

TRƯỜNG ĐẠI HỌC BÁCH KHOA TP.HỒ CHÍ MINH  
KHOA ĐIỆN - ĐIỆN TỬ  
BỘ MÔN ĐIỀU KHIỂN TỰ ĐỘNG

LÊ BÁ THÀNH ĐẠT - 1510675

LUẬN VĂN TỐT NGHIỆP  
THIẾT KẾ HỆ THỐNG ĐIỀU KHIỂN ROBOT SCARA  
DESIGN **CONTROLING** SYSTEM FOR SCARA ROBOT

KỸ SƯ NGÀNH KỸ THUẬT ĐIỀU KHIỂN VÀ TỰ ĐỘNG HÓA

GIẢNG VIÊN HƯỚNG DẪN  
TS.NGUYỄN HOÀNG GIÁP

TP.HỒ CHÍ MINH, 2019

## LỜI CẢM ƠN

Lời đầu tiên, em xin gửi lời cảm ơn chân thành đối với quý thầy, cô trưởng đại học Bách Khoa nói chung và bộ môn điều khiển và tự động hóa nói riêng. Mọi người đã truyền đạt những kiến thức quý báu để em có thể hoàn thành đề tài luận văn này.

Em xin gửi lời cảm ơn sâu sắc đến thầy Nguyễn Hoàng Giáp. Thầy đã hướng dẫn và chỉ dạy em tận tình để em có thể hoàn thành tốt đề tài luận văn của mình. Đối với em đó là sự may mắn rất lớn khi có thầy hướng dẫn, chỉ bảo trong những ngày tháng qua.

Xin cảm ơn công ty Ajinextek đã tài trợ trang thiết bị để cho em có thể hoàn thành được luận văn, cũng như là tiếp cận thêm với thiết bị thực tế trong công nghiệp.

Cảm ơn quý bạn bè đã giúp đỡ em trong quá trình học tập và thực hiện đề tài luận văn. Cuối cùng, em xin gửi lời cảm ơn lần nữa tới quý thầy, cô, công ty và bạn bè. Mọi người đã giảng dạy, hướng dẫn và đóng góp ý kiến để em có thể tiến bộ và hoàn thiện hơn.

TPHCM, ngày 7 tháng 6 năm 2019

Sinh viên

Lê Bá Thành Đạt

# Mục lục

<b>1 GIỚI THIỆU ĐỀ TÀI</b>	<b>2</b>
1.1 Mục đích . . . . .	2
1.2 Mục tiêu của luận văn . . . . .	2
1.3 Đối tượng và phạm vi nghiên cứu . . . . .	2
1.3.1 Đối tượng nghiên cứu . . . . .	2
1.3.2 Phạm vi nghiên cứu . . . . .	3
1.4 Nhiệm vụ và nội dung của Luận văn . . . . .	3
1.5 Cấu trúc luận văn . . . . .	3
<b>2 TỔNG QUAN VỀ ROBOT CÔNG NGHIỆP</b>	<b>5</b>
2.1 Khái niệm về robot công nghiệp . . . . .	5
2.2 Sự phát triển của robot công nghiệp . . . . .	5
2.3 Các thành phần chính của một hệ thống robot công nghiệp . . . . .	8
2.4 ĐỘNG HỌC ROBOT . . . . .	8
2.4.1 Một số khái niệm . . . . .	8
2.4.2 Các phép biến đổi hệ tọa độ . . . . .	9
2.4.3 Phương pháp D-H . . . . .	11
2.4.4 Phương trình động học thuận của robot SCARA . . . . .	13
2.4.5 Ma trận Jacobian . . . . .	15
2.4.6 Singularities . . . . .	17
2.4.7 Phương trình động học ngược robot SCARA . . . . .	19
2.5 Hoạch định quỹ đạo robot . . . . .	21
2.5.1 Tổng quan về quá trình hoạch định quỹ đạo trong không gian làm việc . . . . .	21
2.5.2 Hoạch định quỹ đạo trong miền thời gian . . . . .	22
2.5.3 Hoạch định quỹ đạo trong miền không gian . . . . .	26
2.5.4 Giải bài toán động học ngược . . . . .	30
<b>3 TỔNG QUAN VỀ CẤU TRÚC HỆ THỐNG</b>	<b>31</b>
<b>4 THIẾT KẾ PHẦN CỨNG VÀ TỦ ĐIỀU KHIỂN</b>	<b>33</b>
4.1 Giới thiệu về các chi tiết phần cứng . . . . .	33
4.1.1 Robot SCARA . . . . .	33
4.1.2 Board điều khiển . . . . .	34
4.1.3 Servo Driver . . . . .	35

4.1.4	Relay . . . . .	37
4.1.5	Van Solenoid . . . . .	37
4.2	Thiết kế tủ điện điều khiển . . . . .	38
4.2.1	Hệ thống nguồn . . . . .	38
4.2.2	Tủ điện sau khi được thiết kế . . . . .	40
<b>5</b>	<b>THIẾT KẾ PHẦN MỀM ĐIỀU KHIỂN VÀ MÔ PHỎNG ROBOT SCARA</b>	<b>41</b>
5.1	Phần mềm trên board điều khiển . . . . .	41
5.1.1	Giới thiệu về phần mềm Keil C . . . . .	41
5.1.2	Chương trình ngắt timer 2 . . . . .	42
5.1.3	Chương trình ngắt timer 3 . . . . .	43
5.1.4	Chương trình ngắt timer 5 . . . . .	44
5.1.5	Chương trình ngắt truyền thông USART3 . . . . .	44
5.1.6	Lập trình thuật toán hoạch định quỹ đạo trên board điều khiển	45
5.2	Phần mềm trên máy tính . . . . .	50
5.2.1	Giới thiệu phần mềm Visual Studio 2008 và thư viện MFC . . . . .	50
5.2.2	Giới thiệu về phần mềm đã thiết kế . . . . .	51
5.3	Giới thiệu về OpenGL . . . . .	52
5.3.1	Giới thiệu chung . . . . .	52
5.3.2	Xây dựng tính năng đồ họa sử dụng OpenGL . . . . .	54
5.4	Chức năng mô phỏng của phần mềm . . . . .	59
5.4.1	Khởi tạo môi trường OpenGL trong phần mềm . . . . .	60
5.4.2	Thiết kế các chi tiết của robot bằng phần mềm SolidWork . . . . .	63
5.4.3	Xây dựng module đọc tập tin định dạng STL . . . . .	66
5.4.4	Mô hình hóa robot SCARA bằng OpenGL . . . . .	69
5.4.5	Tạo chuyển động cho robot . . . . .	71
5.5	Giao diện mô phỏng . . . . .	72
5.5.1	Sơ đồ khối phần mô phỏng . . . . .	72
5.5.2	Giao diện Option . . . . .	72
5.5.3	Giao diện Jog . . . . .	73
5.5.4	Giao diện Point . . . . .	75
5.5.5	Giao diện Execute . . . . .	76
5.5.6	Giao diện Graph . . . . .	77
5.5.7	Giao diện PTP . . . . .	78
5.6	Chức năng điều khiển của phần mềm . . . . .	78
5.6.1	General Tab . . . . .	80
5.6.2	Setting Tab . . . . .	80
5.6.3	New Tab . . . . .	81
5.6.4	Program Tab . . . . .	83
5.6.5	Axis Tab . . . . .	86
5.6.6	History Tab . . . . .	87
5.6.7	Up-Download Tab . . . . .	88
5.7	Phương thức truyền thông giữa board điều khiển và máy tính . . . . .	89
5.7.1	Chuẩn giao tiếp RS232C . . . . .	89

5.7.2	Khung truyền và nhận dữ liệu ở máy tính khi giao tiếp với board điều khiển . . . . .	90
5.7.3	Giao tiếp giữa board điều khiển và máy tính . . . . .	94
<b>6</b>	<b>KẾT QUẢ VÀ HƯỚNG PHÁT TRIỂN</b>	<b>97</b>
6.1	Kết quả . . . . .	97
6.2	Hướng phát triển . . . . .	111

# Danh sách hình vẽ

2.1	Điện toán đám mây mở ra một kỷ nguyên mới cho robot . . . . .	7
2.2	Hình ảnh robot bên trong một nhà máy . . . . .	7
2.3	Sơ đồ một hệ thống robot công nghiệp . . . . .	8
2.4	Tọa độ suy rộng của robot . . . . .	9
2.5	Quy tắc bàn tay phải . . . . .	9
2.6	Phép tịnh tiến . . . . .	10
2.7	Quay u quanh trục z $90^\circ$ rồi quay quanh trục y $90^\circ$ . . . . .	11
2.8	Các thông số Denavit-Hartenberg . . . . .	12
2.9	Hệ trục tọa độ gắn trên khâu chấp hành . . . . .	13
2.10	Hệ trục tọa độ cho robot SCARA . . . . .	14
2.11	Robot SCARA trong mặt phẳng Oxy . . . . .	20
2.12	Các bước hoạch định quỹ đạo Robot . . . . .	22
2.13	Quá trình hoạch định quỹ đạo trong miền thời gian . . . . .	23
2.14	Đồ thị s, v, a trên quỹ đạo robot . . . . .	24
2.15	Đồ thị s, v, a trên quỹ đạo robot . . . . .	25
2.16	Quá trình hoạch định quỹ đạo trong miền không gian . . . . .	27
2.17	Các bước quá trình Inverse Kinematic . . . . .	30
3.1	Cấu trúc hệ thống điều khiển robot SCARA . . . . .	31
4.1	Robot SCARA . . . . .	33
4.2	Board điều khiển . . . . .	34
4.3	Terminal kết nối board điều khiển và Servo Driver . . . . .	35
4.4	Servo Driver điều khiển 4 trục robot . . . . .	36
4.5	Module relay điều khiển van solenoid . . . . .	37
4.6	Van solenoid . . . . .	38
4.7	CB cấp nguồn cho hệ thống . . . . .	38
4.8	Contactor . . . . .	39
4.9	Bộ lọc nhiễu nguồn . . . . .	39
4.10	Nguồn 24VDC cấp cho board điều khiển . . . . .	40
4.11	Tủ điện điều khiển . . . . .	40
5.1	Phần mềm Keil C uVision 5 . . . . .	41
5.2	Chương trình thực thi trong ngắt Timer 2 của chip STM . . . . .	42
5.3	Chương trình thực thi trong ngắt Timer 3 của chip STM . . . . .	43
5.4	Chương trình thực thi trong ngắt Timer 5 của chip STM . . . . .	44
5.5	Chương trình thực thi trong ngắt truyền thông của chip STM . . . . .	45

5.6	Tính toán các giá trị thời gian của profile chuyển động . . . . .	46
5.7	Hoạch định quỹ đạo trong miền thời gian . . . . .	47
5.8	Hoạch định quỹ đạo đường thẳng . . . . .	48
5.9	Hoạch định quỹ đạo đường thẳng . . . . .	49
5.10	Gải bài toán động học ngược . . . . .	50
5.11	Sơ đồ khối của chương trình . . . . .	52
5.12	OpenGL . . . . .	53
5.13	Cơ chế hoạt động của OpenGL . . . . .	53
5.14	Quá trình xây dựng tính năng đồ họa . . . . .	54
5.15	Phép chiếu trực giao . . . . .	57
5.16	Phép chiếu phối cảnh . . . . .	58
5.17	Hệ tọa độ chuẩn trong OpenGL . . . . .	59
5.18	Các phép biến đổi trong OpenGL . . . . .	59
5.19	Quá trình mô hình hóa các chi tiết robot trong OpenGL . . . . .	60
5.20	Khâu 1 robot SCARA . . . . .	63
5.21	Khâu 2 robot SCARA . . . . .	63
5.22	Khâu 3 robot SCARA . . . . .	64
5.23	Khâu 4 robot SCARA . . . . .	64
5.24	Tay kẹp robot SCARA . . . . .	65
5.25	Thiết lập tùy chọn cho file STL . . . . .	65
5.26	Cấu trúc file định dạng STL . . . . .	66
5.27	File ROBOT.txt chứa tên file STL . . . . .	66
5.28	Sơ đồ giải thuật đọc file STL . . . . .	67
5.29	Mô hình robot SCARA xây dựng bằng OpenGL . . . . .	71
5.30	Sơ đồ khối phần mô phỏng . . . . .	72
5.31	Giao diện Option . . . . .	73
5.32	Giao diện Jog . . . . .	74
5.33	Giao diện Point . . . . .	76
5.34	Giao diện thực thi chương trình . . . . .	77
5.35	Giao diện graph . . . . .	77
5.36	Giao diện PTP . . . . .	78
5.37	Giao diện chính của chương trình . . . . .	79
5.38	Giao diện cài đặt Setting . . . . .	81
5.39	Sơ đồ tổ chức của New Tab . . . . .	82
5.40	Giao diện tạo project mới và các file trong một project . . . . .	82
5.41	Hộp thoại chỉ dẫn vị trí lưu file . . . . .	82
5.42	Giao diện Program Tab . . . . .	83
5.43	Giao diện Command View . . . . .	83
5.44	Giao diện Point View . . . . .	84
5.45	Giao diện File View . . . . .	84
5.46	Giao diện Teaching . . . . .	84
5.47	Giao diện Result . . . . .	85
5.48	Quá trình biên dịch chương trình robot . . . . .	85
5.49	Giao diện trực 1 . . . . .	86
5.50	Giao diện History . . . . .	88

5.51	Sơ đồ tổ chức và chức năng của Upload và Download Tab . . . . .	88
5.52	Giao diện Upload và Download . . . . .	89
5.53	Khung dữ liệu truyền thông . . . . .	90
5.54	Phương thức giao tiếp giữa board và máy tính . . . . .	94
5.55	Sơ đồ thực thi của Thread WatchCom . . . . .	95
5.56	Chương trình phục vụ ngắt Timer trên máy tính . . . . .	96
6.1	Dồ thị biểu diễn gia tốc end-effector trên quỹ đạo . . . . .	97
6.2	Dồ thị biểu diễn vận tốc end-effector trên quỹ đạo . . . . .	98
6.3	Dồ thị biểu diễn độ dời end-effector trên quỹ đạo . . . . .	98
6.4	Dồ thị biểu diễn gia tốc end-effector trên mỗi trục tọa độ . . . . .	99
6.5	Dồ thị biểu diễn vận tốc end-effector trên mỗi trục tọa độ . . . . .	99
6.6	Dồ thị biểu diễn độ dời end-effector trên mỗi trục tọa độ . . . . .	99
6.7	Dồ thị biểu diễn vị trí end-effector trên mỗi trục tọa độ . . . . .	100
6.8	Giá trị biến khớp 1 và 2 . . . . .	100
6.9	Dồ thị biểu diễn giá trị khớp 3 . . . . .	101
6.10	Dồ thị biểu diễn vận tốc biến khớp 1 và khớp 2 . . . . .	101
6.11	Dồ thị biểu diễn vận tốc biến khớp 3 . . . . .	102
6.12	<b>Dường</b> đi trong không gian . . . . .	102
6.13	Dồ thị biểu diễn gia tốc end-effector trên quỹ đạo . . . . .	102
6.14	Dồ thị biểu diễn vận tốc end-effector trên quỹ đạo . . . . .	103
6.15	Dồ thị biểu diễn độ dời end-effector trên quỹ đạo . . . . .	103
6.16	Dồ thị biểu diễn gia tốc end-effector trên mỗi trục tọa độ . . . . .	104
6.17	Dồ thị biểu diễn vận tốc end-effector trên mỗi trục tọa độ . . . . .	104
6.18	Dồ thị biểu diễn độ dời end-effector trên mỗi trục tọa độ . . . . .	104
6.19	Dồ thị biểu diễn vị trí end-effector trên mỗi trục tọa độ . . . . .	105
6.20	Dồ thị biểu diễn giá trị khớp 3 . . . . .	105
6.21	Dồ thị biểu diễn vận tốc biến khớp 1 và khớp 2 . . . . .	106
6.22	Dồ thị biểu diễn vận tốc biến khớp 3 . . . . .	106
6.23	<b>Dường</b> đi trong không gian . . . . .	106
6.24	Dồ thị biểu diễn gia tốc end-effector trên quỹ đạo . . . . .	107
6.25	Dồ thị biểu diễn vận tốc end-effector trên quỹ đạo . . . . .	107
6.26	Dồ thị biểu diễn độ dời end-effector trên quỹ đạo . . . . .	108
6.27	Dồ thị biểu diễn vị trí end-effector trên mỗi trục tọa độ . . . . .	108
6.28	Dồ thị biểu diễn giá trị khớp 1 và khớp 2 . . . . .	108
6.29	Dồ thị biểu diễn giá trị khớp 3 . . . . .	109
6.30	Dồ thị biểu diễn vận tốc biến khớp 1 và khớp 2 . . . . .	109
6.31	Dồ thị biểu diễn vận tốc biến khớp 3 . . . . .	110
6.32	<b>Dường</b> đi trong không gian . . . . .	110

# Danh sách bảng

2.1	Bảng thông số DH robot SCARA . . . . .	14
2.2	Công thức tính giá trị phần tử ma trận Jacobian . . . . .	16
4.1	Bảng thông số động cơ của Robot . . . . .	34
4.2	Thông số kỹ thuật của board điều khiển . . . . .	35

# Chương 1

## GIỚI THIỆU ĐỀ TÀI

### 1.1 Mục đích

Thiết kế một hệ thống điều khiển robot SCARA hoàn chỉnh từ những thiết bị trong công nghiệp như: Board điều khiển xuất xung, Servo Driver, Robot SCARA Ngoài ra, để người người có thể tương tác với máy tính cũng như học cách sử dụng robot, một phần mềm được thiết kế với chức năng điều khiển robot cũng như tích hợp tính năng mô phỏng.

### 1.2 Mục tiêu của luận văn

Luận văn khi hoàn thành sẽ phải đáp ứng được các mục tiêu sau đây:

- Hệ thống được thiết kế có khả năng điều khiển robot SCARA hoạt động một cách chính xác và ổn định trong môi trường công nghiệp.
- Thiết kế phần mềm với tính năng mô phỏng và điều khiển thực tế.
- Lập trình robot cho một ứng dụng cụ thể trong công nghiệp.

### 1.3 Đối tượng và phạm vi nghiên cứu

#### 1.3.1 Đối tượng nghiên cứu

- Robot SCARA.
- Các phương trình động học của Robot SCARA.
- Lý thuyết về hoạch định quỹ đạo robot.
- Phương thức mô phỏng robot.
- Các thiết bị điều khiển robot trong công nghiệp như board xuất xung, Servo Driver.
- Giao thức truyền thông giữa máy tính và board điều khiển.

### 1.3.2 Phạm vi nghiên cứu

Luận văn tập trung vào việc xây dựng hệ thống điều khiển robot bao gồm phần cứng và phần mềm. Trong quá trình thực hiện Luận văn, cần tìm hiểu nhiều lĩnh vực khác nhau nhằm đáp ứng được yêu cầu Luận văn. Phạm vi nghiên cứu của luận văn bao gồm nhiều mảng kiến thức:

- Tìm hiểu và thiết kế mô hình bằng phần mềm vẽ cơ khí SolidWorks.
- Xây dựng phương trình động học robot.
- Xây dựng giải thuật hoạch định quỹ đạo robot.
- Cách thức hoạt động và cách kết hợp nhiều thiết bị công nghiệp để tạo thành hệ thống phần cứng hoàn chỉnh.
- Kỹ thuật mô phỏng robot.
- Giao thức truyền thông giữa board điều khiển và máy tính.
- Thiết kế phần mềm giao tiếp với bộ điều khiển, mô phỏng mô hình robot bằng ngôn ngữ C++.

## 1.4 Nhiệm vụ và nội dung của Luận văn

Để thực hiện được luận văn, cần thực hiện các nhiệm vụ quan trọng:

- Thiết kế mô hình robot bằng SolidWorks đúng với các thông số thật, sau đó xây dựng mô hình bằng thư viện OpenGL
- Thiết kế tủ điện.
- Tìm hiểu cách thức giao tiếp giữa board điều khiển và máy tính.
- Thiết kế chức năng mô phỏng của phần mềm
- Thiết kế phần mềm hoạt động hiệu quả, có tính thẩm mỹ, đáp ứng được các mục tiêu đã đề ra.

Nội dung cơ bản của Luận văn là xây dựng hệ thống điều khiển robot từ những thiết bị trong công nghiệp, xây dựng phần mềm để giao tiếp với board điều khiển. Phần mềm có chức năng điều khiển và mô phỏng hoạt động robot.

## 1.5 Cấu trúc luận văn

Cấu trúc luận văn được trình bày gồm:

- Chương 1: Giới thiệu đề tài - trình bày khái quát về mục đích, đối tượng và phạm vi nghiên cứu của luận văn.

- Chương 2: Giới thiệu tổng quan về robot công nghiệp - trình bày tổng quan về robot công nghiệp, các khái niệm liên quan đến robot và lí thuyết về hoạch định quỹ đạo.
- Chương 3: Tổng quan hệ thống - trình bày cấu trúc tổng quan của hệ thống được thiết kế trong luận văn.
- Chương 4: Thiết kế phần cứng - trình bày về các chi tiết phần cứng và hệ thống phần cứng xây dựng.
- Chương 5: Thiết kế phần mềm - trình phần mềm được xây dựng trên board điều khiển và máy tính.
- Chương 6: Kết quả và hướng phát triển - trình bày kết quả thu được và định hướng luận văn.

## Chương 2

# TỔNG QUAN VỀ ROBOT CÔNG NGHIỆP

### 2.1 Khái niệm về robot công nghiệp

Robot công nghiệp có thể được định nghĩa theo một số tiêu chuẩn sau:

- Theo tiêu chuẩn AFNOR của Pháp: Robot công nghiệp là một cơ cấu chuyển động có thể lập trình, lặp lại các chương trình, tổng hợp các chương trình đặt ra trên các trục tọa độ, còn có khả năng định vị, định hướng, di chuyển các đối tượng vật chất như chi tiết, dụng cụ, gắp lắp theo những hành trình thay đổi đã được chương trình hóa nhằm thực hiện các nhiệm vụ công nghệ khác nhau.
- Theo tiêu chuẩn RIA của Mỹ (Robot Institute of America): Robot là một tay máy vận năng có thể lặp lại các chương trình, được thiết kế để di chuyển vật liệu, chi tiết, dụng cụ, hoặc các thiết bị chuyên dụng thông qua các chương trình chuyển động có thể thay đổi để hoàn thành các nhiệm vụ khác nhau.
- Theo tiêu chuẩn TOCT 25686-85 của Nga: Robot công nghiệp là một tay máy tự động, được đặt cố định hoặc di động được, liên kết giữa một tay máy và một hệ thống điều khiển theo chương trình, có thể lặp lại để hoàn thành các chức năng vận động và điều khiển trong quá trình sản xuất.

