction
        with lock:
            Euler[axis] += step
            if Euler[axis] > 180:
                Euler[axis] -= 360
            elif Euler[axis] < -180:
                Euler[axis] += 360
            self.camera_thread.states['rxyz'][['RX','RY','RZ'][axis]] = Euler[axis]
            if axis == 0:
                self.lbl_rx.setText(f"{Euler[axis]:.1f}")
            elif axis == 1:
                self.lbl_ry.setText(f"{Euler[axis]:.1f}")
            elif axis == 2:
                self.lbl_rz.setText(f"{Euler[axis]:.1f}")

    def update_egm_state(self, state):
        self.lbl_egm_state.setText(f"EGM State: {state}")

    def reset_xyz_position(self):
        global Pos, lock
        reset_values = [500.0, 0.0, 600.0]
```

```
        with lock:
            Pos = reset_values.copy()
            self.camera_thread.states['xyz'] = {'X': 500, 'Y': 0, 'Z': 600}
        self.lbl_x.setText(f"{reset_values[0]:.1f}")
        self.lbl_y.setText(f"{reset_values[1]:.1f}")
        self.lbl_z.setText(f"{reset_values[2]:.1f}")

    def reset_rxyz_rotation(self):
        global Euler, lock
        reset_rxyz_values = [180.0, 0.0, 180.0]

        with lock:
            Euler = reset_rxyz_values.copy()
            self.camera_thread.states['rxyz'] = {'RX': 180, 'RY': 0, 'RZ': 180}
        self.lbl_x.setText(f"{reset_rxyz_values[0]:.1f}")
        self.lbl_y.setText(f"{reset_rxyz_values[1]:.1f}")
        self.lbl_z.setText(f"{reset_rxyz_values[2]:.1f}")

# ==================== RWS Functions ====================
    def toggle_rws_connection(self):
        if not self.rws_connected:
            try:
                self.rws = RwsClient(
                    self.rws_ip.text(),
                    port=int(self.rws_port.text()),
                    username=self.rws_user.text(),
                    password=self.rws_pass.text()
                )
                # Test connection
                identity = self.rws.get_identity()
                if "error" in identity:
                    raise Exception(identity["error"])

                self.rws_connected = True
                self.btn_rws_connect.setText("Disconnect RWS")

                QMessageBox.information(self, "Success", "Connected to RWS!")
                self.add_log("Connected to RWS!")
```

```
            print("Connected to RWS!")
            self.update_motor_status()
            self.update_rapid_status()
            self.update_opmode()
            self.update_IO_status()
            self.show_identity()
            self.rws.register_user_local()
            self.update_speed()

            # Start subscription for real-time updates
            if self.rws.subscribe(self):
                print("Subscribed to RWS events")
            else:
                QMessageBox.warning(self, "Warning", "Failed to subscribe to events")

        except Exception as e:
            QMessageBox.critical(self, "Error", f"Connection failed: {str(e)}")
            print("Connection failed!")
    else:
        # Close WebSocket connection
        if self.rws and self.rws.ws_client:
            self.rws.ws_client.close()
            if self.rws.subscription_thread:
                self.rws.subscription_thread.join(timeout=1.0)

        self.rws = None
        self.rws_connected = False
        print("Disconnected to RWS!")
        self.add_log("Disconnected to RWS!")
        self.btn_rws_connect.setText("Connect RWS")
        self.btn_rws_connect.setStyleSheet("")
        self.lbl_motor_status.setText("Motor Status: Disconnected")
        self.lbl_rapid_status.setText("Rapid Status: Disconnected")
        self.lbl_IO_status.setText("Gripper Status: Disconnected")
        self.lbl_opmode.setText("Operation Mode: Disconnected")

def process_rws_event(self, event_xml):
    namespace = {'xhtml': 'http://www.w3.org/1999/xhtml'}
    try:
```

```
            root = ET.fromstring(event_xml)