Do đó, robot công nghiệp có thể được hiểu là những thiết bị tự động, linh hoạt được lập trình để thực hiện những chức năng lao động công nghiệp của con người với hiệu suất cao.

### 2.2 Sự phát triển của robot công nghiệp

Năm 1921, thuật ngữ "Robot" ra đời, xuất phát từ tiếng Séc(Czech) "Robota". Năm 1954, George Charles Devol đã phát minh ra robot công nghiệp đầu tiên.

Cánh tay robot đầu tiên được sản xuất vào năm 1961. Những cánh tay robot công nghiệp tiếp tục được phát triển trong những năm 1960 và 1970 trên khắp thế

giới. Sự cạnh tranh từ các công ty trên toàn thế giới tiếp tục tạo ra nhu cầu cao cho robot công nghiệp làm thúc đẩy ngành công nghiệp robot phát triển hơn. Trong những năm gần đây, khoa học kỹ thuật ngày càng phát triển theo đó là sự phát triển về số lượng và chất lượng. Chính vì vậy, robot có thêm nhiều tính năng mới và giá thành giảm nên được ứng dụng rộng rãi trong sản xuất.

Robot đã có những bước tiến đáng kể hơn nửa thế kỷ qua. Robot đầu tiên được ứng dụng trong công nghiệp vào những năm 60 để thay thế con người làm các công việc nặng nhọc, nguy hiểm trong môi trường độc hại. Do nhu cầu cần ăn nhập ngày càng nhiều với quá trình sản xuất phức tạp nên robot công nghiệp cần có những khả năng thích ứng linh hoạt và thông minh hơn. Ngày nay, ngoài ứng dụng trong chế tạo máy thì các ứng dụng robot trong y tế, chăm sóc sức khỏe, nông nghiệp, đóng tàu, xây dựng, an ninh quốc phòng và gia đình đang có nhu cầu gia tăng là động lực cho các robot địa hình và robot dịch vụ phát triển.

Có thể kể đến một số loại robot được quan tâm nhiều thời gian qua là: Tay máy robot (Robot Manipulators), Robot di động (Mobile Robots), Robot phỏng sinh học (Bio Inspired Robots) và Robot cá nhân (Personal Robots). Robot di động được nghiên cứu nhiều như Xe tự hành trên mặt đất AGV (Autonomous Guided Vehicles), Robot tự hành dưới nước AUV (Autonomous Underwater Vehicles), Máy bay không người lái UAV (Unmanned Arial Vehicles). Với Robot phỏng sinh học, các nghiên cứu thời gian qua tập trung vào 2 loại chính là Robot đi (Walking robots) và Robot dáng người (Humanoid Robots). Bên cạnh đó, các loại robot phỏng sinh học dưới nước như robot cá, các cấu trúc chuyển động phỏng theo sinh vật biển cũng được nhiều nhóm nghiên cứu phát triển. Hiện nay các ứng dụng của robot đang có xu hướng chuyển sang các ứng dụng thường nhật như Robot gia đình (home robots) và Robot cá nhân (Personal robots). Mặc dù về cấu trúc của các loại robot có khác nhau nhưng các nghiên cứu hiện nay đều hướng về các ứng dụng dịch vụ và hoạt động của robot trong các môi trường tự nhiên. Với sự phát triển của xã hội và quá trình hiện đại hóa ở các nước phát triển thì nhiều dịch vụ mới được hình thành làm thay đổi quan điểm về robot từ robot phục vụ công nghiệp sang robot phục vụ cho các nhu cầu xã hội và nhu cầu cá nhân của con người.

Các chuyên gia đánh giá, với cuộc cách mạng công nghiệp 4.0, ứng dụng robot trong dây chuyền sản xuất công nghiệp sẽ là một trong những trọng tâm của một cuộc cách mạng công nghệ lớn sau Internet. Việc tích hợp các hệ thống trí tuệ nhân tạo(AI) vào trong các hệ thống điều khiển giúp robot có thể thông minh hơn, thực hiện nhiều công việc phức tạp hơn. Bên cạnh đó, việc áp dụng điện toán đám mây có thể mở ra một kỉ nguyên mới. Các robot trong tương lai có thể được lập trình sẵn hoặc sử dụng công nghệ điện toán đám mây để phân tích dữ liệu cũng như trao đổi thông tin giữa các robot với nhau. Robot công nghiệp ứng dụng trong điện toán đám mây có nhiều tiềm năng “giải phóng” rào cản về năng lực xử lý dữ liệu. Vẫn đề mà các loại Robot “offline” trước nay vẫn mắc phải. Với sự tăng trưởng chóng mặt của khoa học và công nghệ, mới đây, một cảnh báo vừa được Tổ chức Hợp tác và Phát triển kinh tế (OECD) đưa ra: “Gần một nửa công việc trên thế giới có thể bị xóa sổ hoặc thay thế hoàn toàn trong 2 thập kỷ tới bởi tự động hóa, robot và toàn cầu hóa.”.

Nghiên cứu, ứng dụng robot công nghiệp tự động hóa, gia tăng mô hình “nhà máy thông minh” tại các khu công nghiệp đang là xu hướng phát triển mạnh tại Việt



Hình 2.1: Điện toán đám mây mở ra một kỷ nguyên mới cho robot

Nam hiện nay. Mới đây, Chính phủ đã chấp thuận dự án đầu tư 300 triệu USD xây dựng nhà máy sản xuất chíp ứng dụng cho robot và các sản phẩm khác với công suất 400 triệu chíp/năm.

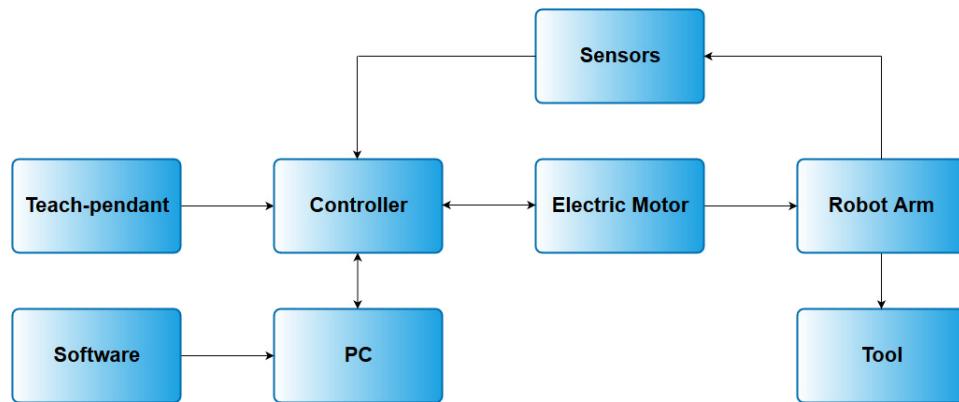
Với những ưu điểm vượt trội so với con người như độ chính xác cao, độ an toàn, độ bền, tăng năng suất lao động, đáp ứng được yêu cầu chất lượng cao trong sản xuất, việc ứng dụng robot trong nhiều khâu sản xuất của ngành công nghiệp đang ngày càng trở nên phổ biến không chỉ ở Việt Nam mà còn trên thế giới.



Hình 2.2: Hình ảnh robot bên trong một nhà máy

## 2.3 Các thành phần chính của một hệ thống robot công nghiệp

Một robot công nghiệp thường bao gồm các thành phần chính như: cánh tay robot, bộ điều khiển, các động cơ điện, dụng cụ gắn lên khâu chấp hành cuối, máy tính, thiết bị dạy học, cảm biến và các phần mềm lập trình.



Hình 2.3: Sơ đồ một hệ thống robot công nghiệp

Cánh tay robot: là kết cấu cơ khí gồm các khâu liên kết với nhau bằng khớp động để tạo nên chuyển động của robot.

Nguồn động lực là các động cơ điện, các hệ thống xi-lanh khí nén thủy lực để tay máy hoạt động.

Dụng cụ thao tác gắn lên khâu cuối của robot, tùy vào ứng dụng mà có thể sử dụng dụng cụ khác nhau: dạng tay kẹp để cầm nắm đối tượng, mỏ hàn, đầu phun sơn...

Teach-pendant là thiết bị dùng để dạy cho robot các thao tác cần thiết theo yêu cầu làm việc, sau đó robot thực hiện lặp lại các động tác đã được ghi lại.

Các phần mềm lập trình và các chương trình điều khiển được cài đặt trên máy tính, dùng để điều khiển robot thông qua bộ điều khiển(Controller).

Hệ thống cảm biến giúp bộ điều khiển biết được các trạng thái của robot.

## 2.4 ĐỘNG HỌC ROBOT

### 2.4.1 Một số khái niệm

Bậc tự do của robot (DOF: Degrees Of Freedom):

Bậc tự do là khả năng chuyển động của một cơ cấu (chuyển động quay hoặc tịnh tiến). Để dịch chuyển được một vật thể trong không gian, cơ cấu chấp hành của robot phải đạt một số bậc tự do. Bậc tự do của robot có thể tính theo công thức:

$$W = \sum_{i=1}^5 (6n - ip_i)$$

Trong đó:

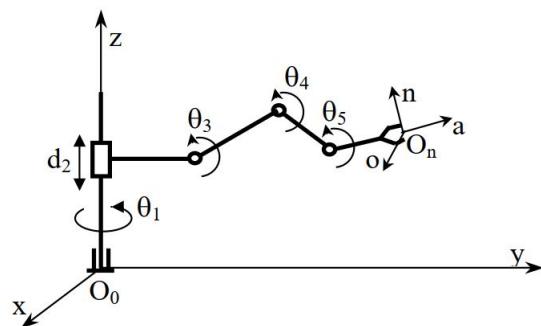
n: Số khâu động

$p_i$ : Số khớp loại i ( $i = 1, 2, 3, 4, 5$ )

Đối với các cơ cấu có các khâu được nối với nhau bằng khớp quay hoặc tịnh tiến thì số bậc tự do bằng với số khâu động.

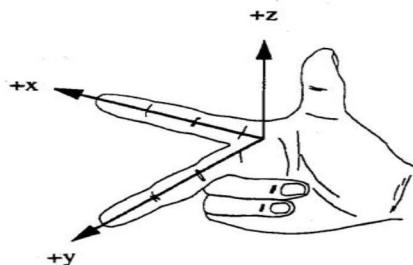
Hệ tọa độ (Coordinate frames)

Mỗi robot thường bao gồm nhiều khâu liên kết (link) với nhau qua các khớp (joint) tạo thành một xích động học xuất phát từ một khâu cơ bản (base) đứng yên. Hệ tọa độ gắn với khâu cơ bản được gọi là hệ tọa độ cơ bản (hệ tọa độ chuẩn). Các hệ tọa độ trung gian khác gắn với các khâu động gọi là hệ tọa độ suy rộng. Trong từng thời điểm, các tọa độ suy rộng (biến khớp) xác định cấu hình của robot bằng các dịch chuyển của khớp tịnh tiến hoặc khớp quay.



Hình 2.4: Tọa độ suy rộng của robot

Các hệ tọa độ gắn trên các khâu của robot phải tuân theo quy tắc bàn phải tay. Trong robot thường dùng chữ O và chỉ số n để chỉ hệ tọa độ gắn trên khâu n.

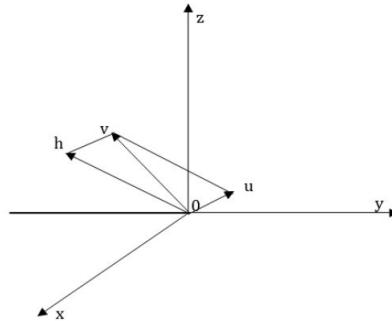


Hình 2.5: Quy tắc bàn tay phải

### 2.4.2 Các phép biến đổi hệ tọa độ

Cho  $u$  là vector điểm biểu diễn cần biến đổi,  $v$  là vector sau khi đã biến đổi,  $H$  là ma trận chuyển đổi thì  $v$  được tính theo công thức:  $v = H.u$

- Phép tịnh tiến:



Hình 2.6: Phép tịnh tiến

Cho một điểm cần tịnh tiến với vector dãn  $\vec{h} = a\vec{i} + b\vec{j} + c\vec{k}$ , ma trận chuyển đổi H được định nghĩa:

$$H = Trans(a, b, c) = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Với  $u = [x \ y \ z \ 1]^T$  thì  $v$  được xác định bởi:

$$v = H.u = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x+a \\ y+b \\ z+c \\ 1 \end{bmatrix}$$

Như vậy bản chất của phép biến đổi tịnh tiến là phép cộng giữa vector dãn và vectơ cần biến đổi.

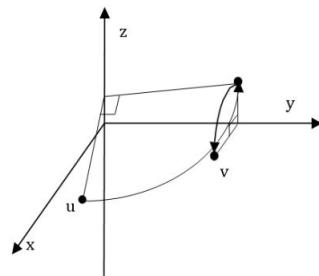
- Phép quay (Rotation) quanh các trục toa độ:

Giả sử cần quay một điểm quanh một trục tọa độ x, y, z với góc quay  $\theta^o$ , các ma trận chuyển đổi được tính theo các công thức sau:

$$Rot(x, \theta^o) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Rot(y, \theta^o) = \begin{bmatrix} \cos \theta & 0 & \sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Rot(z, \theta^o) = \begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Hình 2.7: Quay u quanh trục z  $90^\circ$  rồi quay quanh trục y  $90^\circ$

### 2.4.3 Phương pháp D-H

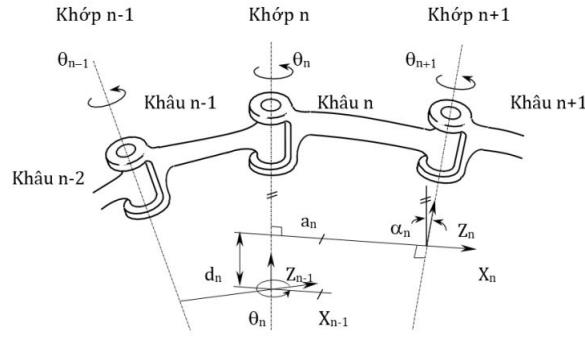
Bất kì một robot nào cũng có thể xem là tập hợp các khâu gắn liền với các khớp. Nếu đặt cho mỗi khâu một hệ trục tọa độ và sử dụng các phép biến đổi thì có thể mô tả vị trí và hướng tương đối giữa các hệ tọa độ.

Một robot có n khớp và n+1 khâu. Chỉ số các khớp bắt đầu từ 1 đến n, còn khâu bắt đầu từ 0 đến n. Khớp (i) kết nối khâu (i-1) và khâu (i). Để chọn một hệ trục tọa độ cho các khâu thì phải tuân theo một số nguyên tắc chung sau:

- Trục  $z_i$  nằm dọc theo trục của khớp (i+1).
- Trục  $x_i$  nằm dọc theo đường vuông góc chung hướng từ  $z_{i-1}$  đến  $z_i$ . Trong trường hợp 2 trục giao nhau thì  $\vec{x}_i = \vec{z}_{i-1} \times \vec{z}_i$ , trục  $\vec{y}_i = \vec{z}_i \times \vec{x}_i$ .

Xác định bộ thông số Denavit-Hartenberg:

- Gọi  $a_i$  là khoảng cách giữa trục  $z_{i-1}$  và trục  $z_i$  dọc theo trục  $x_i$ .
- Gọi  $\alpha_i$  là góc xoay  $z_{i-1}$  quanh trục  $x_i$  đến song song với  $z_i$ .
- Gọi  $d_i$  là khoảng cách dọc theo trục  $z_{i-1}$  giữa  $x_{i-1}$  và  $x_i$ .
- Gọi  $\theta_i$  là góc xoay  $x_{i-1}$  quanh trục  $z_i$  đến song song với trục  $x_i$ .



Hình 2.8: Các thông số Denavit-Hartenberg

Xác định biến khớp:

- Nếu khớp (i) là khớp quay thì  $\theta_i$  là biến khớp.
- Nếu khớp (i) là khớp tịnh tiến thì  $d_i$  là biến khớp.

Trên cơ sở các hệ tọa độ đã được gắn trên các khâu của robot thì mối quan hệ giữa các hệ tọa độ nối tiếp nhau (i-1) và (i) được xác định bởi phép quay và tịnh tiến sau:

- Quay quanh  $z_{i-1}$  một góc  $\theta_i$ .
- Tịnh tiến dọc theo  $z_{i-1}$  một đoạn  $d_i$ .
- Tịnh tiến dọc theo  $x_{i-1}$ ,  $x_i$  một đoạn  $a_i$ .
- Quay quanh  $x_i$  một góc xoắn  $\alpha_i$ .

Bốn phép biến đổi thuần nhất này thể hiện mối quan hệ của hệ trục tọa độ thuộc khâu thứ (i) so với hệ trục tọa độ thuộc khâu thứ (i-1). Tích của bốn phép biến đổi trên được gọi là ma trận chuyển đổi  ${}^{i-1}T_i$  có công thức tính như sau:

$${}^{i-1}T_i = \text{Rot}(z, \theta_i).\text{Trans}(0, 0, d_i).\text{Trans}(a_i, 0, 0).\text{Rot}(x, \alpha_i)$$

$${}^{i-1}T_i = \begin{bmatrix} \cos \theta_i & -\sin \theta_i \cos \alpha_i & \sin \theta_i \sin \alpha_i & a_i \cos \theta_i \\ \sin \theta_i & \cos \theta_i \cos \alpha_i & -\cos \theta_i \sin \alpha_i & a_i \sin \theta_i \\ 0 & \sin \alpha_i & \cos \alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.1)$$

Khi robot có n khâu thì ma trận chuyển đổi từ hệ tọa độ n về 0 được xác định theo công thức:  ${}^0T_n = {}^0T_1{}^1T_2\dots{}^{n-1}T_n$ . Gọi P là vị trí đầu công tác thì  ${}^n r_p$  được tính theo biểu thức sau:

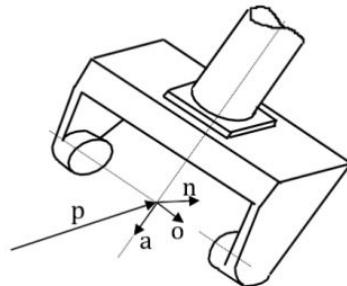
$${}^n r_p = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \Rightarrow {}^0 r_p = {}^0T_n {}^n r_p \rightarrow \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = {}^0T_n \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

#### 2.4.4 Phương trình động học thuận của robot SCARA

Mục đích của quá trình tính toán động học **ngược** là xác định vị trí và hướng của điểm tác động cuối(end-effector) từ các giá trị biến khớp đã cho. Nói cách khác, bài toán động học **ngược** có đầu vào chính là các giá trị biến khớp và đầu ra là ma trận biến đổi từ khâu cơ bản cho đến điểm tác động cuối(end-effector) của robot:

$${}^0T_6 = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Các phần tử  $p_x, p_y, p_z$  chính là các giá trị tọa độ của robot trong không gian. Trong khi đó, các vector  $n_x, n_y, n_z, o_x, o_y, o_z, a_x, a_y, a_z$  là các thành phần của vector  $\vec{n}, \vec{o}, \vec{a}$  dùng để xác định hướng của robot.

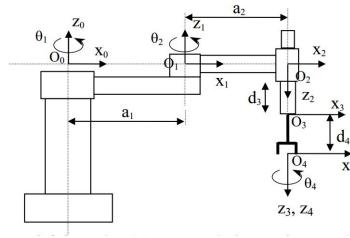


Hình 2.9: Hệ trục tọa độ gắn trên khâu chấp hành

Các vector trên hệ tọa độ gắn vào khâu chấp hành được biểu diễn như sau:

- $\vec{n}$ : Vector pháp tuyến với  $(\vec{o}, \vec{a})$  (Normal).
- $\vec{o}$ : Vector có hướng mà các ngón tay nắm vào khi cầm nắm đối tượng (Occupation).
- $\vec{a}$ : Vector có hướng tiếp cận đối tượng (Approach).

Áp dụng nguyên tắc Denavit-Hartenberg, chọn hệ trục tọa độ như sau:



Hình 2.10: Hệ trục tọa độ cho robot SCARA

Dựa trên các thông số kĩ thuật và đo thực tế giá trị  $a_1 = 415$  mm và  $a_2 = 235$  mm. Bảng thông số DH của robot SCARA được xác định như sau:

Khâu	$\theta_i(^o)$	$\alpha_i(^o)$	$a_i(mm)$	$d_i(mm)$
1	$\theta_1^*$	0	415	0
2	$\theta_2^*$	180	235	0
3	0	0	0	$d_3^*$
4	$\theta_4^*$	0	0	$d_4$

Bảng 2.1: Bảng thông số DH robot SCARA

Từ biểu thức (2.1) có thể tính được các ma trận chuyển đổi sau:

$${}^0T_1 = \begin{bmatrix} \cos \theta_1 & -\sin \theta_1 & 0 & a_1 \cos \theta_1 \\ \sin \theta_1 & \cos \theta_1 & 0 & a_1 \sin \theta_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$${}^1T_2 = \begin{bmatrix} \cos \theta_2 & \sin \theta_2 & 0 & a_2 \cos \theta_2 \\ \sin \theta_2 & -\cos \theta_2 & 0 & a_2 \sin \theta_2 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$${}^2T_3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$${}^3T_4 = \begin{bmatrix} \cos \theta_4 & -\sin \theta_4 & 0 & 0 \\ \sin \theta_4 & \cos \theta_4 & 0 & 0 \\ 0 & 0 & 1 & d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{aligned}
{}^0T_2 = {}^0T_1^1T_2 &= \begin{bmatrix} \cos(\theta_1 + \theta_2) & \sin(\theta_1 + \theta_2) & 0 & a_1 \cos(\theta_1) + a_2 \cos(\theta_1 + \theta_2) \\ \sin(\theta_1 + \theta_2) & -\cos(\theta_1 + \theta_2) & 0 & a_1 \sin(\theta_1) + a_2 \sin(\theta_1 + \theta_2) \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
{}^0T_3 = {}^0T_1^1T_2^2T_3 &= \begin{bmatrix} \cos(\theta_1 + \theta_2) & \sin(\theta_1 + \theta_2) & 0 & a_1 \cos(\theta_1) + a_2 \cos(\theta_1 + \theta_2) \\ \sin(\theta_1 + \theta_2) & -\cos(\theta_1 + \theta_2) & 0 & a_1 \sin(\theta_1) + a_2 \sin(\theta_1 + \theta_2) \\ 0 & 0 & -1 & -d_3 - d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
\Rightarrow {}^0T_4 = {}^0T_1^1T_2^2T_3^3T_4 &= \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned}$$

Các công thức tính giá trị phần tử của ma trận chuyển đổi  ${}^0T_4$  được xác định như sau:

$$\begin{aligned}
n_x &= \cos(\theta_1 + \theta_2 - \theta_4) \\
n_y &= \sin(\theta_1 + \theta_2 - \theta_4) \\
n_z &= 0 \\
o_x &= \sin(\theta_1 + \theta_2 - \theta_4) \\
o_y &= -\cos(\theta_1 + \theta_2 - \theta_4) \\
o_z &= 0 \\
a_x &= 0 \\
a_y &= 0 \\
a_z &= -1 \\
p_x &= a_1 \cos(\theta_1) + a_2 \cos(\theta_1 + \theta_2) \\
p_y &= a_1 \sin(\theta_1) + a_2 \sin(\theta_1 + \theta_2) \\
p_z &= -d_3 - d_4
\end{aligned}$$

Vậy điểm tác động cuối của robot có tọa độ  $(p_x, p_y, p_z)$ . 3 vector  $\vec{n}, \vec{o}, \vec{a}$  chính là các vector xác định hướng. Như vậy, với các giá trị biến khớp  $[\theta_1, \theta_2, d_3, \theta_4]$ , phương trình động học thuận của robot đã được xác định.

#### 2.4.5 Ma trận Jacobian

- Giới thiệu ma trận Jacobian:

Trong điều khiển robot, giá trị vận tốc là một trong những giá trị quan trọng cần tính toán. Khi hoạch định quỹ đạo trong không gian làm việc, vận tốc của đầu công tác (end-effector) bao gồm vận tốc dài và vận tốc quay. Do vậy, cần phải có một biểu thức thể hiện mối quan hệ giữa vận tốc không gian làm việc và không gian khớp. Ma trận Jacobian là ma trận thể hiện mối quan hệ giữa các giá trị vận tốc trong không gian khớp và không gian làm việc.

Ở phần trên, phương trình động học thuận của robot có thể được biểu diễn bằng ma trận chuyển đổi giữa đầu công tác và gốc tọa độ:

$$T = \begin{bmatrix} R_e(q) & p_e(q) \\ 0 & 1 \end{bmatrix}$$

trong đó  $q = [q_1, q_2, d_3, q_4]$  là các giá trị biến khớp.

Mục đích của ma trận Jacobian là thể hiện mối quan hệ giữa vận tốc biến khớp và vận tốc trong không gian làm việc. Gọi  $p_e, w_e$  là vận tốc dài, vận tốc quay của đầu công tác. Mối quan hệ của vận tốc trong không gian biến khớp và vận tốc không gian làm việc có thể biểu diễn:

$$\begin{aligned} \dot{p}_e &= J_P(q)\dot{q} \\ w_e &= J_O(q)\dot{q} \end{aligned}$$

$J_P$  là ma trận ( $3 \times n$ ) biểu diễn quan hệ giữa vận tốc dài và vận tốc khớp trong khi  $J_O$  là ma trận biểu diễn vận tốc quay và vận tốc khớp của robot. Đối với robot SCARA thì  $n = 4$  tương ứng với 4 bậc tự do.

Có thể viết lại 2 phương trình trên như sau:

$$v_e = \begin{bmatrix} \dot{p}_e \\ w_e \end{bmatrix} = J_{(q)}\dot{q}$$

- Tính ma trận Jacobian cho robot SCARA: Ma trận Jacobian của robot SCARA là một ma trận ( $6 \times 4$ ), tương ứng với 4 bậc của robot: 3 trực quay và 1 trực tịnh tiến.

Gọi  $R$  là ma trận quay trong các ma trận chuyển đổi tương ứng giữa các hệ tọa độ,  $O$  là vị trí của gốc tọa độ tương ứng. Công thức tính từng thành phần của ma trận Jacobian được trình bày ở bảng dưới.

	Prismatic	Revolute
Linear	$R_{i-1}^0 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$	$R_{i-1}^0 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \times (O_n^0 - O_{i-1}^0)$
Rotational	$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$	$R_{i-1}^0 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$

Bảng 2.2: Công thức tính giá trị phần tử ma trận Jacobian

Ma trận Jacobian có dạng như sau:

$$J = \begin{bmatrix} Jv \\ Jw \end{bmatrix} = \begin{bmatrix} J_{v1} & J_{v2} & J_{v3} & J_{v4} \\ J_{w1} & J_{w2} & J_{w3} & J_{w4} \end{bmatrix}$$

Từng phần tử trong ma trận J được tính theo công thức:

$$\begin{aligned}
 J_{v1} &= R_0^0 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \times (O_4^0 - O_0^0) = \begin{bmatrix} -a_1 \sin(\theta_1) - a_2 \sin(\theta_1 + \theta_2) \\ a_1 \cos(\theta_1) + a_2 \cos(\theta_1 + \theta_2) \\ 0 \end{bmatrix} \\
 J_{v2} &= R_1^0 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \times (O_4^0 - O_1^0) = \begin{bmatrix} -a_2 \sin(\theta_1 + \theta_2) \\ a_2 \cos(\theta_1 + \theta_2) \\ 0 \end{bmatrix} \\
 J_{v3} &= R_2^0 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} \\
 J_{v4} &= R_3^0 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \times (O_4^0 - O_3^0) \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \\
 J_{w1} &= R_0^0 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}; J_{w2} = R_1^0 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \\
 J_{w3} &= \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}; J_{w4} = R_3^0 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}
 \end{aligned}$$

## 2.4.6 Singularities

- Giới thiệu về singularities:

Ở phần trước, ma trận Jacobian dùng để xác định mối quan hệ giữa vận tốc end-effector trong không gian làm việc và vận tốc biến khớp theo công thức liên hệ:

$$\begin{aligned}
 v_e &= J_q \dot{q} \\
 \Rightarrow \dot{q} &= J(q)^{-1} v_e
 \end{aligned}$$

Ma trận Jacobian là ma trận chuyển đổi tương ứng với từng điểm sẽ có giá trị khác nhau. Công thức trên chỉ áp dụng được khi ma trận J là ma trận vuông khả nghịch. Trong quá trình robot di chuyển, có những điểm mà tại đó ma trận J không phải là một ma trận vuông khả nghịch, những điểm đó gọi là điểm kì dị(singularity).

Điểm kì dị là những điểm trên quỹ đạo di chuyển của robot có những đặc điểm sau:

- Điểm kì dị biểu diễn những cấu hình của robot mà tại đó tính linh động của cấu trúc robot bị giảm đi và không thể đặt một chuyển động tùy ý lên điểm tác động cuối(end-effector).
- Khi robot dịch tới điểm kì dị thì phương trình động học ngược có thể có nhiều nghiệm.

- Trong lân cận của điểm kì dị, một sự biến đổi nhỏ của vị trí trong không gian làm việc cũng có thể gây ra một vận tốc cực kì lớn cho các khớp.

Điểm kì dị của robot có thể phân chia thành:

- Boundary singularity: điểm kì dị xảy ra khi robot duỗi thẳng ra hoặc là gấp lại hết cỡ. Nói cách khác, điểm kì dị biên xảy ra khi robot đạt đến giới hạn biên của không gian làm việc. Để tránh việc robot đi vào điểm kì dị, trong quá trình hoạch định quỹ đạo, phải kiểm tra điều kiện đường đi có đạt đến biên giới của không gian làm việc hay không.
- Internal singularity: điểm kì dị xảy ra trong vùng làm việc của robot và nó được sinh ra bởi sự sắp xếp của 2 hay nhiều trực hoắc là muốn đạt đến một cấu hình đặc biệt. Không giống như boundary singularity, điểm kì dị này thường nguy hiểm hơn vì nó xảy ra ở bên trong không gian làm việc khi robot được hoạch định theo một quỹ đạo nào đó.

- Xác định điểm kì dị của robot SCARA:

Từ ma trận Jacobian tính được ở phần trước, mối liên hệ giữa vận tốc trong không gian làm việc và vận tốc biến khớp:

$$\begin{cases} w_x = 0 \\ w_y = 0 \\ w_z = \dot{q}_1 + \dot{q}_2 - \dot{q}_4 \end{cases}$$

$$\begin{aligned} \dot{\mathbf{X}} &= J_v \dot{\mathbf{q}} \\ \Rightarrow \begin{bmatrix} x \\ y \\ z \end{bmatrix} &= \begin{bmatrix} -a_1 \sin \theta_1 - a_2 \sin(\theta_1 + \theta_2) & -a_2 \sin(\theta_1 + \theta_2) & 0 \\ a_1 \cos \theta_1 + a_2 \cos(\theta_1 + \theta_2) & a_2 \cos(\theta_1 + \theta_2) & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{d}_3 \end{bmatrix} \\ \Rightarrow \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{d}_3 \end{bmatrix} &= \begin{bmatrix} -a_1 \sin \theta_1 - a_2 \sin(\theta_1 + \theta_2) & -a_2 \sin(\theta_1 + \theta_2) & 0 \\ a_1 \cos \theta_1 + a_2 \cos(\theta_1 + \theta_2) & a_2 \cos(\theta_1 + \theta_2) & 0 \\ 0 & 0 & -1 \end{bmatrix}^{-1} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \end{aligned}$$

Khi robot di chuyển vào điểm kì dị thì ma trận Jacobian bị mất bậc, không còn là ma trận vuông khả nghịch. Do vậy, điều kiện để ma trận  $J_v$  khả nghịch là  $\det(J_v) \neq 0$ . Vậy điểm kì dị của robot SCARA chính là nghiệm của phương trình  $\det(J_v) = 0$ :

$$\begin{aligned}
& \det(J_v) = 0 \\
& \Rightarrow \det \left( \begin{bmatrix} -a_1 \sin \theta_1 - a_2 \sin(\theta_1 + \theta_2) & -a_2 \sin(\theta_1 + \theta_2) & 0 \\ a_1 \cos \theta_1 + a_2 \cos(\theta_1 + \theta_2) & a_2 \cos(\theta_1 + \theta_2) & 0 \\ 0 & 0 & -1 \end{bmatrix} \right) = 0 \\
& \Rightarrow a_1 a_2 \cos(\theta_1 + \theta_2) - a_1 a_2 \sin(\theta_1 + \theta_2) = 0 \\
& \Rightarrow \tan(\theta_1 + \theta_2) = \tan(\theta_1) \\
& \Rightarrow \theta_1 + \theta_2 = \theta_1 + k\pi \\
& \Rightarrow \theta_2 = k\pi \\
& \Rightarrow \theta_2 = 0^\circ \vee \theta_2 = 180^\circ
\end{aligned}$$

Như vậy, robot SCARA có 2 điểm kì dị là khi robot duỗi ra và robot gấp lại thành 1 đường thẳng. Trường hợp điểm kì dị khi  $\theta_2 = 180^\circ$  có thể không chế bằng không cho robot di chuyển ra khỏi giới hạn làm việc. Trường hợp còn lại khi  $\theta_2 = 0$  có thể tránh được bằng cách kiểm tra các vị trí trên đường đi của robot có đạt đến giá trị biên ngoài vùng biên giới làm việc.

#### 2.4.7 Phương trình động học ngược robot SCARA

Ma trận chuyển đổi hay phương trình động học thuận chỉ cho biết vị trí hay là hướng của robot trong không gian. Trong thực tế, để hoạch định quỹ đạo cũng như điều khiển robot thì các giá trị phải là các giá trị biến khớp. Bài toán động học ngược yêu cầu tìm ra mối liên hệ giữa không gian biến khớp và không gian làm việc. Nói cách khác, việc tìm các giá trị biến khớp  $(\theta_1, \theta_2, d_3, \theta_4)$  thông qua ma trận chuyển đổi  ${}^0T_4$  chính là bài toán động học ngược của robot.

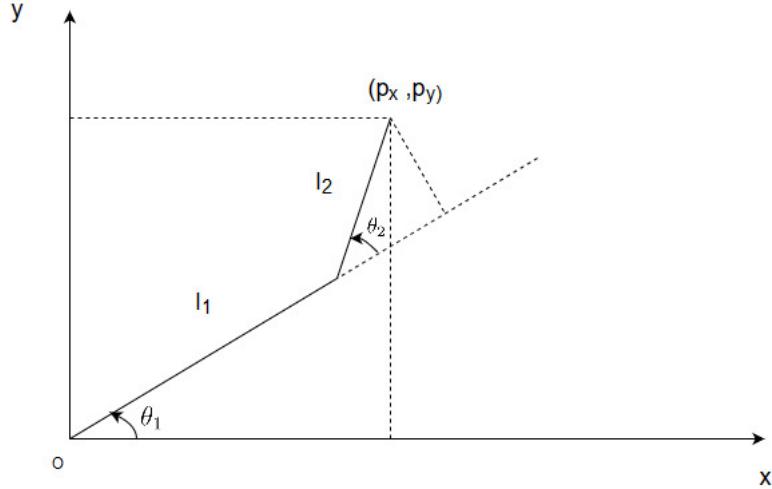
Ở phần trước, bài toán động học thuận đã xác định được ma trận chuyển đổi:

$${}^0T_4 = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \end{bmatrix}$$

Với:

$$\begin{aligned}
& \begin{cases} px = a_1 \cos \theta_1 + a_2 \cos(\theta_1 + \theta_2) \\ py = a_1 \sin \theta_1 + a_2 \sin(\theta_1 + \theta_2) \\ pz = -d_3 - d_4 \end{cases} \\
& \Rightarrow \begin{cases} {}^{p_x} = a_1^2 \cos^2 \theta_1 + 2a_1 a_2 \cos \theta_1 \cos(\theta_1 + \theta_2) + a_2^2 \cos^2(\theta_1 + \theta_2) \\ {}^{p_y} = a_1^2 \sin^2 \theta_1 + 2a_1 a_2 \sin \theta_1 \sin(\theta_1 + \theta_2) + a_2^2 \sin^2(\theta_1 + \theta_2) \\ {}^{p_z} = -d_3 - d_4 \end{cases} \\
& \quad {}^{p_x} + {}^{p_y} = a_1^2 + a_2^2 + 2a_1 a_2 \cos \theta_2 \\
& \quad \Rightarrow \cos \theta_2 = \frac{{}^{p_x} + {}^{p_y} - a_1^2 - a_2^2}{2a_1 a_2} \\
& \quad \Rightarrow \sin \theta_2 = \pm \sqrt{1 - \cos^2 \theta_2}
\end{aligned}$$

Xét trên mặt phẳng Oxy:



Hình 2.11: Robot SCARA trong mặt phẳng Oxy

$$\theta_1 = \text{atan}2(p_y, p_x) - \text{atan}2(a_2 \sin \theta_2, a_1 + a_2 \cos \theta_2)$$

Cũng từ ma trận chuyển đổi  ${}^0T_4$  thu được:

$$\begin{aligned} \begin{cases} n_x = \cos(\theta_1 + \theta_2 - \theta_4) \\ n_y = \sin(\theta_1 + \theta_2 - \theta_4) \end{cases} \\ \Rightarrow \tan(\theta_1 + \theta_2 - \theta_4) = \frac{n_y}{n_x} \\ \Rightarrow \theta_1 + \theta_2 - \theta_4 = \text{atan}\left(\frac{n_y}{n_x}\right) + k\pi \\ \Rightarrow \theta_4 = \theta_1 + \theta_2 - \text{atan}\left(\frac{n_y}{n_x}\right) - k\pi \end{aligned}$$

Vậy phương trình động học ngược của robot SCARA là:

$$\begin{cases} \theta_1 = \text{atan}2(p_y, p_x) - \text{atan}2(a_2 \sin \theta_2, a_1 + a_2 \cos \theta_2) \\ \theta_2 = \pm \text{atan}2(\sin \theta_2, \cos \theta_2) \\ d3 = -p_z - d_4 \\ \theta_4 = \theta_1 + \theta_2 - \text{atan}\left(\frac{n_y}{n_x}\right) - k\pi \end{cases}$$

Phương trình động học ngược của SCARA trả về 2 giá trị nghiệm của  $\theta_2$ , do đó tạo ra nhiều tập nghiệm khác nhau. Trong quá trình hoạch định quỹ đạo và tính toán để đưa vào bộ điều khiển, giá trị của  $\theta_2$  phải được chọn sao cho vị trí tiếp theo gần với vị trí hiện tại nhất. Nói cách khác, ta chọn giá trị  $\theta_2$  sao cho  $\Delta\theta_2$  giữa 2 thời điểm là nhỏ nhất.

## 2.5 Hoạch định quỹ đạo robot

Trong các ứng dụng của Robot công nghiệp, ta thường gặp 2 trường hợp như sau:

Trường hợp 1: Khâu chấp hành cuối của robot chỉ cần đạt được vị trí và hướng tại các điểm nút. Đây chính là phương pháp điều khiển điểm (PTP). Tại đó, bàn tay robot thực hiện các thao tác cầm nắm hoặc buông nả các đối tượng. Đây là trường hợp các robot thực hiện công việc vận chuyển và trao đổi phôi liệu trong một hệ thống tự động linh hoạt robot hóa. Bàn tay robot không tham gia trực tiếp vào các công việc phức tạp như hàn, cắt kim loại,... Các điểm nút là mục tiêu quan trọng nhất, còn dạng đường đi tới các điểm nút là vấn đề thứ yếu. Trong trường hợp này, robot thường được lập trình bằng phương pháp dạy học, không cần tính toán chương trình động học ngược robot, chuyển động mong muốn được ghi lại như một tập hợp các góc khớp để robot thực hiện lại khi làm việc.

Trường hợp 2: Khâu chấp hành cuối phải xác định đường đi qua các điểm nút theo thời gian thực. Đó là trường hợp các tay máy **trực tiếp** tham gia vào thực hiện các công việc như sơn, hàn, cắt kim loại... Vấn đề hoạch định quỹ đạo chuyển động trong trường hợp này rất cần thiết, quyết định chất lượng thực hiện các công việc mà robot đảm nhận. Trong trường hợp này, cần phải dùng đến các phương trình động học thuận và động học ngược.

Do vậy, tùy vào nhiệm vụ của robot để chọn phương pháp hoạch định quỹ đạo phù hợp. Mục đích của việc hoạch định quỹ đạo là tập giá trị đầu vào cho bộ điều khiển robot sao cho tay máy có thể di chuyển theo quỹ đạo đã định sẵn. Thông thường, robot di chuyển trên quỹ đạo với một vận tốc và gia tốc quy định trước. Do vậy, cần phải thiết lập một quy luật chuyển động theo thời gian để đáp ứng được các giới hạn đó, đồng thời giúp robot có thể di chuyển một cách mượt mà.

Quỹ đạo phải đảm bảo:

- Liên tục về vị trí
- Liên tục về vận tốc
- Liên tục về gia tốc

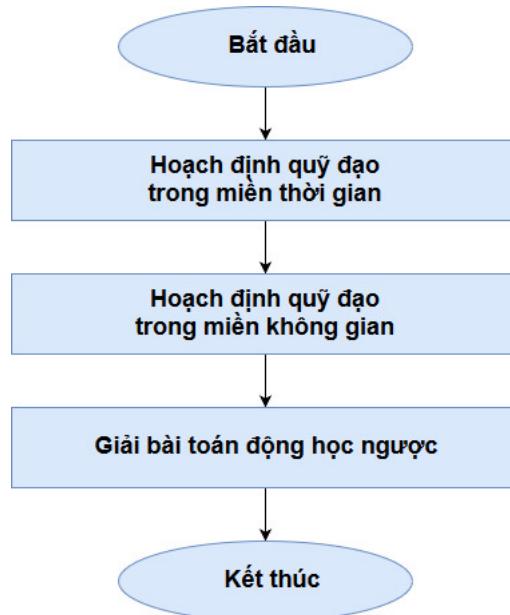
Có 2 phương pháp hoạch định quỹ đạo phổ biến:

- Hoạch định quỹ đạo trong không gian biến khớp.
- Hoạch định quỹ đạo trong không gian làm việc.

Trong luận văn, phương pháp hoạch định quỹ đạo trong không gian làm việc được sử dụng để điều khiển chuyển động cho robot SCARA.

### 2.5.1 Tổng quan về quá trình hoạch định quỹ đạo trong không gian làm việc

Quá trình hoạch định quỹ đạo trong không gian làm việc được thể hiện như trong hình 2.12.