            # Process speed ratio
            speed_elem = root.find(".//xhtml:li[@class='pnl-speedratio-ev']/xhtml:span",
namespace)
            if speed_elem is not None and speed_elem.text:
                speed_str = speed_elem.text.strip()
                speed = int(speed_str)
                self.speed_changed.emit(speed)

            # Process opmode
            opmode_elem = root.find(".//xhtml:li[@class='pnl-opmode-ev']/xhtml:span",
namespace)
            if opmode_elem is not None and opmode_elem.text:
                mode = opmode_elem.text.lower()
                self.opmode_changed.emit(mode)

            # Process motor state
            ctrl_elem = root.find(".//xhtml:li[@class='pnl-ctrlstate-ev']/xhtml:span", namespace)
            if ctrl_elem is not None and ctrl_elem.text:
                state = ctrl_elem.text.lower()
                if 'on' in state:
                    self.motor_state_changed.emit(True)
                elif 'off' in state:
                    self.motor_state_changed.emit(False)

        except Exception as e:
            print(f"Error processing RWS event: {e}")

    def update_speed_ui(self, speed):
        self.slider.setValue(speed)
        self.lbl_speed.setText(f"{speed}%")

    def update_opmode_ui(self, mode):
        if "auto" in mode:
            self.lbl_opmode.setText("Operation Mode: AUTO")
            self.lbl_opmode.setStyleSheet("color: green; font-weight: bold;")
        elif "man" in mode:
            self.lbl_opmode.setText("Operation Mode: MANUAL")
```

```
        self.lbl_opmode.setStyleSheet("color: red; font-weight: bold;")

    def update_motor_ui(self, state):
        if state:
            self.lbl_motor_status.setText("Motor Status: ON")
            self.lbl_motor_status.setStyleSheet("color: green; font-weight: bold;")
        else:
            self.lbl_motor_status.setText("Motor Status: OFF")
            self.lbl_motor_status.setStyleSheet("color: red; font-weight: bold;")

    def mode_auto(self):
        if self.rws_connected:
            code = self.rws.set_opmode("auto")
            if code == 202:
                QMessageBox.information(self, "Succesfully","Change Mode Success!")
                self.add_log("Auto mode selected!")
                self.rws.opmode_ack()
            else:
                QMessageBox.critical(self, "Error", f"Failed to changes mode: {code}")

    def mode_man(self):
        if self.rws_connected:
            code = self.rws.set_opmode("man")
            if code == 202:
                QMessageBox.information(self, "Succesfully","Change Mode Success!")
                self.add_log("Manual mode selected!")
            else:
                QMessageBox.critical(self, "Error", f"Failed to changes mode: {code}")

    def motor_on(self):
        if self.rws_connected:
            code = self.rws.set_ctrl_state("motoron")
            if code == 204:
                self.update_motor_status()
                self.add_log("Motor On state!")
            else:
                QMessageBox.critical(self, "Error", f"Failed to start motors: {code}")

    def motor_off(self):
```

```
        if self.rws_connected:
            code = self.rws.set_ctrl_state("motoroff")
            if code == 204:
                self.update_motor_status()
                self.add_log("Motor Off state!")
            else:
                QMessageBox.critical(self, "Error", f"Failed to stop motors: {code}")


    def set_speed(self, value):
        self.lbl_speed.setText(f"{value}%")
        if self.rws_connected:
            code = self.rws.set_speed(value)
            self.add_log("Speed Adjusted!")
            if code != 204:
                QMessageBox.warning(self, "Warning", f"Failed to set speed: {code}")

    def update_speed(self):
        if self.rws_connected:
            speed_1 = self.rws.get_speed()
            self.slider.setValue(speed_1)
            self.lbl_speed.setText(f"{speed_1}%")

    def ON_IO(self,lvalue):
        if self.rws_connected:
            code = self.rws.set_IO("1")
            if code == 204:
                self.rws.set_IO(lvalue)
                self.add_log("Gripper ON!")
            else:
                QMessageBox.critical(self, "Error", f"Failed to changes IO values: {code}")

    def OFF_IO(self,lvalue):
        if self.rws_connected:
            code = self.rws.set_IO("0")
            if code == 204:
                self.rws.set_IO(lvalue)
                self.add_log("Gripper OFF!")
            else:
```