Hình 2.12: Các bước hoạch định quỹ đạo Robot

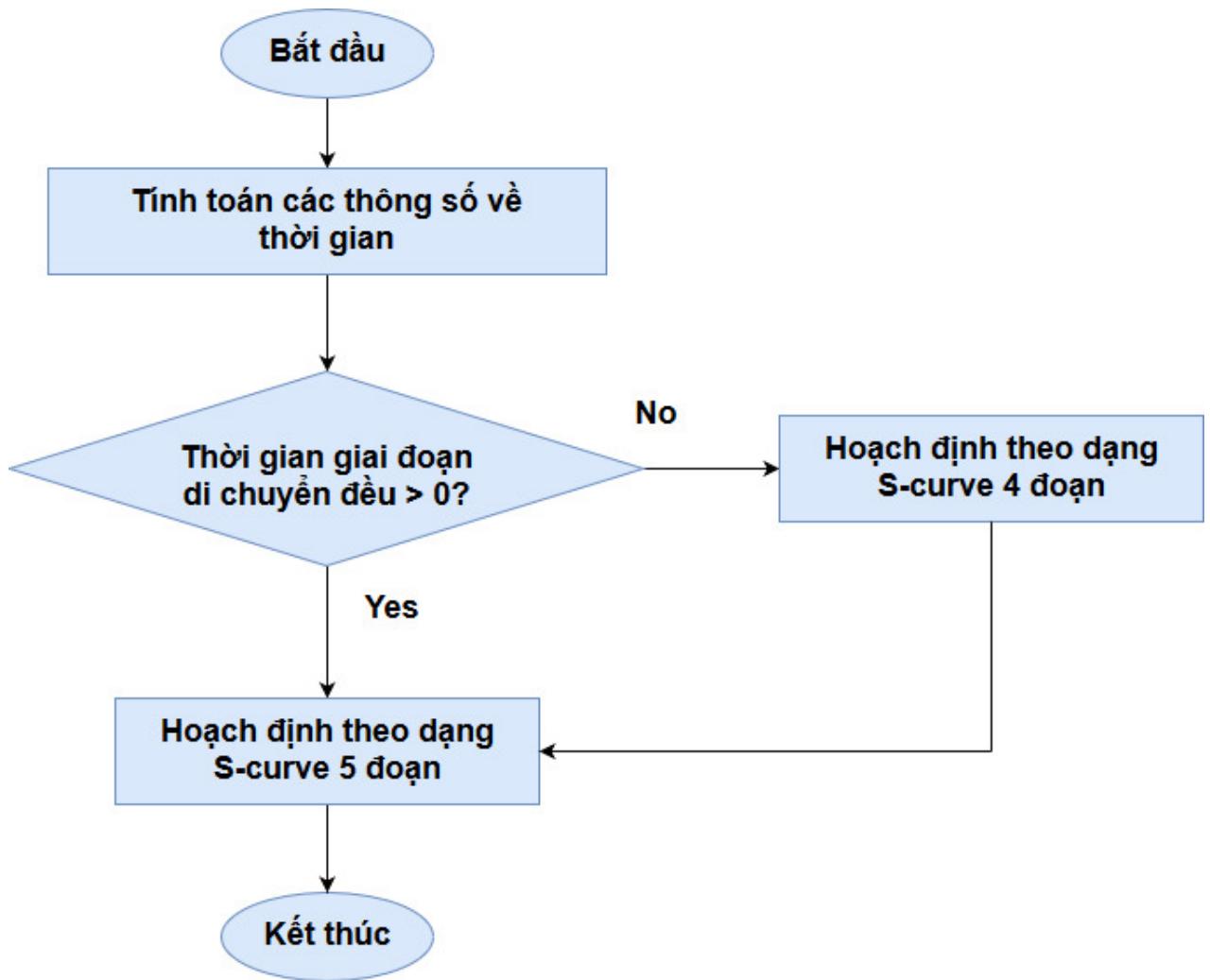
Quá trình hoạch định quỹ đạo trong không gian làm việc được chia làm 3 giai đoạn:

- Hoạch định quỹ đạo trong miền thời gian: tạo ra các profile về vị trí, vận tốc và gia tốc cũng chính là luật thời gian chuyển động trên đường đi.
- Quá trình hoạch định quỹ đạo trong miền không gian, tạo ra vector vị trí của robot trong không gian làm việc.
- Giải bài toán động học ngược: Sử dụng phương trình động học ngược của robot để tìm giá trị góc khớp tương ứng với vị trí trong không gian làm việc tìm được ở bước trên.

### 2.5.2 Hoạch định quỹ đạo trong miền thời gian

Mục đích của quá trình là tạo nên các profile gia tốc, vận tốc và vị trí của robot thực hiện trên quỹ đạo theo một dạng mà người lập trình mong muốn, nhằm tạo nên sự mượt mà trong quá trình di chuyển cũng như đảm bảo điều quan trọng nhất là sự chính xác về vị trí. Để robot đạt được sự mượt mà trong quá trình di chuyển, người lập trình cần chú ý tới vận tốc và gia tốc của điểm cuối(end effector). Có nhiều dạng vận tốc để điều khiển robot đạt được điều này, một trong số đó là dạng vận tốc đường cong S-curve.

Giả sử robot muốn di chuyển từ điểm A đến điểm B. Trong quá trình chuyển động, vận tốc đạt giá trị cực đại là  $v_{max}$ , gia tốc cực đại là  $a_{max}$ , độ dài quỹ đạo trong không gian làm việc là L. Với các giá trị  $L, v_{max}, a_{max}$ , trong quá trình di chuyển, robot sẽ đạt vận tốc lớn nhất là  $v_{lim}$ . Dựa vào giá trị  $v_{lim}$  để chọn dạng profile vận tốc phù hợp:



Hình 2.13: Quá trình hoạch định quỹ đạo trong miền thời gian

- $v_{lim} = v_{max}$ : hoạch định quỹ đạo theo kiểu đường cong Scurve 5 đoạn.

Từ đồ thị s, v, a của quỹ đạo Scurve, ta dễ dàng xác định được:

$$t_m = \frac{v_{max}}{a_{max}}$$

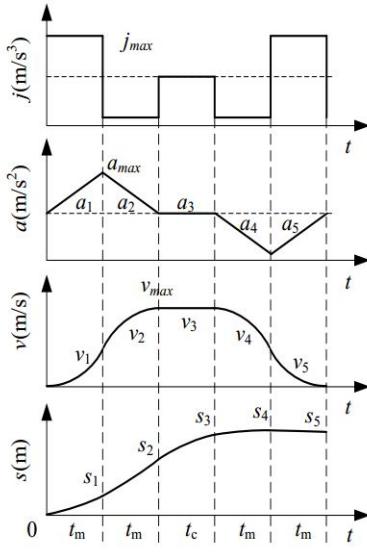
$$L = v_{max}(2t_m + t_c) \Rightarrow t_c = \frac{L}{v_{max}} - 2t_m$$

Quãng đường di chuyển, vận tốc và gia tốc trong suốt quỹ đạo của robot tuân theo quy luật ở các hệ phương trình sau:

$$\text{Đặt } j_{max} = \frac{a_{max}}{t_m}$$

Với  $t \in [0, t_m]$ :

$$\begin{cases} a_1 = j_{max}t \\ v_1 = \frac{1}{2}j_{max}t^2 \\ s_1 = \frac{1}{6}j_{max}t^3 \end{cases}$$



Hình 2.14: Đồ thị s, v, a trên quỹ đạo robot

Với  $t \in [t_m, 2t_m]$ :

$$\begin{cases} a_2 = j_{max}t_m - j_{max}(t - t_m) \\ v_2 = v_1 + j_{max}t_m(t - t_m) - \frac{1}{2}j_{max}(t - t_m)^2 \\ s_2 = s_1 + v_1(t - t_m) + \frac{1}{2}j_{max}t_m(t - t_m)^2 - \frac{1}{6}j_{max}(t - t_m)^3 \end{cases}$$

Với  $t \in [2t_m, 2t_m + t_c]$ :

$$\begin{cases} a_3 = 0 \\ v_3 = v_1 + \frac{1}{2}j_{max}t_m^2 \\ s_3 = s_2 + [v_1 + \frac{1}{2}j_{max}t_m^2](t - 2t_m) \end{cases}$$

Với  $t \in [2t_m + t_c, 3t_m + t_c]$ :

$$\begin{cases} a_4 = -j_{max}[t - (2t_m + t_c)] \\ v_4 = v_3 - \frac{1}{2}j_{max}[t - (2t_m + t_c)]^2 \\ s_4 = s_3 + v_3[t - (2t_m + t_c)] - \frac{1}{6}j_{max}[t - (2t_m + t_c)]^3 \end{cases}$$

Với  $t \in [3t_m + t_c, 4t_m + t_c]$ :

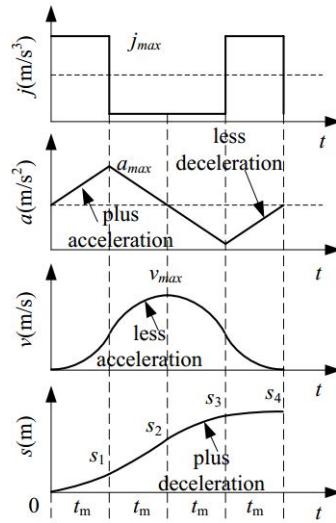
$$\begin{cases} a_5 = -j_{max}t_m + j_{max}[t - (3t_m + t_c)] \\ v_5 = v_4 - j_{max}t_m[t - (3t_m + t_c)] + \frac{1}{2}j_{max}[t - (3t_m + t_c)]^2 \\ s_5 = s_4 + v_4[t - (3t_m + t_c)] - \frac{1}{2}j_{max}t_m[t - (3t_m + t_c)]^2 + \frac{1}{6}j_{max}[t - (3t_m + t_c)]^3 \end{cases}$$

- $v_{lim} < v_{max}$ : hoạch định quỹ đạo theo kiểu đường cong Scurve 4 đoạn.

Với giá trị  $t_c$  tính được ở mục trên, có 2 khả năng xảy ra:

- $t_c > 0$ : Dạng quỹ đạo có 5 đoạn như ở trên.

- $t_c \leq 0$ : Lúc này giá trị  $v_{lim} \leq v_{max}$ , vận tốc tối đa của robot trong quá trình không đạt đến  $v_{max}$ . Do đó, quỹ đạo chỉ có 4 đoạn, vận tốc không có đoạn di chuyển đều.



Hình 2.15: Đồ thị  $s$ ,  $v$ ,  $a$  trên quỹ đạo robot

Với quỹ đạo Scurve 4 đoạn, giá trị  $v_{max}$  được tính lại cũng như giá trị  $t_m$ .  
 Dựa vào đồ thị:

$$a_{max}t_m = v_{max} \quad (2.2)$$

$$L = 2t_m v_{max} \quad (2.3)$$

Thay (2.2) vào (2.3) ta được:

$$L = 2a_{max}t_m^2 \Rightarrow t_m = \sqrt{\frac{L}{2a_{max}}}$$

Quy luật quãng đường, vận tốc, gia tốc của robot trên quỹ đạo được xác định theo các hệ phương trình dưới đây:

$$\text{Đặt } j_{max} = \frac{a_{max}}{t_m}$$

Với  $t \in [0, t_m]$ :

$$\begin{cases} a_1 = j_{max}t \\ v_1 = \frac{1}{2}j_{max}t^2 \\ s_1 = \frac{1}{6}j_{max}t^3 \end{cases}$$

Với  $t \in [t_m, 2t_m]$ :

$$\begin{cases} a_2 = j_{max}t_m - j_{max}(t - t_m) \\ v_2 = v_1 + j_{max}t_m(t - t_m) - \frac{1}{2}j_{max}(t - t_m)^2 \\ s_2 = s_1 + v_1(t - t_m) + \frac{1}{2}j_{max}t_m(t - t_m)^2 - \frac{1}{6}j_{max}(t - t_m)^3 \end{cases}$$

Với  $t \in [2t_m, 3t_m]$ :

$$\begin{cases} a_3 = -j_{max}(t - 2t_m) \\ v_3 = v_2 - \frac{1}{2}j_{max}(t - 2t_m)^2 \\ s_3 = s_2 + v_2(t - 2t_m) - \frac{1}{6}j_{max}(t - 2t_m)^3 \end{cases}$$

Với  $t \in [3t_m, 4t_m]$ :

$$\begin{cases} a_4 = -j_{max}t_m + j_{max}(t - 3t_m) \\ v_4 = v_3 - j_{max}t_m(t - 3t_m) + \frac{1}{2}j_{max}(t - 3t_m)^2 \\ s_4 = s_3 + v_3(t - 3t_m) - \frac{1}{2}j_{max}t_m(t - 3t_m)^2 + \frac{1}{6}j_{max}(t - 3t_m)^3 \end{cases}$$

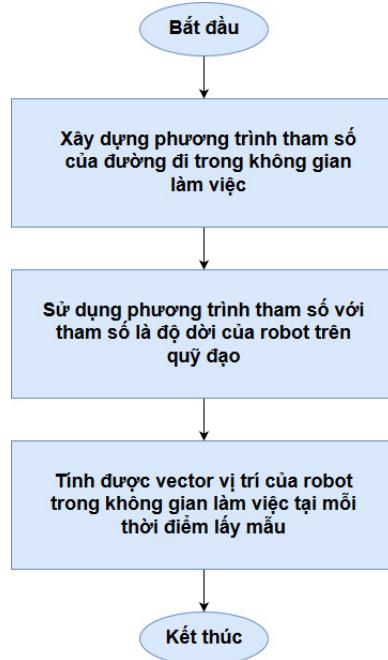
Phương trình được đưa ra ở trên chính là phương trình dạng liên tục của profile vận tốc. Trong thực tế, các bộ điều khiển lại ở miền rời rạc. Do đó, không thể sử dụng trực tiếp những phương trình giống như đã được trình bày. Để thực hiện được điều này, ta thực hiện lấy mẫu trên quỹ đạo với thời gian lấy mẫu là  $T_s$ . Giá trị quãng đường, vận tốc, gia tốc tại thời điểm thời gian tức thời thứ k được định nghĩa như bên dưới:

$$\begin{cases} s(t = kT_s) = s_k \\ v(t = kT_s) = v_k \\ a(t = kT_s) = a_k \end{cases}$$

Việc chọn thời gian lấy mẫu ảnh hưởng nhiều đến chất lượng của quỹ đạo. Với giá trị  $T_s$  lớn sẽ dẫn đến sai số, tuy nhiên nếu  $T_s$  quá nhỏ thì thời gian tính toán sẽ mất nhiều thời gian. Do thời gian trên mỗi đoạn di chuyển không phải bao giờ cũng là bội số của  $T_s$  nên cần phải thực hiện làm tròn để tính số điểm trên mỗi đoạn. Tuy nhiên, cần phải đảm bảo số điểm trên toàn bộ quỹ đạo sau khi làm tròn không quá sai lệch nhiều so với ban đầu cũng như đảm bảo các điều kiện giới hạn về vận tốc và gia tốc.

### 2.5.3 Hoạch định quỹ đạo trong miền không gian

Các bước của quá trình hoạch định quỹ đạo trong miền không gian làm việc được trình bày như trong sơ đồ hình 2.16.



Hình 2.16: Quá trình hoạch định quỹ đạo trong miền không gian

Quá trình hoạch định quỹ đạo trong miền thời gian trả về các profile: độ dời robot thực hiện, vận tốc và gia tốc trên quỹ đạo. Với mỗi dạng đường đi, đều có thể biểu diễn dưới dạng phương trình tham số. Mỗi giá trị của độ dời  $s$  được sinh ra từ quá trình hoạch định quỹ đạo trong miền thời gian tương ứng với một vị trí khác nhau trên đường đi. Do đó, có thể sử dụng  $s$  là tham số trong phương trình tham số đường đi. Từ phương trình tham số có được, lần lượt thực hiện tính toán giá trị vị trí theo từng trục của robot theo công thức:

$$\begin{cases} p_x = g_x(s(t)) \\ p_y = g_y(s(t)) \\ p_z = g_z(s(t)) \end{cases}$$

Kết thúc quá trình hoạch định quỹ đạo trong miền không gian, mọi vị trí của robot trên đường đi tại các thời điểm lấy mẫu đều có thể xác định được.

Trong công nghiệp, 2 dạng đường đi được sử dụng phổ biến nhất là đoạn thẳng và đường tròn.

- Đoạn thẳng: Giả sử đường đi mong muốn từ điểm  $p_i$  đến điểm  $p_f$  là một đoạn thẳng.

Phương trình tham số của đường đi là:

$$p(s) = p_i + \frac{s}{\|p_f - p_i\|}(p_f - p_i)$$

Ta có:  $p(0) = p_i$  và  $p(\|p_f - p_i\|) = p_f$ . Chiều trên đường đi bởi phương trình tham số đoạn thẳng là từ  $p_i$  đến  $p_f$ . Đạo hàm phương trình tham số theo s:

$$\frac{dp}{ds} = \frac{1}{\|p_f - p_i\|}(p_f - p_i)$$

$$\frac{d^2p}{ds^2} = 0$$

- Đường tròn: Giả sử quỹ đạo trong không gian là đường tròn có tâm là  $O'$ . Gắn hệ tọa độ  $O'x'y'z'$ , trục  $z'$  vuông góc với mặt phẳng đường tròn và đi qua tâm đường tròn. Gọi  $P$  là một điểm trên đường tròn có chiều dài cung tính từ điểm ban đầu  $P_i$  là  $s$ .

Trong hệ tọa độ  $O'x'y'z'$ , phương trình tham số của đường tròn:

$$p_{(s)}' = \begin{bmatrix} \rho \cos\left(\frac{s}{\rho}\right) \\ \rho \sin\left(\frac{s}{\rho}\right) \\ 0 \end{bmatrix}$$

với  $\rho$  là bán kính của đường tròn.

Như vậy, trong hệ tọa độ Oxyz, phương trình tham số của đường tròn trở thành:

$$p(s) = OO' + Rp_{(s)}'$$

trong đó  $R$  là ma trận xoay biến đổi từ hệ tọa độ  $O'x'y'z'$  thành hệ tọa độ Oxyz,  $R$  có thể được viết:

$$R = [x' \quad y' \quad z']$$

$x'$ ,  $y'$ ,  $z'$  biểu diễn vector đơn vị của hệ tọa độ được biểu diễn trong hệ tọa độ Oxyz. Đạo hàm phương trình tham số:

$$\frac{dp}{ds} = R \begin{bmatrix} -\sin\left(\frac{s}{\rho}\right) \\ \cos\left(\frac{s}{\rho}\right) \\ 0 \end{bmatrix}$$

$$\frac{d^2p}{ds^2} = R \begin{bmatrix} -\cos\left(\frac{s}{\rho}\right)/\rho \\ \sin\left(\frac{s}{\rho}\right)/\rho \\ 0 \end{bmatrix}$$

## 1. Quy hoạch vị trí

Gọi  $x_e$  là vector trong không gian làm việc biểu diễn vị trí của end-effector. Hoạch định quỹ đạo trong không gian làm việc có nghĩa là xác định hàm  $x_e(t)$  để end-effector có thể di chuyển từ điểm đầu đến điểm cuối trong khoảng thời gian  $t_f$  trên đường đi với quy luật thời gian chuyển động được chỉ ra.

$p_e = f(s)$  là vector  $(3 \times 1)$  biểu diễn tham số đường đi, là một hàm của độ dài quỹ đạo di chuyển  $s$ ; end-effector di chuyển từ  $p_i$  đến  $p_f$  trong thời gian  $t_f$ . Giả

sử gốc của độ dài cung tại  $p_i$  và chiều gây ra trên  $\Gamma$  là từ  $p_i$  đến  $p_f$ . Độ dài cung biến đổi từ  $s = 0$  tại thời điểm  $t = 0$  đến  $s = s_f$  tại thời điểm  $t = t_f$ . Quy luật thời gian chuyển động trên đường đi được biểu diễn bằng  $s(t)$  được sinh ra bởi quá trình hoạch định trong miền thời gian

Vận tốc của điểm  $p_e$  trên đường đi được xác định bằng:

$$\dot{p}_e = \dot{s} \frac{dp_e}{ds} = \dot{s}t$$

trong đó  $t$  là vector tiếp tuyến của đường đi tại điểm  $p$ .

Công thức tính vận tốc và gia tốc trên mỗi dạng quy đạo:

- Đoạn thẳng:

$$\begin{aligned}\dot{p}_e &= \frac{\dot{s}}{\|p_f - p_i\|}(p_f - p_i) = \dot{s}t \\ \ddot{p}_e &= \frac{\ddot{s}}{\|p_f - p_i\|}(p_f - p_i) = \ddot{s}t\end{aligned}$$

- Cung tròn:

$$\begin{aligned}\dot{p}_e &= R \begin{bmatrix} -\dot{s} \sin(\frac{s}{\rho}) \\ \dot{s} \cos(\frac{s}{\rho}) \\ 0 \end{bmatrix} \\ \ddot{p}_e &= R \begin{bmatrix} -\dot{s}^2 \cos(\frac{s}{\rho})/\rho - \ddot{s} \sin(\frac{s}{\rho}) \\ -\dot{s}^2 \sin(\frac{s}{\rho})/\rho + \ddot{s} \cos(\frac{s}{\rho}) \\ 0 \end{bmatrix}\end{aligned}$$

## 2. Quy hoạch hướng

Hướng của đầu công tác biểu thị bằng ma trận quay  $R = R_n^0$ . Ở thời điểm ban đầu,  $R(t = 0) = R_i$ , ở thời điểm cuối  $R(t = t_f) = R_f$ . Ma trận chuyển đổi từ  $R_i$  sang  $R_f$  là  $R_i^f$  sao cho  $R_f = R_i R_i^f$ , nghĩa là  $R_i^f = R_i^T R_f$ .

Biết  $R_i, R_f$ , tính được:

$$\begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

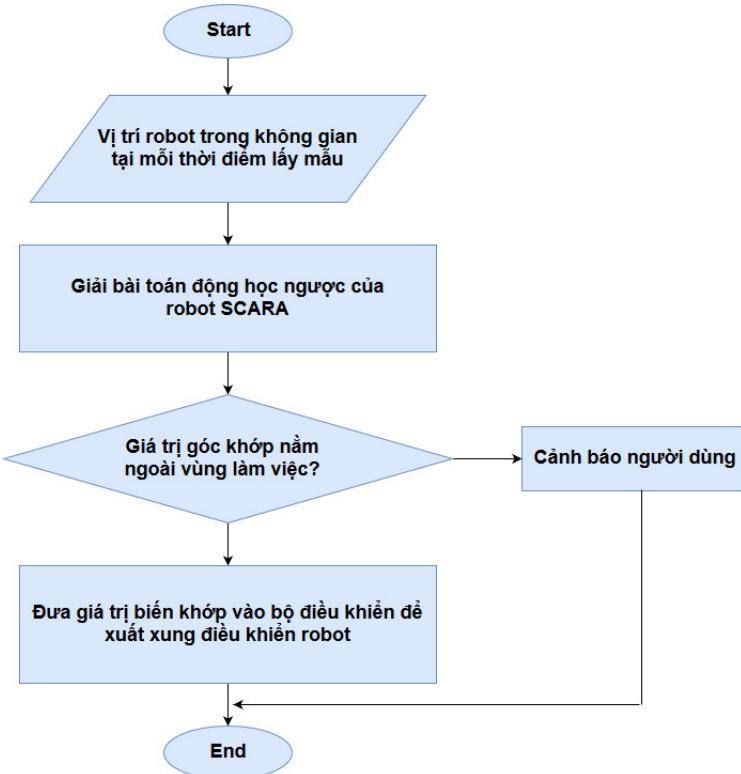
Yêu cầu đặt ra: tìm một ma trận  $R_t(t)$  theo tham số  $s(t)$  sao cho  $R_t(t = 0) = I, R_t(t = t_f) = I$ . Có thể xem ma trận  $R_i^f$  là kết quả hệ trực giao quanh vector  $\vec{r}$  một góc  $\alpha$  xác định bởi:

$$\begin{aligned}\alpha &= \cos^{-1} \left( \frac{r_{11} + r_{22} + r_{33} - 1}{2} \right) \\ r &= \frac{1}{2 \sin \alpha} \begin{bmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{bmatrix}\end{aligned}$$

Với  $r, \alpha$  đã biết, suy ra được:

$$\begin{cases} R(s) = R_i \text{Rot}(r, \alpha(s)) \\ \alpha(s) = s\alpha, s \in [0, 1] \end{cases}$$

#### 2.5.4 Giải bài toán động học ngược



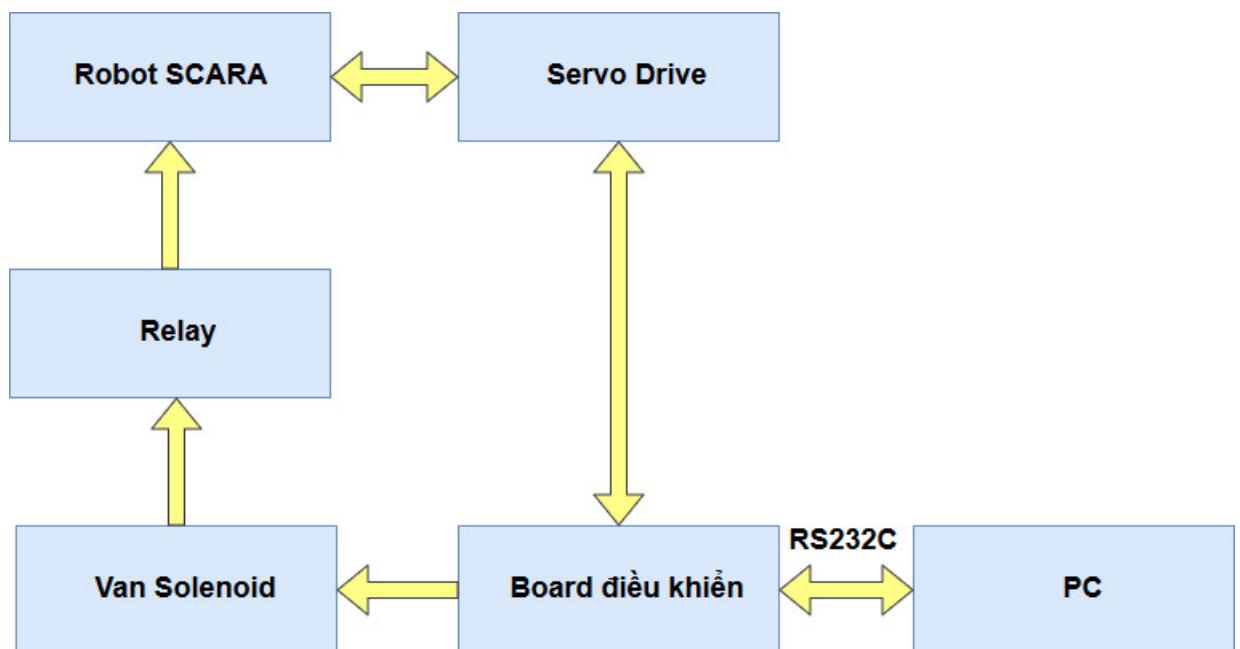
Hình 2.17: Các bước quá trình Inverse Kinematic

Với vector vị trí trong không gian làm việc xác định được sau quá trình quy hoạch vị trí, sử dụng phương trình động học ngược được xây dựng ở chương 3, dễ dàng xác định được các giá trị biến khớp  $\theta_1, \theta_2, d_3$ . Giá trị biến khớp  $\theta_4$  được tính từ ma trận xoay được xây dựng trong quá trình quy hoạch hướng. Bên cạnh đó, cần phải kiểm tra lại các giá trị biến khớp được tính ra với các điều kiện ràng buộc về không gian làm việc của robot. Nếu quỹ đạo đặt ra không nằm trong vùng làm việc, hệ thống sẽ đưa ra cảnh báo cho người dùng biết để có thể lựa chọn đường đi khác thích hợp hơn. Như vậy, từ kết quả thu được từ quá trình Path Planning, quá trình Inverse Kinematic tìm ra được các giá trị góc khớp của robot. Từ đó, bộ điều khiển ra lệnh cho board điều khiển có thể tính và xuất ra số lượng xung tương ứng để điều khiển robot theo đúng quỹ đạo hoạch định.

## Chương 3

# TỔNG QUAN VỀ CẤU TRÚC HỆ THỐNG

Cấu trúc của hệ thống được trình bày như trong sơ đồ dưới đây:



Hình 3.1: Cấu trúc hệ thống điều khiển robot SCARA

Hệ thống được thiết kế bao gồm:

- Phần cứng:
  - Robot SCARA: là kết cấu cơ khí, thực hiện các chuyển động cung như tháo tách gấp, thả vật.
  - Board điều khiển có nhiệm vụ xuất xung ra các Driver để điều khiển động cơ của robot SCARA. Ngoài ra để đóng, mở tay kẹp của robot, tín hiệu đóng, ngắt cũng được điều khiển bởi board điều khiển.

- Servo Driver: nhận xung từ board điều khiển, qua đó điều khiển động cơ của robot theo yêu cầu lập trình. Ngoài ra, Servo Driver còn có nhiệm vụ đọc xung encoder từ động cơ robot, gửi về board điều khiển.
- Relay: Nhận tín hiệu điều khiển Output từ board điều khiển để chuyển trạng thái van solenoid, qua đó thực hiện đóng mở tay kẹp robot.
- Van solenoid: nhận tín hiệu điều khiển từ board điều khiển thông qua relay để chuyển trạng thái đóng, mở tay kẹp robot.
- PC: thực hiện giao tiếp với board điều khiển thông qua giao thức RS232C để điều khiển hoạt động hệ thống cũng như lấy trạng thái hệ thống lên máy tính.

- Phần mềm:

- Phần mềm ở trên chip STM32 của board điều khiển: thực hiện chức năng giao tiếp, nhận lệnh từ PC để điều khiển chip CAMC-QI, từ đó xuất xung điều khiển hoạt động robot.
- Phần mềm tương tác với người dùng ở PC: phần mềm có chức năng điều khiển robot chạy thực tế và tích hợp tính năng mô phỏng

## Chương 4

# THIẾT KẾ PHẦN CỨNG VÀ TỦ ĐIỆN ĐIỀU KHIỂN

### 4.1 Giới thiệu về các chi tiết phần cứng

#### 4.1.1 Robot SCARA

1. Giới thiệu về Robot SCARA: SCARA là viết tắt của "Selective Compliant Articulated Robot Arm" được ra đời vào năm 1979 tại trường đại học Yamanashi (Nhật Bản). SCARA là một robot mới nhằm đáp ứng sự đa dạng của các quá trình sản xuất, đặc biệt là sử dụng trong các công việc lắp ráp tải trọng nhỏ theo phương thẳng đứng. Robot SCARA là Robot gồm có 4 trục bao gồm 3 trục xoay và 1 trục tịnh tiến.



Hình 4.1: Robot SCARA

## 2. Các thông số động cơ AC Servo của robot.

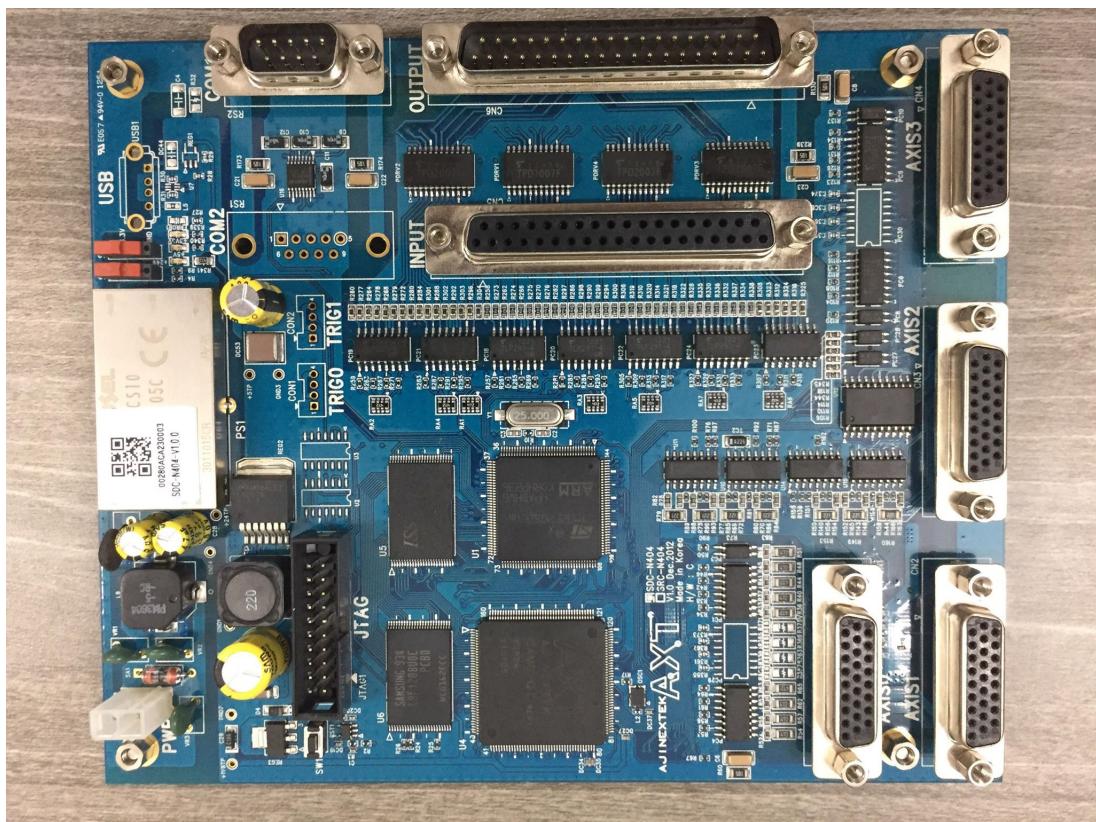
Robot SCARA là robot 4 trục tự do, mỗi khớp được điều khiển chuyển động bằng một Harmonic AC Servo.

STT	Hãng SX	Tên Servo motor	Đặc tính kỹ thuật	TS truyền
1	Mitsubishi	HC-KFS23	200W[0,9N.m]3000rpm[1.1A]200V	80:1
2	Yaskawa	SGMAH-01AAA21	100W[0,318N.m]3000rpm[0,9A]200V	50:1
3	Yaskawa	SGMAH-01AAA21	100W[0,318N.m]3000rpm[0,9A]200V	1:5
4	Yaskawa	SGMAH-01AAA21	100W[0,318N.m]3000rpm[0,9A]200V	8:1

Bảng 4.1: Bảng thông số động cơ của Robot

### 4.1.2 Board điều khiển

Board điều khiển standalone sử dụng chip ARM-Cortex M3 để điều khiển chuyển động của 4 trục. Board điều khiển có chức năng xuất xung để điều khiển robot, đồng thời có chức năng thực hiện chức năng bật tắt Output là tín hiệu đóng, mở bàn kẹp.



Hình 4.2: Board điều khiển

	Specification
CPU	ARM 32-bit Cortex-M3 CPU 12MHz
RAM	2MB SRAM
Flash	64MB Nand Flash Memory
Communication	2 x RS232C(COM1, COM2)
Language	AXTPL(Ajniextek Program Laguage)
Motion	Motion 4 axis with 26 pin DBUS connector x 4
Digital Input/Output	(37 pin DSBUS Connector x 2) 27 Input/27 Output
Motion I/O	8 Input/8 Output
Power Requirement	24VDC(18-26VDC), Max 3.0A, Typical 2.5A

Bảng 4.2: Thông số kỹ thuật của board điều khiển

Thông số kỹ thuật của bộ điều khiển:

Bộ điều khiển được nối với 2 Terminal để đưa tín hiệu ra cho Servo Driver cũng như nhận các tín hiệu trả về. Mỗi Terminal dùng để kết nối với 2 trực.



Hình 4.3: Terminal kết nối board điều khiển và Servo Driver

#### 4.1.3 Servo Driver

##### 1. Khái niệm:

Servo Drive là một bộ khuếch đại điện tử đặc biệt được sử dụng để cung cấp điện servomechanisms. Nó theo dõi phản hồi từ cơ chế servome và liên tục điều chỉnh độ lệch từ các hành vi dự kiến.

##### 2. Chức năng:

Một servo drive nhận được tín hiệu lệnh từ một hệ thống điều khiển, khuếch đại tín hiệu, và truyền dòng điện cho một động cơ servo để tạo ra chuyển động tỉ lệ với tín hiệu lệnh. Thông thường, tín hiệu lệnh đại diện cho vận tốc mong muốn nhưng cũng có thể biểu diễn một momen hoặc vị trí mong muốn. Một cảm biến gắn vào động cơ thực tế phản hồi tình trạng thực tế của động cơ quay trở về servo drive. Động cơ servo sau đó so sánh trạng thái động cơ thực tế với tình

trạng động cơ được chỉ định. Sau đó thay đổi tần số điện áp hoặc độ rộng xung tới động cơ để sửa lỗi cho bất kì độ lệch nào từ trạng thái lệnh.

Trong một hệ thống điều khiển được cấu hình đúng cách, động cơ servo quay với **vận tốc** rất gần với tín hiệu vận tốc nhận được từ hệ thống điều khiển.

### 3. Servo Driver được sử dụng trong hệ thống:

Robot SCARA có 4 động cơ AC Servo nên cần 4 driver để điều khiển. Mỗi driver điều khiển một động cơ. Do đó, cần phải lựa chọn driver phù hợp với động cơ AC Servo. Hệ thống sử dụng 1 servo driver của hãng Mitsubishi MR-J2S-20A và 3 servo driver của hãng Yaskawa SGDH-01AE.



Hình 4.4: Servo Driver điều khiển 4 trục robot

Thông số kỹ thuật của Servo Driver Mitsubishi MR-J2S-20A:

- Điện áp nguồn cấp: 3 pha 200VAC hoặc 1 pha 230VAC.
- Điện áp ra: 3 pha 200-230 V.
- Công suất 200W.
- Loại Servo: Amplifier
- Loại motor tương thích: HC-KFS23, HC-KFS23B.
- Phương pháp điều khiển: điều chế xung PWM, điều chỉnh dòng điện.

Thông số kỹ thuật của Servo Driver Yaskawa SGDH-01AE:

- Điện áp nguồn cấp: 1 pha 230VAC hoặc 3 pha 200-230 VAC.
- Điện áp ra: 3 pha 200 VAC.
- Công suất: 100W.

- Dòng điện: 0.89A.
- Phương pháp điều khiển: điều chế xung PWM, điều chỉnh dòng điện.

#### 4.1.4 Relay

Tín hiệu Output xuất ra từ board điều khiển có dòng nhỏ nên không thể trực tiếp dùng tín hiệu để điều khiển van solenoid. Để có thể đóng, cắt được van thì phải sử dụng một relay trung gian để kích hoạt.



Hình 4.5: Module relay điều khiển van solenoid

Module Relay với opto cách ly nhỏ gọn, có opto và transistor cách ly giúp cho việc sử dụng trở nên an toàn hơn với board mạch chính, mạch được sử dụng để đóng ngắt nguồn điện công suất cao AC hoặc DC, có thể chọn đóng khi kích mức cao hoặc thấp bằng jumper.

Tiếp điểm đóng ngắt gồm 3 tiếp điểm chính là: NC(thường đóng), NO(thường mở), và COM(chân chung) được cách ly an toàn với board mạch chính, ở trạng thái bình thường khi chưa kích NC sẽ nối với COM, khi có tín hiệu kích thì COM sẽ chuyển sang tiếp xúc với NC.

Các thông số kỹ thuật của module relay:

- Sử dụng điện áp nuôi 24VDC.
- Relay tiêu thụ dòng khoảng 80mA.
- Điện thế đóng ngắt tối đa: AC250V - 10A hoặc DC30V - 10A.
- Có đèn báo đóng, ngắt trên module.
- Có thể chọn mức tín hiệu kích hoạt là 0 hoặc 1.

#### 4.1.5 Van Solenoid

Van điện từ khí nén hay còn gọi là van điều khiển bằng điện. Nó cho phép thực hiện đóng, mở van một cách tự động thay vì sử dụng bằng tay.

Để điều khiển tay kẹp của robot đóng, mở, trong luận văn sử dụng van solenoid 5/2 của hãng AIRTAC.

Van solenoid được sử dụng là loại van 5 cửa: 1 cửa vào, 2 cửa ra và 2 cửa để xả khí dư. 2 đầu ra của khí tương đương với 2 trạng thái đóng, mở tay kẹp.



Hình 4.6: Van solenoid

## 4.2 Thiết kế tủ điện điều khiển

Tủ điện được thiết kế bao gồm các chi tiết phần cứng đã nêu. Bên cạnh đó còn sử dụng các thiết bị khác để xây dựng được tủ điện hoàn chỉnh.

### 4.2.1 Hệ thống nguồn

- CB dùng để cấp nguồn 220VAC cho hệ thống.



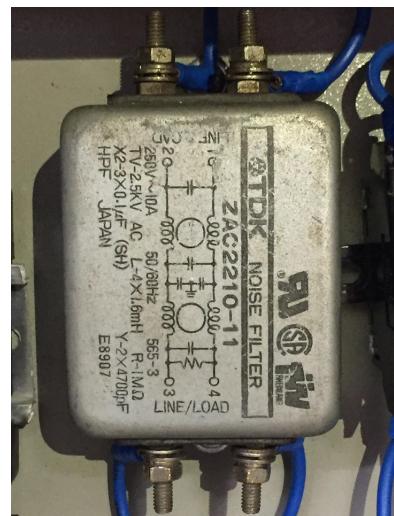
Hình 4.7: CB cấp nguồn cho hệ thống

- Contactor dùng để thực hiện đóng mạch khi bật CB nhằm giảm các gai nhiễu sinh ra khi vừa mới bật nguồn, giúp hệ thống hoạt động ổn định.



Hình 4.8: Contactor

- Bộ lọc nhiễu nguồn AC 220V giúp nguồn đầu vào của driver ổn định hơn và lọc các sóng nhiễu từ động cơ vào hệ thống. Bộ lọc giúp các thiết bị xung quanh hoạt động ổn định, tăng độ chính xác và tuổi thọ.



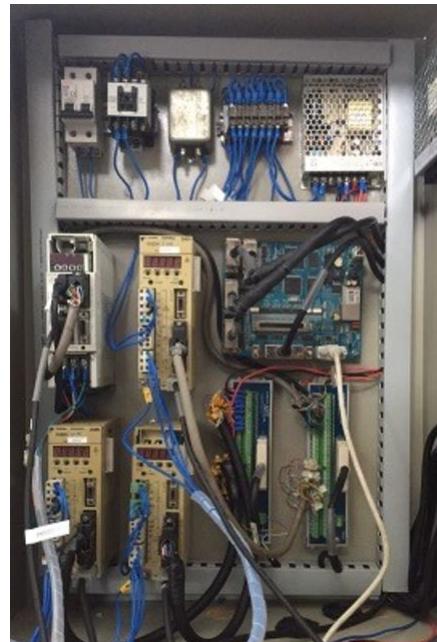
Hình 4.9: Bộ lọc nhiễu nguồn

- Nguồn DC 24V để cấp nguồn cho board điều khiển.



Hình 4.10: Nguồn 24VDC cấp cho board điều khiển

#### 4.2.2 Tủ điện sau khi được thiết kế



Hình 4.11: Tủ điện điều khiển

# Chương 5

## THIẾT KẾ PHẦN MỀM ĐIỀU KHIỂN VÀ MÔ PHỎNG ROBOT SCARA

### 5.1 Phần mềm trên board điều khiển

Board điều khiển có chip xử lý trung tâm là ARM-Cortex M3, để lập trình được hoạt động của board, phần mềm trên board điều khiển được lập trình bằng phần mềm Keil C.

#### 5.1.1 Giới thiệu về phần mềm Keil C

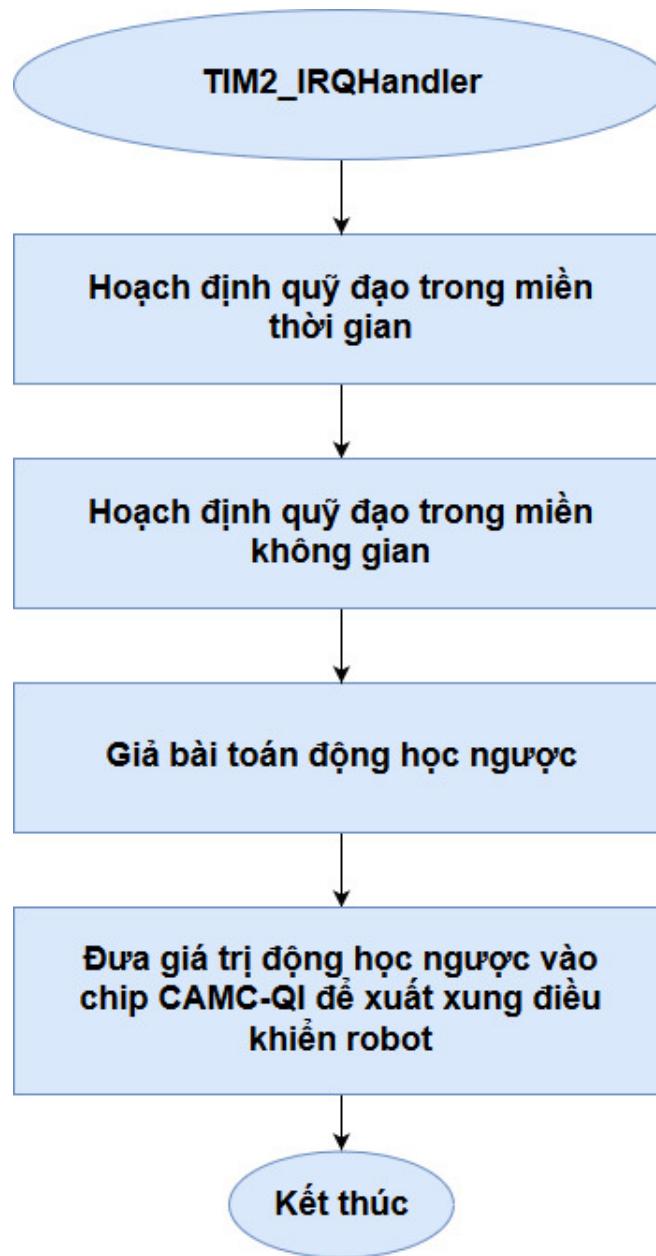


Hình 5.1: Phần mềm Keil C uVision 5

Keil C là một phần mềm hỗ trợ cho người dùng lập trình vi điều khiển các dòng khác nhau. Keil C giúp người dùng soạn thảo và biên dịch chương trình C hay ASM thành ngôn ngữ máy để nạp vào vi điều khiển giúp chúng ta tương tác giữa vi điều khiển và người lập trình.