```
            QMessageBox.critical(self, "Error", f"Failed to changes IO values: {code}")

    def show_identity(self):
        if self.rws_connected:
            identity_info = self.rws.get_identity()
            if "error" in identity_info:
                self.lbl_identity.setText(f"Error: {identity_info['error']}")
                return
            display_text = f"Name: {identity_info['name']}\nType: {identity_info['type']}"
            self.lbl_identity.setText(display_text)

    def start_rapid(self):
        if self.rws_connected:
            code = self.rws.start_rapid()
            if code == 204:
                QMessageBox.information(self, "Success", "RAPID started!")
                self.add_log("RAPID program started!")

            else:
                QMessageBox.critical(self, "Error", f"Failed to start RAPID: {code}")

    def stop_rapid(self):
        if self.rws_connected:
            code = self.rws.stop_rapid()
            if code == 204:
                QMessageBox.information(self, "Success", "RAPID stopped!")
                self.add_log("RAPID program stopped!")

            else:
                QMessageBox.critical(self, "Error", f"Failed to stop RAPID: {code}")

    def reset_rapid(self):
        if self.rws_connected:
            code = self.rws.reset_rapid()
            if code == 204:
                QMessageBox.information(self, "Success", "RAPID reseted!")
                self.add_log("RAPID program has been reset!")
            else:
                QMessageBox.critical(self, "Error", f"Failed to reset PP: {code}")
```

```
def update_rapid_status(self):
    if self.rws_connected:
        state = self.rws.get_rapid_state()
        if "running" in state:
            self.lbl_rapid_status.setText("Rapid Status: Running")
            self.lbl_rapid_status.setStyleSheet("color: green; font-weight: bold;")
        elif "stopped" in state:
            self.lbl_rapid_status.setText("Rapid Status: Stopped")
            self.lbl_rapid_status.setStyleSheet("color: red; font-weight: bold;")
        else:
            self.lbl_rapid_status.setText(f"Rapid Status: {state}")
            self.lbl_rapid_status.setStyleSheet("color: orange; font-weight: bold;")

def update_motor_status(self):
    if self.rws_connected:
        state = self.rws.get_ctrl_state()
        if "motoron" in state:
            self.lbl_motor_status.setText("Motor Status: ON")
            self.lbl_motor_status.setStyleSheet("color: green; font-weight: bold;")
        elif "motoroff" in state:
            self.lbl_motor_status.setText("Motor Status: OFF")
            self.lbl_motor_status.setStyleSheet("color: red; font-weight: bold;")
        else:
            self.lbl_motor_status.setText(f"Motor Status: {state}")
            self.lbl_motor_status.setStyleSheet("color: orange; font-weight: bold;")

def update_opmode(self):
    if self.rws_connected:
        state = self.rws.get_opmde()
        if "auto" in state:
            self.lbl_opmode.setText("Operation Mode: AUTO")
            self.lbl_opmode.setStyleSheet("color: green; font-weight: bold;")
        elif "manr" in state:
            self.lbl_opmode.setText("Operation Mode: MANUAL")
            self.lbl_opmode.setStyleSheet("color: red; font-weight: bold;")
        else:
            self.lbl_opmode.setText(f"Operation Mode: {state}")
```

```
            self.lbl_opmode.setStyleSheet("color: orange; font-weight: bold;")

    def update_IO_status(self):
        if self.rws_connected:
            state = self.rws.get_IO_state()
            if "1" in state:
                self.lbl_IO_status.setText("Gripper Status: ON")
                self.add_log("Gipper ON!")
                self.lbl_IO_status.setStyleSheet("color: green; font-weight: bold;")
            elif "0" in state:
                self.lbl_IO_status.setText("Gripper: OFF")
                self.add_log("Gripper OFF!")
                self.lbl_IO_status.setStyleSheet("color: red; font-weight: bold;")
            else:
                self.lbl_IO_status.setText(f"Gripper Status: {state}")
                self.lbl_IO_status.setStyleSheet("color: orange; font-weight: bold;")