Trong chương trình của board điều khiển sẽ có một số hàm chính liên quan đến việc truyền nhận dữ liệu cũng như là tạo ra các profile chuyển động cho robot. Một số chương trình ngắn được sử dụng trong chương trình bao gồm: TIM2\_IRQHandler, TIM1\_IRQHandler, TIM3\_IRQHandler, TIM5\_IRQHandler, DMA1\_Stream3\_IRQHandler.

### 5.1.2 Chương trình ngắt timer 2

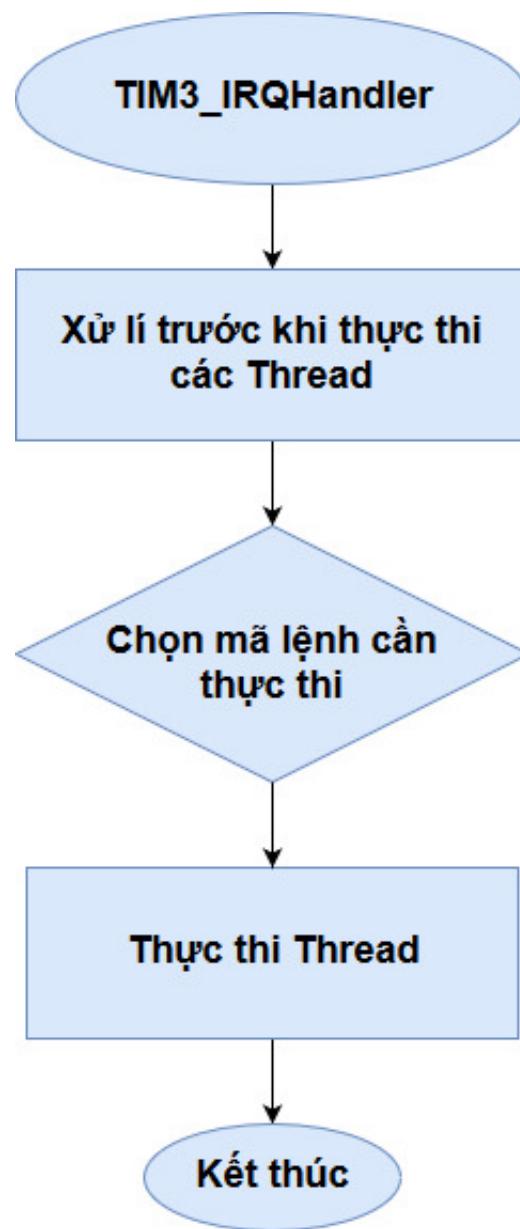


Hình 5.2: Chương trình thực thi trong ngắt Timer 2 của chip STM

Giải thích sơ đồ chương trình ngắt:

Khi vào chương trình, chương trình thực hiện quá trình hoạch định quỹ đạo trong miền thời gian. Nói cách khác là đi tính các giá trị về vị trí, vận tốc và gia tốc của robot trong quá trình di chuyển trên quỹ đạo. Sau đó thực hiện quá trình hoạch định quỹ đạo trong miền không gian làm việc để tạo ra vector vị trí trong không gian làm việc tại mỗi thời điểm lấy mẫu. Dựa vào vector vị trí của robot, sử dụng phương trình động học ngược để tìm ra các giá trị biến khớp và đưa vào chip CAMCI-QI để xuất xung ra Servo Driver.

### 5.1.3 Chương trình ngắt timer 3

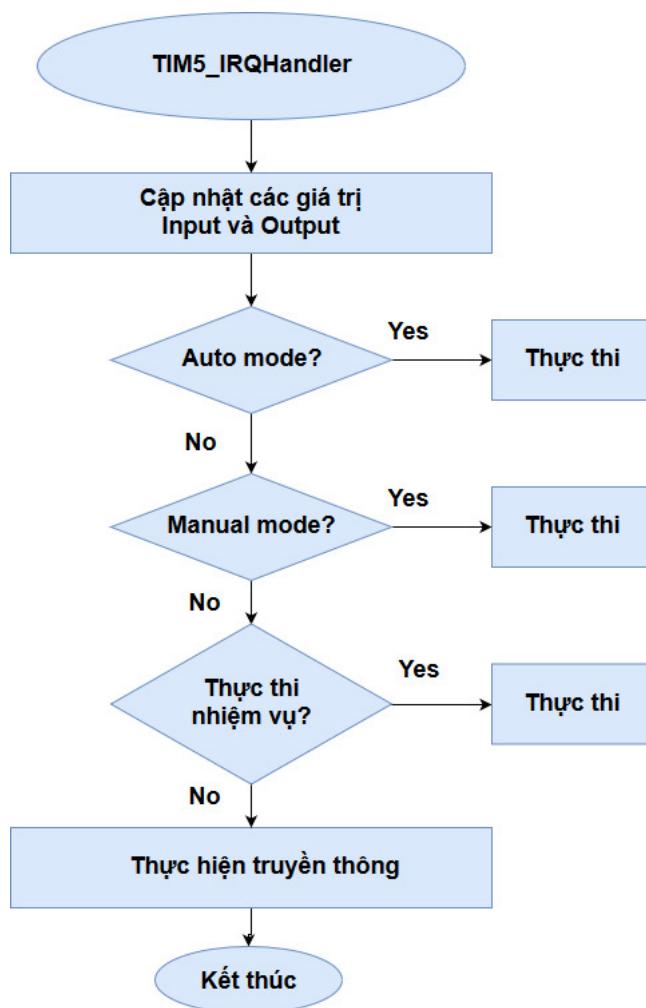


Hình 5.3: Chương trình thực thi trong ngắt Timer 3 của chip STM

Giải thích sơ đồ chương trình ngắt:

Khi vào chương trình ngắt, Timer sẽ tiền xử lí các thread cần thực thi, sau đó dựa vào mã lệnh sau khi được giải mã để lựa chọn lệnh tương ứng cần thực hiện và cho thực thi thread tương ứng.

#### 5.1.4 Chương trình ngắt timer 5



Hình 5.4: Chương trình thực thi trong ngắt Timer 5 của chip STM

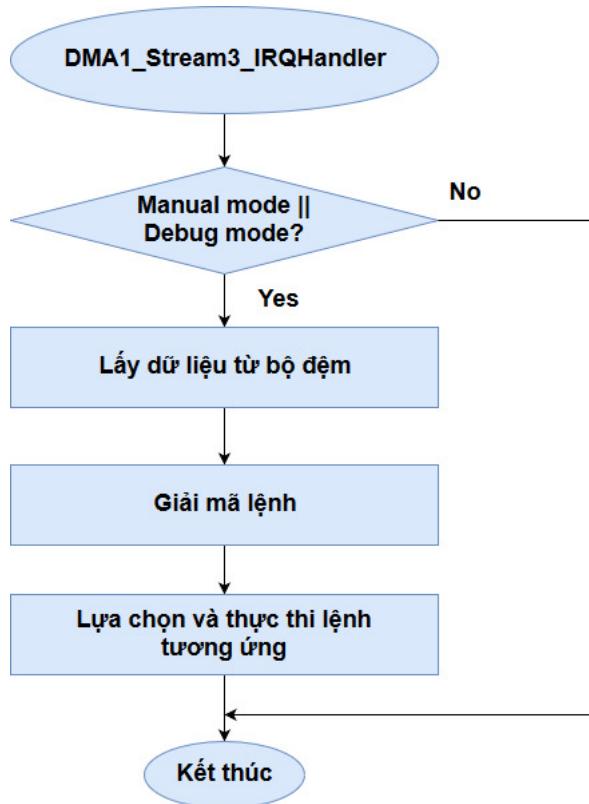
Giải thích sơ đồ chương trình ngắt Timer 5:

Timer 5 có nhiệm vụ quét các trạng thái cũng như là dữ liệu của hệ thống, cùng với đó là dữ liệu nhận về từ máy tính.

Sau đó kiểm tra lệnh ở chế độ Auto mode hay Manual mode, có thực thi chương trình hay không, và cuối cùng là thực hiện truyền thông với máy tính. Do đó, dữ liệu luôn được cập nhật mới nhất.

#### 5.1.5 Chương trình ngắt truyền thông USART3

USART3 sử dụng ngắt DMA1\_Stream3\_IRQHandler để thực hiện quá trình giao tiếp với máy tính.



Hình 5.5: Chương trình thực thi trong ngắt truyền thông của chip STM

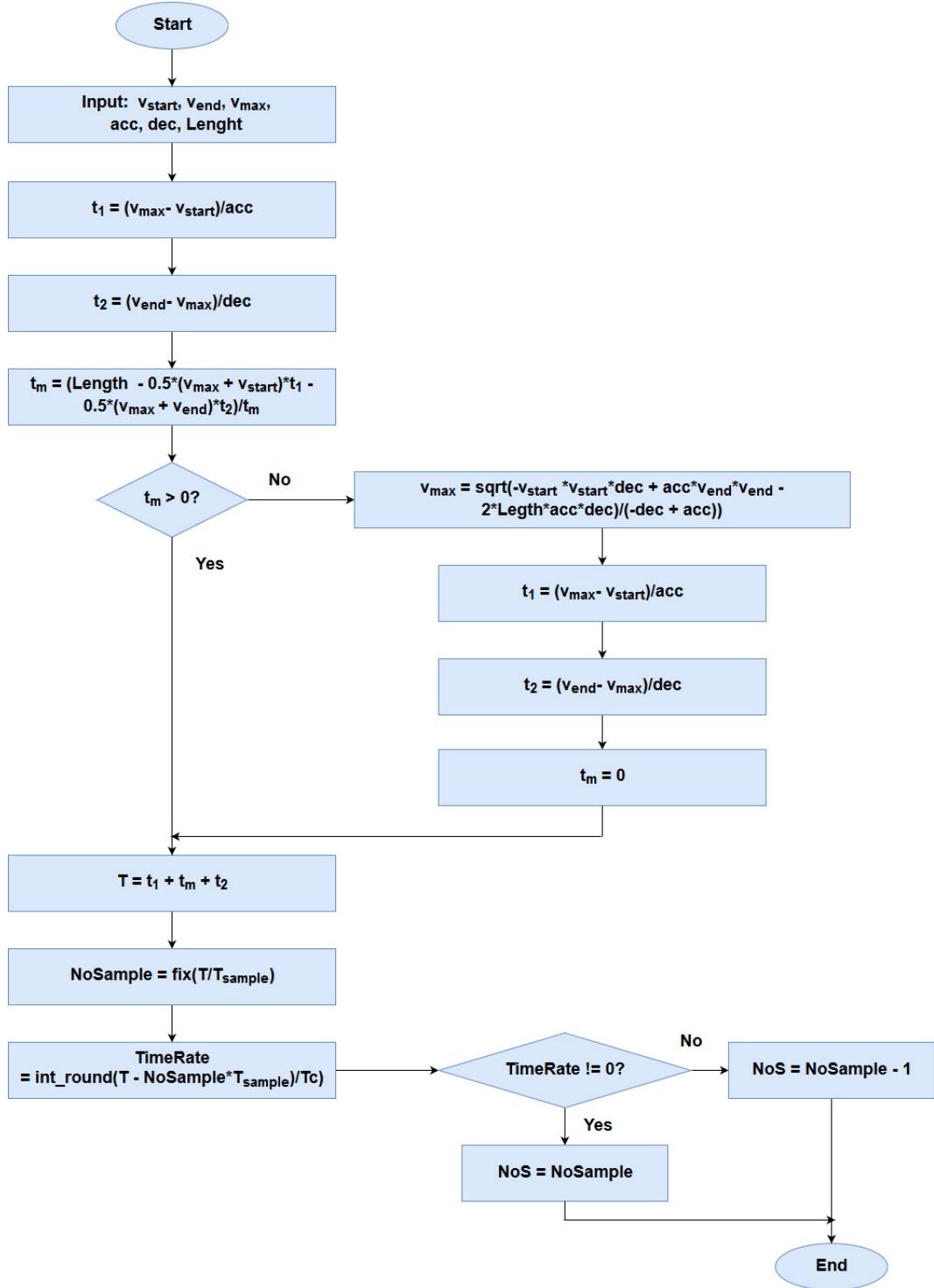
Giải thích sơ đồ chương trình ngắn:

Khi có sự kiện ngắt xảy ra, board điều khiển nhận được lệnh từ máy tính. Đầu tiên, CPU sẽ kiểm tra đang hoạt động ở chế độ nào. Nếu hoạt động ở chế độ Manual mode hoặc chế độ Debug mode, CPU thực hiện việc lấy dữ liệu từ bộ đệm, sau đó thực hiện giải mã lệnh và cuối cùng là thực thi công việc tương ứng với lệnh nhận được từ máy tính gửi về.

### 5.1.6 Lập trình thuật toán hoạch định quỹ đạo trên board điều khiển

Lí thuyết về hoạch định quỹ đạo đã được trình bày trong chương 2. Do vậy, phần này tập trung vào việc lập trình thuật toán đã nêu ra ở phần trước. Sau đó đưa thuật toán vào board điều khiển để tạo ra chuyển động cho robot. Các quá trình hoạch định quỹ đạo trong miền thời gian, không gian và động học ngược lần lượt được thực hiện trong chương trình ngắn của Timer 2.

Trước khi thực hiện quá trình di chuyển trên quỹ đạo, các thông số thời gian phải được xác định trước. Quá trình tính toán các thông số thời gian được trình bày như trong sơ đồ hình 5.7.

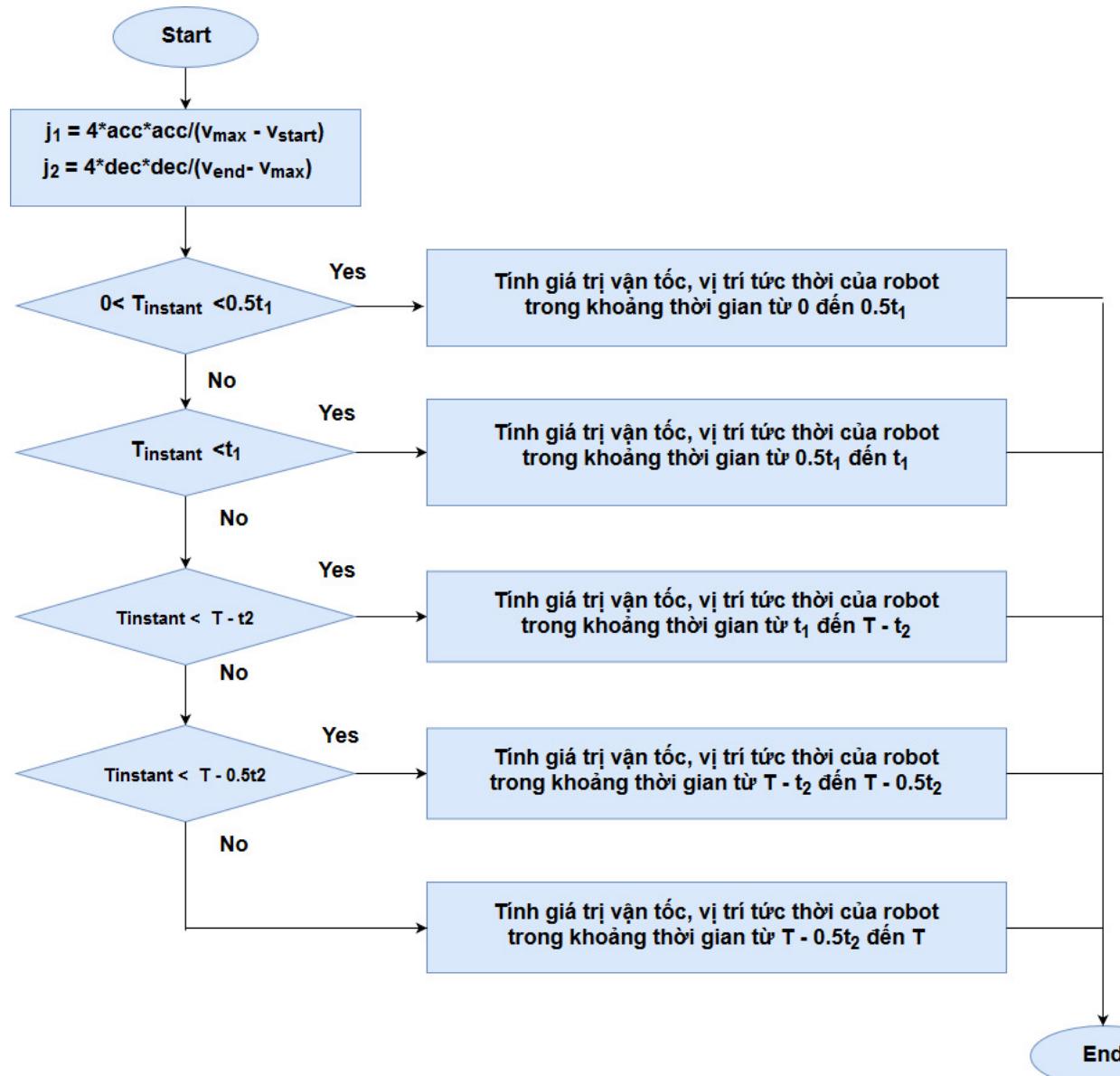


Hình 5.6: Tính toán các giá trị thời gian của profile chuyển động

Giải thích thuật toán: Với giá trị được đưa vào trên quỹ đạo, tính được các thông số thời gian trên đoạn tăng tốc, di chuyển đều và đoạn giảm tốc. Trường hợp không có đoạn di chuyển đều, vận tốc tối đa được tính lại. Sau đó tính lại thời gian các đoạn

tăng tốc, giảm tốc và thời gian trên đoạn di chuyển đều bằng 0. Giá trị thời gian sau khi tính xong được xử lí làm tròn nhằm tạo nên sai số ít nhất trong quá trình hoạch định quỹ đạo. Như vậy, các giá trị thời gian trên mỗi đoạn đã được xác định.

1. Hoạch định quỹ đạo trong miền thời gian:



Hình 5.7: Hoạch định quỹ đạo trong miền thời gian

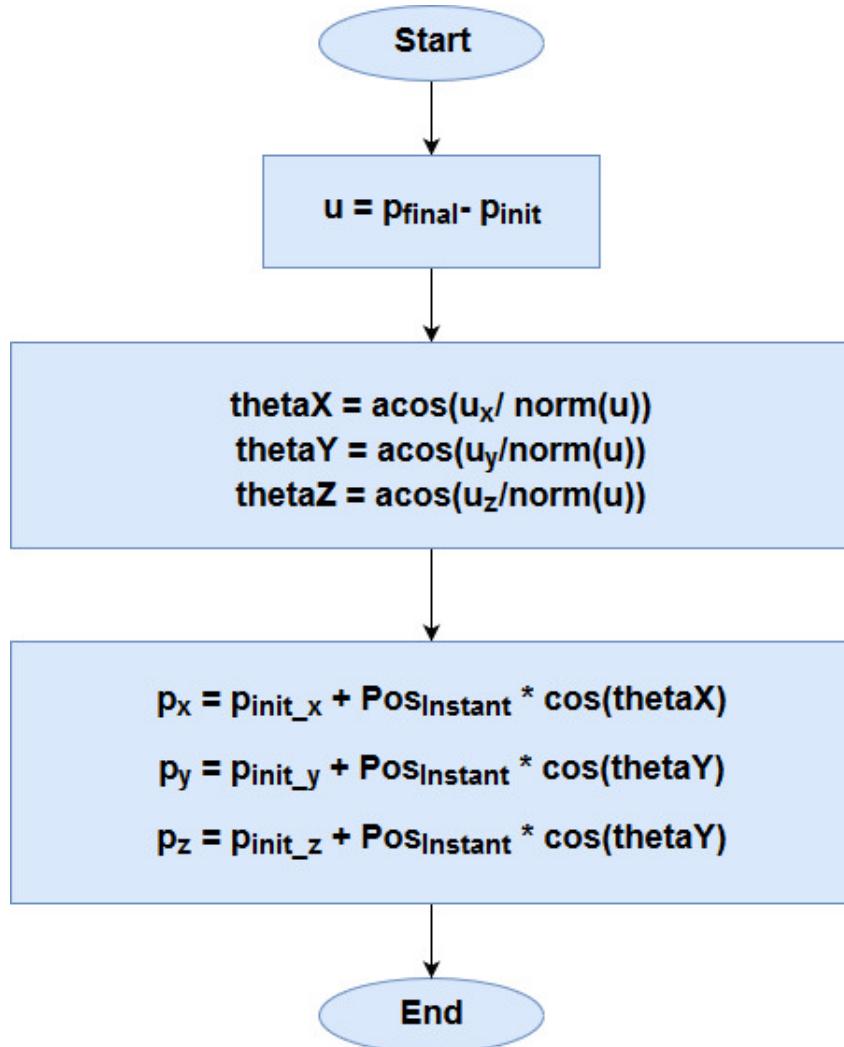
Giải thích thuật toán: Với các thông số thời gian vừa tính được ở trên, chia quỹ đạo thành nhiều điểm tương ứng với mỗi thời gian lấy mẫu, áp dụng vào phương trình của profile vận tốc và vị trí, kết quả thu được là giá trị vận tốc và vị trí tức thời tại mỗi thời điểm xảy ra ngắt timer.

## 2. Hoạch định quỹ đạo trong miền không gian

Tùy vào dạng đường đi mà hoạch định quỹ đạo khác nhau, dưới đây là giải thuật lập trình hoạch định đường đi là đường thẳng đi qua 2 điểm và đường tròn đi qua 3 điểm trong không gian.

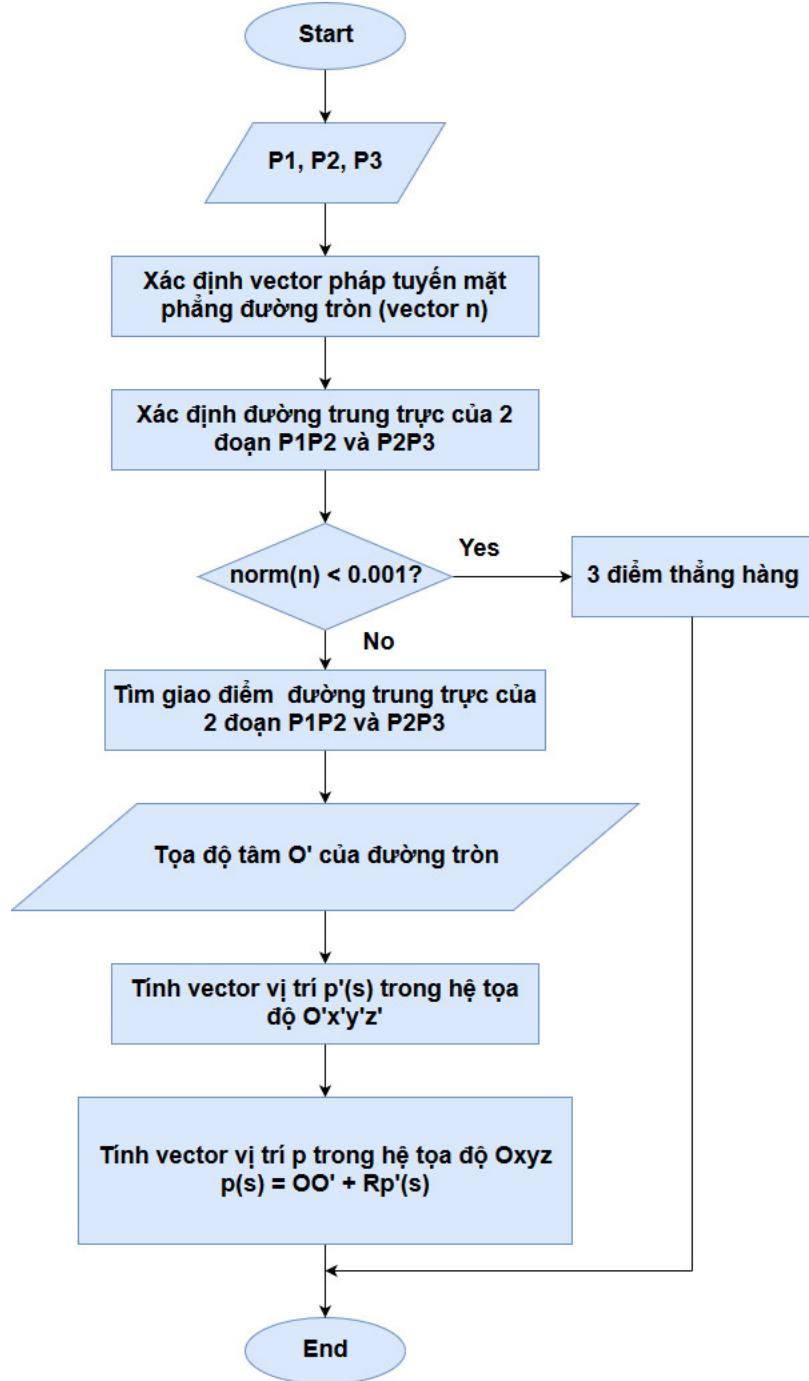
Từ phương trình tham số của 2 dạng đường đi này được trình bày ở chương 3, thuật toán hoạch định đường đi của 2 loại đường này được biểu diễn như sau:

- Đường thẳng:



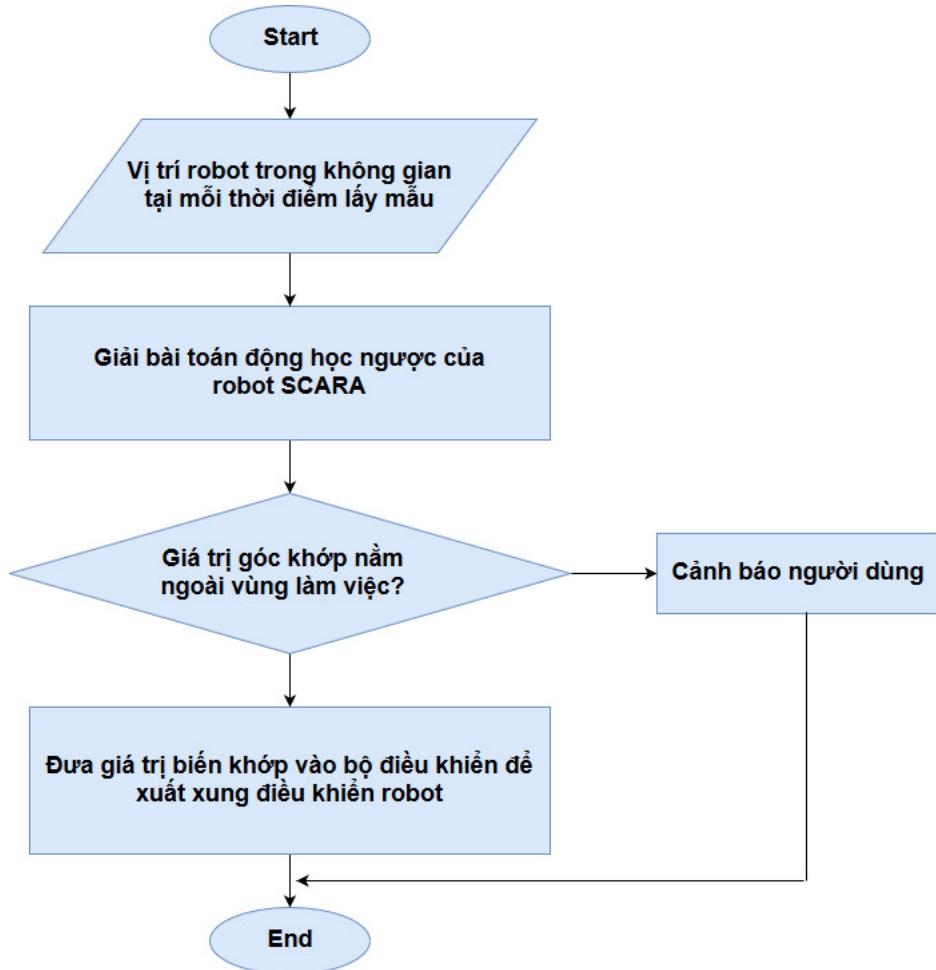
Hình 5.8: Hoạch định quỹ đạo đường thẳng

- Đường tròn đi qua 3 điểm:



Hình 5.9: Hoạch định quỹ đạo đường thẳng

3. Giải bài toán động học ngược:



Hình 5.10: Giải bài toán động học ngược

## 5.2 Phần mềm trên máy tính

### 5.2.1 Giới thiệu phần mềm Visual Studio 2008 và thư viện MFC

Visual Studio 2008 là một môi trường phát triển tích hợp(IDE), nó được dùng để phát triển ứng dụng máy tính, cũng như các trang web, ứng dụng web và các dịch vụ web. Visual Studio sử dụng nền tảng từ Window API, Windows Forms, Windows Store và Microsoft Silverlight. Nó là trình soạn thảo mã hỗ trợ IntelliSense cũng như cài tiến mã nguồn. Visual Studio hỗ trợ nhiều ngôn ngữ lập trình khác nhau, cho phép trình biên dịch mã và gỡ lỗi để hỗ trợ hầu như mọi ngôn ngữ lập trình. Các ngôn ngữ tích hợp gồm C, C++, VB.Net, C#, cùng nhiều ngôn ngữ khác nhau nhờ vào việc cài đặt các gói ứng dụng riêng lẻ.

Thư viện MFC (Microsoft Foundation Class) là một tính năng nổi bật được cung cấp bởi Visual Studio. MFC là một thư viện các lớp (class, OOP) trong ngôn ngữ C++, dùng cho việc lập trình trên Window. Nó được xây dựng dựa trên cơ sở các hàm thư viện API của Window. Người lập trình có thể xây dựng ứng dụng nhanh và ít tốn công sức hơn khi sử dụng MFC so với việc sử dụng đơn thuần các hàm thư viện API của Windows. Trong lập trình bằng thư viện MFC, ta vẫn có thể gọi các hàm Window API.

Trong một ứng dụng MFC, người lập trình không gọi hàm Window API trực tiếp mà sẽ tạo các object từ những lớp được xây dựng sẵn trong thư viện MFC và gọi những phương thức của object đó. Đa số các phương thức của MFC Class có cùng tên với những hàm Windows API.

MFC tạo ra một Application Framework giúp:

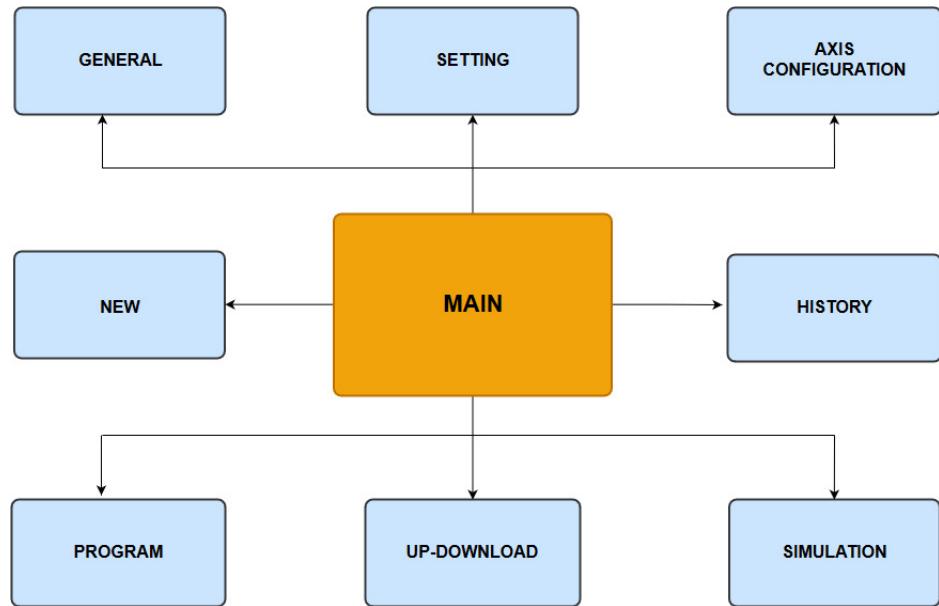
- Thiết lập kiến trúc của một ứng dụng một cách nhất quán và khoa học.
- Che giấu đi nhiều chi tiết mà Windows API đòi hỏi, giúp người lập trình dễ sử dụng và đỡ tốn thời gian hơn.

Do vậy, để xây dựng nên một phần mềm điều khiển và mô phỏng robot trên máy tính, cũng như tạo ra một giao diện sinh động, trực quan cho người sử dụng, **tôi sử dụng thư viện MFC của Microsoft để lập trình bằng phần mềm Visual Studio.**

### 5.2.2 Giới thiệu về phần mềm đã thiết kế

Chương trình được viết bằng ngôn ngữ C++ trên nền tảng MFC, sau khi thiết kế, chương trình có các chức năng như sau:

- Điều khiển robot:
  - Giao tiếp được với board điều khiển SDC-N0404 thông qua giao tiếp RS232C.
  - Thiết lập các thông số điều khiển trên từng trục của bộ điều khiển.
  - Điều khiển robot bằng chế độ Jog Mode.
  - Lưu lại những vị trí robot cần đi qua sau đó lập trình cho robot theo kiểu dạy học.
  - Có thể xây dựng chương trình cho robot bằng ngôn ngữ lập trình robot.
  - Quan sát vị trí của các trục trong quá trình chạy.
- Mô phỏng robot:
  - Các chế độ Jog Mode: Joint, Base, Tool.
  - Mô phỏng hoạt động robot theo kiểu dạy học đi qua nhiều điểm.
  - Đồ thị quan sát chuyển động lúc chạy mô phỏng.
  - **Mô phỏng robot thực hiện 1 ứng dụng sắp xếp vật vào các hộp.**



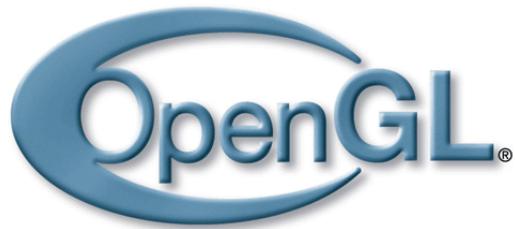
Hình 5.11: Sơ đồ khái niệm của chương trình

Sơ đồ khái niệm thể hiện cấu trúc của chương trình theo hướng đối tượng. Chương trình được tạo thành từ các Dialog, mỗi dialog sẽ được quản lý bởi một class khác nhau. Trên mỗi Dialog bao gồm các Control như: Button, Edit Control, Combo Box,... với sự kiện kích hoạt khác nhau. Mỗi khái niệm trong sơ đồ thực hiện một chức năng riêng để cấu tạo nên chương trình hoàn chỉnh. Khi thực hiện chương trình, khái niệm MAIN sẽ được khởi tạo đầu tiên, sau đó tùy thuộc vào chức năng mà đi đến các hộp thoại khác nhau. Class quản lý khái niệm MAIN chính là Parent Class(lớp mẹ). Các class quản lý ở các dialog thấp hơn được gọi là Child Class (lớp con). Từ lớp mẹ, có thể truy cập đến từng thành phần trong từng lớp con theo cấu trúc hướng đối tượng. Ngược lại, để truy cập đến một class mẹ từ class con có thể sử dụng phương thức HWND GetParent().

## 5.3 Giới thiệu về OpenGL

### 5.3.1 Giới thiệu chung

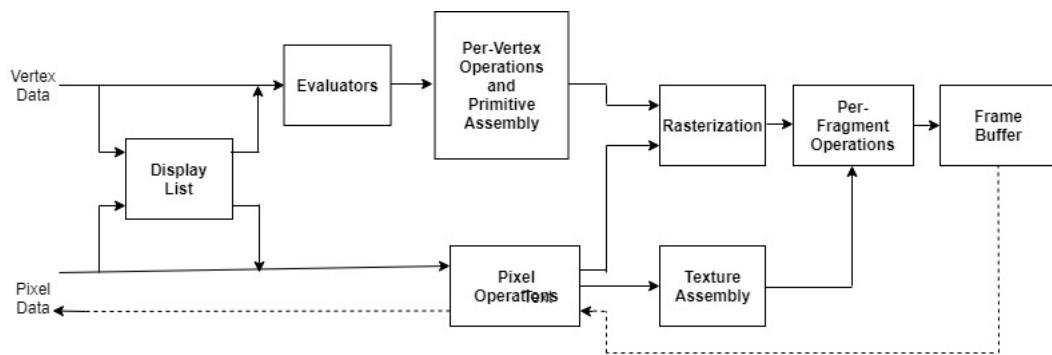
OpenGL (Open Graphic Library) là một thư viện đồ họa tốc độ cao và độc lập với hệ thống giao diện các hệ điều hành. Tiền thân của OpenGL là IRIS GL do hãng **Silicon Graphic Library Inc** phát triển cho các WorkStation đồ họa tốc độ cao từ năm 1982. Sau đó từ năm 1992 **thì** OpenGL đã trở thành một chuẩn công nghiệp và đặc tính kỹ thuật của OpenGL do Ủy ban kỹ thuật ARB (Architecture Review Board) phê chuẩn.



Hình 5.12: OpenGL

Công nghệ đồ họa đang ngày càng được ứng dụng rộng rãi trong cuộc sống và ngày nay nó đã phát triển tương đối mạnh mẽ ở Việt Nam. Trong kĩ thuật thì việc sử dụng các ứng dụng của OpenGL cũng DirectX vào việc mô phỏng các cơ cấu máy móc, các hoạt động của robot công nghiệp trước khi thực hiện mô hình thực tế là một giải pháp được các kĩ sư cũng như các viện nghiên cứu lựa chọn do tính ưu việt của nó trong việc thể hiện các mô hình đồ họa động trong không gian 3 chiều.

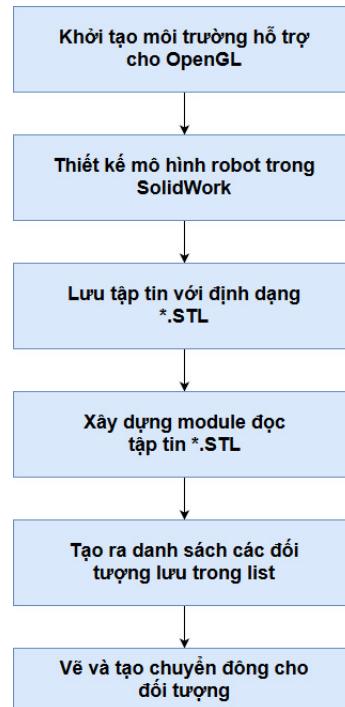
Cơ chế hoạt động của OpenGL:



Hình 5.13: Cơ chế hoạt động của OpenGL

OpenGL có cơ chế hoạt động theo kiểu ống dẫn tức là đầu ra của giai đoạn trước là **đầu** vào của giai đoạn sau. Từ sơ đồ, các giai đoạn của cơ chế hoạt động có chức năng:

- Display List: là nơi lưu lại một số lệnh để xử lí sau.
- Evaluators: Xấp xỉ các đường cong và mặt phẳng hình học bằng cách đánh giá các đa thức của dữ liệu đưa vào.
- Per-vertex operations and primitives assembly: Xử lí các primitives (**điểm** **đoạn**, **đa giác**) được mô tả bởi các vertex. Các vertex sẽ được xử lí và các primitives được cắt xén vào viewport để chuẩn bị cho khâu kế tiếp.



Hình 5.14: Quá trình xây dựng tính năng đồ họa

- Rasterization: sinh ra một loạt các địa chỉ framebuffer và các giá trị liên quan bằng cách sử dụng mô tả 2 chiều của điểm, đoạn, đa giác. Mỗi phần tử (fragment) được sinh ra sẽ đưa vào giai đoạn kế tiếp.
- Per-fragment operations: Các tác vụ sau cùng (cập nhật có điều kiện cho framebuffer dựa vào dữ liệu vào và dữ liệu được lưu trữ trước đó của giá trị z (đối với z buffering)), thực hiện trộn màu cho các pixel và làm một số thao tác khác) sẽ được thực hiện trên dữ liệu trước khi nó được chuyển thành pixel và đưa vào framebuffer.

Trong trường hợp dữ liệu vào ở dạng pixel không phải dạng vertex, nó sẽ đưa thẳng qua giai đoạn xử lý pixel. Sau giai đoạn này, dữ liệu ở dạng pixel sẽ được lưu trữ vào texture memory để đưa vào giai đoạn per-fragment operation hoặc đưa vào Rasterization như dữ liệu dạng Vertex (tức là các điểm).

### 5.3.2 Xây dựng tính năng đồ họa sử dụng OpenGL

Quá trình xây dựng chức năng đồ họa sử dụng OpenGL được thể hiện như trong hình 5.14

- Khởi tạo môi trường hỗ trợ cho OpenGL

OpenGL là một thư viện đồ họa độc lập với hệ thống giao diện của các hệ điều hành, do đó các ứng dụng đồ họa sử dụng OpenGL trên các hệ điều hành khác nhau đều phải có một quá trình khởi tạo thích hợp với các hàm xử lý đồ họa của

OpenGL. Quá trình này được thực hiện bằng việc gọi một số hàm thích hợp của thư viện hỗ trợ giao tiếp với OpenGL trong chương trình ứng dụng. Quá trình khởi tạo bao gồm: định nghĩa một kiểu cấu trúc dữ liệu đồ họa phù hợp với quản lý thông tin về định dạng điểm ảnh và lựa chọn các thông số phù hợp phù hợp với hệ thống. Các thông số này bao gồm: kiểu điểm ảnh (RGBA hay color index), bộ đệm hoán đổi đơn hay kép (single/double buffer), độ phân giải màu sắc, khả năng hỗ trợ bộ đệm chiều sâu (Depth buffer), bộ đệm stencil (stencil buffer), bộ đệm tích lũy (accumulation buffer). Dưới đây là kiểu cấu trúc dữ liệu đồ họa trên các hệ điều khác nhau:

- Điều khiển Rendering (Control Rendering)

Sau khi thiết lập định dạng điểm ảnh phù hợp với hệ thống, tất cả các hệ thống đều phải thực hiện việc điều khiển quá trình tô vẽ (rendering). Quá trình này thực hiện các việc sau: Tạo và sử dụng ngữ cảnh tô vẽ (rendering context), thực hiện quá trình đồng bộ hóa (synchronizing execution), hoán đổi bộ đệm (swaping buffer), sử dụng các phông hệ thống...