#==================Camera Function================
    def toggle_camera(self):
        if self.camera_thread.isRunning():
            self.camera_thread.running = False
            self.camera_thread.quit()
            self.btn_camera.setText("Start Camera")
            self.add_log("Stopped Camera!")
            self.camera_label.clear()
        else:
            camera_index = self.camera_combo.currentData()
            self.camera_thread.camera_index = camera_index
            self.camera_thread.running = True
            self.camera_thread.start()
            self.btn_camera.setText("Stop Camera")
            self.add_log("Started Camera!")

    def toggle_hand_tracking(self):
        self.camera_thread.process_hands = not self.camera_thread.process_hands
        self.btn_hand_tracking.setText(
            "Disable Hand Tracking" if self.camera_thread.process_hands
            else "Enable Hand Tracking"
        )
```

```
def update_camera_preview(self, image, landmarks):
    try:
        pixmap = QPixmap.fromImage(image)
        if not pixmap.isNull():
            self.camera_label.setPixmap(
                pixmap.scaled(self.camera_label.width(),
                              self.camera_label.height(),
                              Qt.KeepAspectRatio,
                              Qt.SmoothTransformation)
            )
    except Exception as e:
        print("Error updating preview:", e)

def handle_camera_button(self, command):
    if command[1] in ['X', 'Y', 'Z']:
        axis_map = {'X': 0, 'Y': 1, 'Z': 2}
        direction = 1 if '+' in command else -1
        axis = axis_map[command[1]]
        self.adjust_position(axis, direction)

    elif command.startswith(('+R', '-R')):
        axis_map = {'RX': 0, 'RY': 1, 'RZ': 2}
        rotation_axis = command[1:]
        direction = 1 if command[0] == '+' else -1

        if rotation_axis in axis_map:
            axis = axis_map[rotation_axis]
            self.adjust_rotation(axis, direction)

def handle_gripper_change(self, state):
    if self.rws_connected:
        lvalue = "1" if state else "0"
        code = self.rws.set_IO(lvalue)
        if code == 204:
            self.update_IO_status()
        else:
            self.add_log("Failed to update gripper via camera: {code}")
```

```
    def toggle_hand_control(self):
        self.camera_thread.states['hand_mode'] = not self.camera_thread.states['hand_mode']
        self.add_log("Started Hand Mode!")

#=======================System Log=================================
    def add_log(self, message, level="INFO", source="ABB CRB1500"):
        now = datetime.now().strftime('%Y-%m-%d %H:%M:%S,%f')[:-3]
        log_line = f"{now} - {source} - {level} - {message}"
        self.log_output.append(log_line)
    def clear_log(self):
        self.log_output.clear()
        self.add_log("[INFO] Log cleared by user.")
#==================================================================
    def closeEvent(self, event):
        if self.recv_thread:
            self.recv_thread.running = False
            self.recv_thread.wait()
        if self.send_thread:
            self.send_thread.running = False
            self.send_thread.wait()
        if self.robot_socket:
            self.robot_socket.close()
        if self.camera_thread.isRunning():
            self.camera_thread.running = False
            self.camera_thread.wait()
        if self.rws_connected and self.rws and self.rws.ws_client:
            self.rws.ws_client.close()
            if self.rws.subscription_thread:
                self.rws.subscription_thread.join(timeout=1.0)
        if hasattr(self, 'chart_timer'):
            self.chart_timer.stop()
        event.accept()

if __name__ == "__main__":
    app = QApplication(sys.argv)
    win = MainWindow()
    win.resize(1500, 750)
    win.show()
    sys.exit(app.exec_())
```