- Tô vẽ (rendering) là một quá trình tạo và hiển thị điểm ảnh hai chiều lên các thiết bị kết xuất đồ họa như: màn hình máy tính, máy in,... từ các đối tượng đồ họa ba chiều.
- Ngữ cảnh (context): Trong OpenGL, ngữ cảnh là một khái niệm chỉ một tập hợp đầy đủ các biến trạng thái của OpenGL.
- Biến trạng thái: Các trạng thái trong OpenGL như: định dạng điểm ảnh, màu sắc, ánh sáng,... đều được điều khiển bằng các biến trạng thái tương ứng.
- Ngữ cảnh tô vẽ (rendering context): có thể coi ngữ cảnh tô vẽ của OpenGL như một cổng mà tất cả các hàm của OpenGL phải đi qua để giao tiếp với thiết bị đồ họa phần cứng. Ngữ cảnh tô vẽ tạo ra định dạng điểm ảnh giống hệt với định dạng điểm ảnh của ngữ cảnh thiết bị gắn với nó. Tuy vậy, ngữ cảnh tô vẽ không giống với ngữ cảnh của thiết bị, /một ngữ cảnh thiết bị lưu giữ các thông tin của GDI(Graphic Device Interface), còn một ngữ cảnh tô vẽ lưu giữ các thông tin của OpenGL. Về mặt nào đó, chức năng của ngữ cảnh tô vẽ đối với OpenGL cũng giống như chức năng của ngữ cảnh thiết bị đối với GDI. Một ứng dụng có thể có nhiều ngữ cảnh tô vẽ.
- Quá trình đồng bộ hóa: Quá trình này buộc các hàm của OpenGL hay các hàm đồ họa của hệ thống phải chờ cho đến khi hàm trước đó thực hiện xong. Điều này có nghĩa các lệnh của OpenGL ở trước lệnh chờ của hệ thống sẽ được đảm bảo thực hiện trước các lệnh hệ thống nằm sau lệnh chờ đó. Ngược lại các lệnh OpenGL ở phía sau lệnh chờ hệ thống chỉ được thực hiện khi các lệnh của hệ thống nằm ở trước lệnh chờ của OpenGL đã thực hiện xong.
- Hoán đổi bộ đệm (Swaping buffer): Mục đích chính của chương trình đồ họa là hiển thị các hình ảnh lên màn hình. Một hình ảnh chuyển động cần có 24 khung hình/giây. Các hình ảnh trước khi được đưa lên màn hình sẽ được vẽ lên bộ đệm trung gian, sau đó toàn bộ hình ảnh trên bộ đệm mới được

đưa ra màn hình cùng một lúc. Nếu hệ thống chỉ hỗ trợ một bộ đệm thì những phần vẽ đầu của khung hình sẽ tồn tại trong suốt trong 1/24 giây, nhưng các phần vẽ sau của khung hình sẽ bị xóa ngay khi vừa vẽ để chuẩn bị khung hình tiếp theo. Giải pháp để nâng cao chất lượng điểm ảnh và hạn chế nhược điểm trên là sử dụng bộ đệm kép. Khi hình ảnh trên bộ đệm thứ nhất được đưa ra màn hình trong suốt 1/24 giây, thì cùng lúc, hình ảnh cho khung hình tiếp theo được vẽ trên bộ đệm kia. sau 1/24 giây thì các bộ đệm hoán đổi vị trí cho nhau. Quá trình liên tục như vậy sẽ giúp cải thiện chất lượng đồ họa.

#### • Xây dựng đối tượng 3D

Sau khi tạo thành công môi trường hỗ trợ cho OpenGL, cần xây dựng các đối tượng 3D. Một số thư viện mở rộng của OpenGL cung cấp các đối tượng 3 chiều cơ bản như: hình nón, hình tứ diện, hình cầu,... Tuy nhiên, trên thực tế, các đối tượng có cấu tạo phức tạp nhưng cũng có thể xây dựng từ đa giác nhỏ cơ sở.

Việc xây dựng đối tượng 3D phức tạp bằng OpenGL được thực hiện như sau:

- Thiết kế các mô hình 3 chiều bằng một phần mềm hỗ trợ thiết kế chuyên dụng như AutoCAD hay SolidWorks,..
- Sử dụng các lệnh trong phần mềm đó để xuất các đối tượng 3 chiều ra các dạng tập tin nhị phân hay ASCII có chứa thông tin của đối tượng dưới dạng các đỉnh và quy tắc nối các đỉnh đó thành đa giác. Ví dụ các tập tin \*.BDF, \*.STL,...
- Đọc thông tin các đối tượng từ tập tin đó và sử dụng các hàm vẽ đối tượng cơ bản trong OpenGL để xây dựng lại đối tượng trong OpenGL.

Trong luận văn, các chi tiết của robot SCARA sau khi được thiết kế trong phần mềm SolidWorks sẽ được chuyển thành định dạng tập tin \*.STL.

#### • Quan sát đối tượng 3 chiều

Quan sát đối tượng là khái niệm cho phép các đối tượng hình học 3 chiều hiển thị lên màn hình máy tính (hai chiều) với hiệu ứng trong không gian 3 chiều. Quan sát đối tượng **gồm** các thao tác: Xác định vị trí quan sát, hướng quan sát, phạm vi quan sát, xác định vị trí các đối tượng trong không gian 3 chiều, chọn phép chiếu, chọn khung hình,...

Quá trình chuyển đổi các đối tượng trong không gian 3 chiều thành các điểm ảnh trên màn hình máy tính có thể chia thành:

- Các phép biến đổi: biến đổi vị trí quan sát hay vị trí của vật. Vì OpenGL xây dựng vật thể trên mô hình không gian 3 chiều nên các phép biến đổi sẽ được thực hiện thông qua các phép toán trên ma trận. Quá trình thực hiện biến đổi diễn ra như sau: Sau khi khai báo vật thể thông qua các vertex, ta được có được các điểm biểu diễn trong không gian 3 chiều với tọa độ dưới dạng vector 4 phần tử. Để thực hiện di chuyển, quay, phôi cảnh... ta cần áp dụng các thông số như góc quay, độ dời.. tương ứng với thay đổi muốn thực hiện. OpenGL sẽ dùng các thông số biến đổi được nhập vào (được lưu trữ

ở dạng ma trận  $4 \times 4$ ) và thực hiện các phép toán biến đổi trên các vector nhập vào. Kết quả là tọa độ các vertex được nhập sẽ thay đổi tương ứng. Để chọn ma trận thay đổi, trong OpenGL, ta dùng hàm:

```
void glMatrixMode(GLenum mode)
```

Các mode có thể là:

- \* GL\_MODELVIEW : Các phép toán ma trận sau đó sẽ tác dụng lên ma trận modelview.
- \* GL\_PROJECTION : Các phép toán ma trận sau đó sẽ tác động lên ma trận projection.
- \* GL\_TEXTURE : Các phép toán ma trận sau đó sẽ tác động lên ma trận texture.

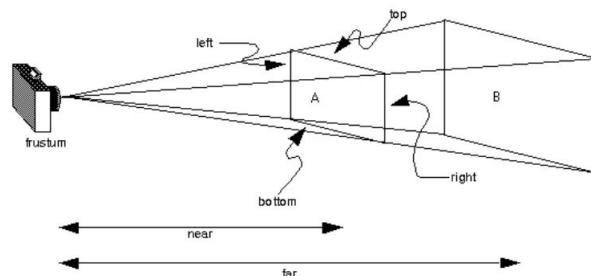
Sau khi gọi hàm và đưa vào tham số tương ứng, ma trận được chọn sẽ trở thành ma trận hiện hành và mọi biến đổi tương ứng sẽ tác động lên ma trận đó.

Để đưa một ma trận biến đổi hiện hành về ma trận đơn vị, dùng hàm :

```
void GLLoadIdentity(void)
```

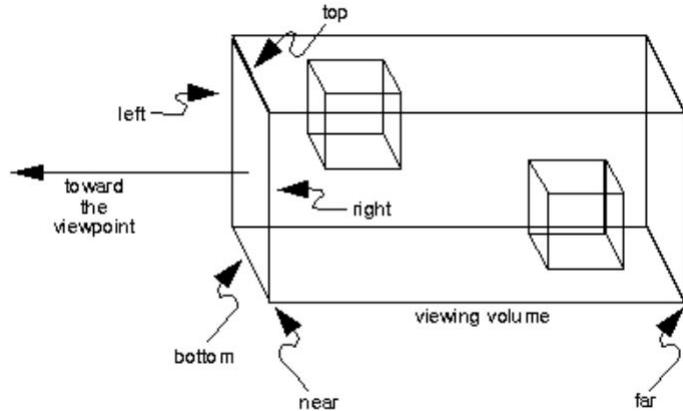
– Các phép chiếu:

- \* Phép chiếu trực giao: Vật thể ở gần ống kính (ở đây là điểm nhìn) thì thấy lớn hơn và ngược lại.



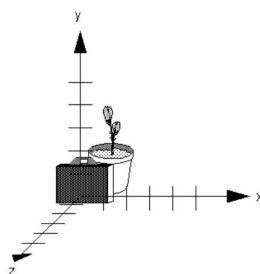
Hình 5.15: Phép chiếu trực giao

- \* Phép chiếu phối cảnh: Các đường thẳng song song trong không gian khi hiển thị lên màn hình cũng sẽ song song với nhau. Nói cách khác, không gian nhìn được xác định bởi một hình hộp chữ nhật.



Hình 5.16: Phép chiếu phối cảnh

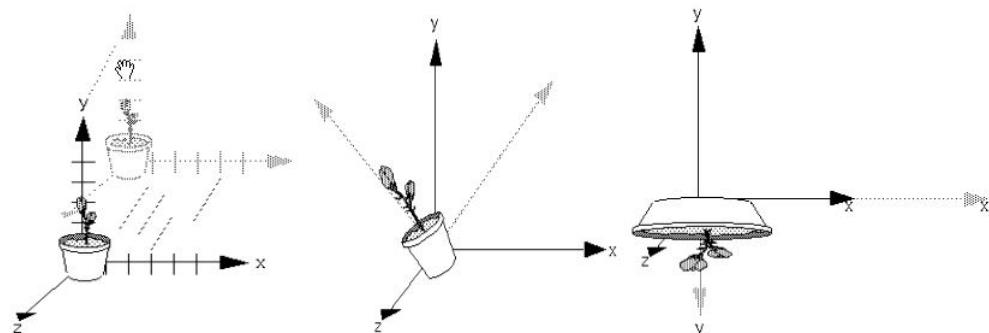
- Chọn khung nhìn (Viewport): Viewport là một dạng khung hình chữ nhật trong cửa sổ chương trình vẽ. Nó là nơi mà vật thể sẽ được vẽ vào (có thể có kích thước nhỏ hơn, bằng hoặc lớn kích thước lớn hơn kích thước cửa sổ chương trình). Hàm void glViewport(GLint x, GLint y, GLsizei width, GLsizei height) xác định một khung nhìn có tọa độ góc trái dưới là (x,y) so với góc trái dưới của sổ và được tính bằng pixel; width và height là kích thước của khung hình. Mặc định, viewport có kích thước bằng kích thước cửa sổ. Tỉ lệ chiều dài và chiều **tông** của viewport nên bằng tỉ lệ tương ứng không không gian nhìn đã xác định thông qua hàm glFrustum() hoặc glOrtho(), nếu không hình ảnh sẽ bị biến dạng.
- Sự biến đổi vị trí quan sát và vị trí vật thể:  
Do sự biến đổi vị trí quan sát và vị trí vật thể có liên quan tương đối với nhau nên trong OpenGL chúng được gom vào một ma trận biến đổi (chỉ dành cho vật thể). Mặc định ban đầu điểm nhìn và vật thể chung một chỗ tại gốc hệ trục tọa độ và phương nhìn theo hướng âm của trục z.



Hình 5.17: Hệ tọa độ chuẩn trong OpenGL

OpenGL cung cấp 3 hàm để thay đổi góc nhìn và hướng nhìn:

- \* glTranslate\*() để thực hiện biến đổi tuyến tính.
- \* glRotate\*() thực hiện phép quay.
- \* glScale\*() dùng để kéo dãn hoặc co nhỏ vật thể.

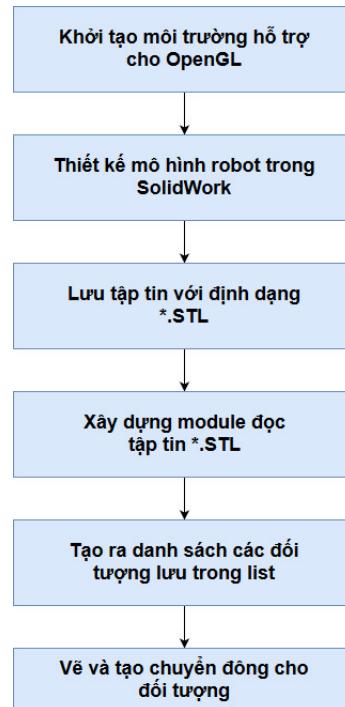


Hình 5.18: Các phép biến đổi trong OpenGL

- Tạo chuyển động cho các đối tượng: Nguyên tắc chung để tạo thành các hình ảnh chuyển động là liên tục vẽ và xóa các vị trí khác nhau của đối tượng. Việc phân tích và tính toán các vị trí vật rắn phải được thực hiện trước khi mô phỏng. Quá trình mô phỏng thực hiện vẽ lại các vị trí đã được tính toán. Vị trí đối tượng trong OpenGL cũng được thay đổi bằng phép nhân các tọa độ của đối tượng với ma trận hiện hành. Giá trị của các ma trận hiện hành được biến đổi bằng các phép biến đổi: tịnh tiến, quay.

## 5.4 Chức năng mô phỏng của phần mềm

Để mô tả một cách trực quan chuyển động của robot cũng như là học cách sử dụng robot trước khi vận hành thực tế, chức năng mô phỏng được đưa vào trong phần mềm. Luận văn sử dụng thư viện OpenGL để mô phỏng hóa các đối tượng trong phần mềm.



Hình 5.19: Quá trình mô hình hóa các chi tiết robot trong OpenGL

#### 5.4.1 Khởi tạo môi trường OpenGL trong phần mềm

Để có thể sử dụng thư viện OpenGL trong MFC, cần phải thiết lập môi trường để OpenGL có thể hoạt động được. Xây dựng một class có tên là COpenGLControl để quản lí các hoạt động liên quan đến hoạt động của OpenGL.

Dầu tiên, ta cần phải tạo định dạng điểm ảnh. Định dạng điểm ảnh sẽ miêu tả cách lưu trữ trong bộ nhớ các phần tử đồ họa mà Windows hiển thị. Thông số được điều khiển bởi định dạng điểm ảnh bao gồm độ sâu màu, phương pháp bộ đệm và giao diện đồ họa được hỗ trợ. Sau đó, tạo ra ngữ cảnh và làm cho nó hiện hành.

- Cài đặt định dạng điểm ảnh cho OpenGL:

```

static PIXELFORMATDESCRIPTOR      pfd =
{
    sizeof(PIXELFORMATDESCRIPTOR),
    1,
    PFD_DRAW_TO_WINDOW | PFD_SUPPORT_OPENGL |
    PFD_DOUBLEBUFFER,
    PFD_TYPE_RGBA,
    32,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    16,
    0, 0, 0, 0, 0, 0,
};

```

```

hdc = GetDC() ->m_hDC;
m_nPixelFormat = ChoosePixelFormat(hdc, &pf);
SetPixelFormat(hdc, m_nPixelFormat, &pf);

```

- Cách tạo ngữ cảnh và làm cho nó hiện hành:

```

hrc = wglCreateContext(hdc);
wglMakeCurrent(hdc, hrc);

```

Sử dụng các thông báo về sự kiện chuột WM\_MOUSEMOVE để:

- Điều chỉnh góc nhìn thông qua các thao tác kéo thả chuột trái.
- Điều chỉnh độ phóng vật thể bằng thao tác nhấn giữ chuột phải.
- **di** chuyển vị trí vật thể bằng nút giữa chuột.

Đoạn code được sử dụng trong chương trình ngắn của sự kiện WM\_MOUSEMOVE:

```

void COpenGLControl::OnMouseMove(UINT nFlags, CPoint point)
{
    int diffX = (int)(point.x - m_dLastX);
    int diffY = (int)(point.y - m_dLastY);

    m_dLastX = (float)point.x;
    m_dLastY = (float)point.y;

    if (nFlags & MK_LBUTTON)
    {
        m_dRotX += (float)0.5 * diffY;
        if (m_dRotX > 360.0 || m_dRotX < -360.0)
            m_dRotX = 0.0;

        m_dRotY += (float)0.5 * diffX;
        if (m_dRotY > 360.0 || m_dRotY < -360.0)
            m_dRotY = 0.0;
    }
    else if (nFlags & MK_RBUTTON)
        m_dZoom -= (float)2.0 * diffY;
    else if (nFlags & MK_MBUTTON)
    {
        m_dPosX += (float)1.5 * diffX;
        m_dPosY -= (float)1.5 * diffY;
    }

    OnDraw(NULL);
    CWnd::OnMouseMove(nFlags, point);
}

```

với OnDraw là hàm có tác dụng điều chỉnh camera, được lập trình như sau:

```
void COpenGLControl::OnDraw(CDC *pDC)
{
    glLoadIdentity();
    glTranslatef(0.0, 0.0, -m_dZoom);
    glTranslatef(m_dPosX, m_dPosY, 0.0);
    glRotatef(m_dRotX, 1.0, 0.0, 0.0);
    glRotatef(m_dRotY, 0.0, 1.0, 0.0);
}
```

Sử dụng thông báo WM\_ONSIZE để thực hiện các thiết lập Viewport và Matrix Mode.

Đoạn code trong chương trình ngắt thông báo WM\_ONSIZE:

```
void COpenGLControl::OnSize(UINT nType, int cx, int cy)
{
    CWnd::OnSize(nType, cx, cy);
    if(cx <= 0 || cy <= 0 || nType == SIZE_MINIMIZED) return;

    glViewport(0, 0, cx, cy);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

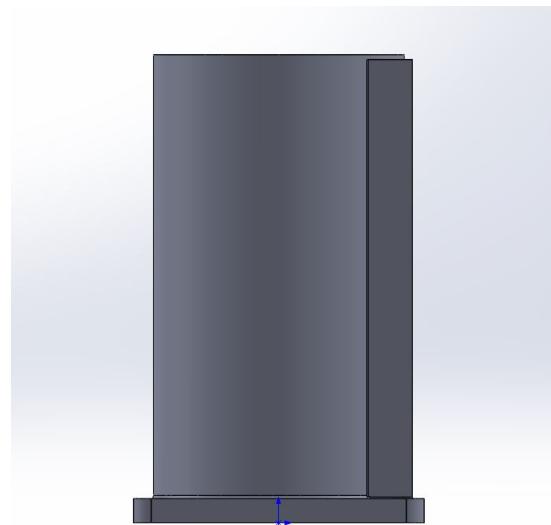
    gluPerspective(45.0, (float)cx/(float)cy, 100.0, 100000.0);
    glMatrixMode(GL_MODELVIEW);
}
```

Để sử dụng OpenGL trong môi trường dialog MFC, cần tạo một biến class COpenGLControl để thiết lập các cài đặt cũng như tạo ra môi trường để sử dụng OpenGL.

### 5.4.2 Thiết kế các chi tiết của robot bằng phần mềm Solid-Work

Trong khi cài đặt hệ tọa độ cho OpenGL thì hệ tọa độ được chọn là hệ tọa độ với trục y thẳng đứng. Do đó, khi thiết kế các chi tiết của robot, ta cũng thiết kế các chi tiết với hệ tọa độ tương tự để khi xuất ra tập tin định dạng \*.STL với tọa độ các điểm tương thích với tọa độ trong OpenGL.

- Khâu 1:



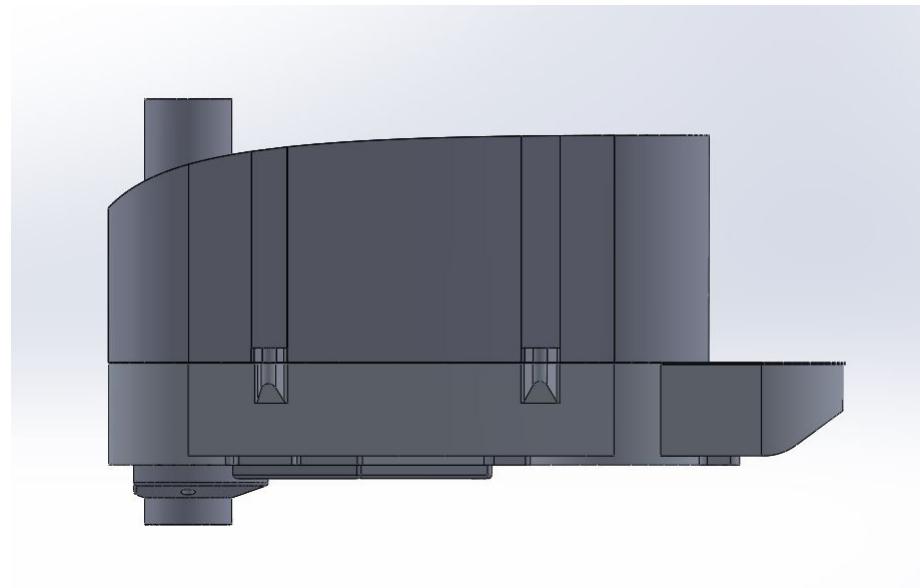
Hình 5.20: Khâu 1 robot SCARA

- Khâu 2:



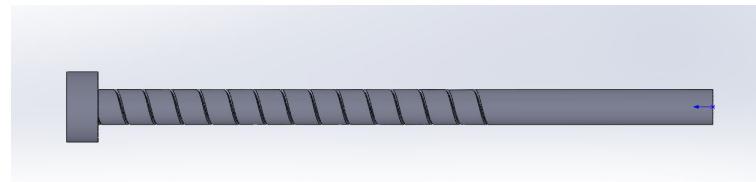
Hình 5.21: Khâu 2 robot SCARA

- Khâu 3:



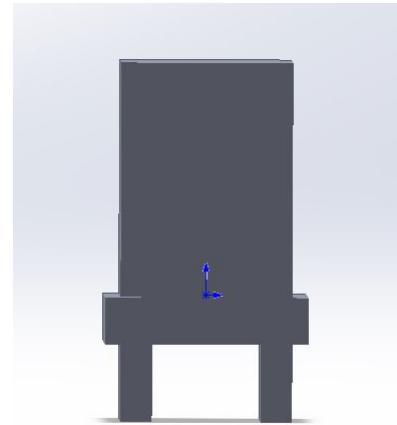
Hình 5.22: Khâu 3 robot SCARA

- Khâu 4:



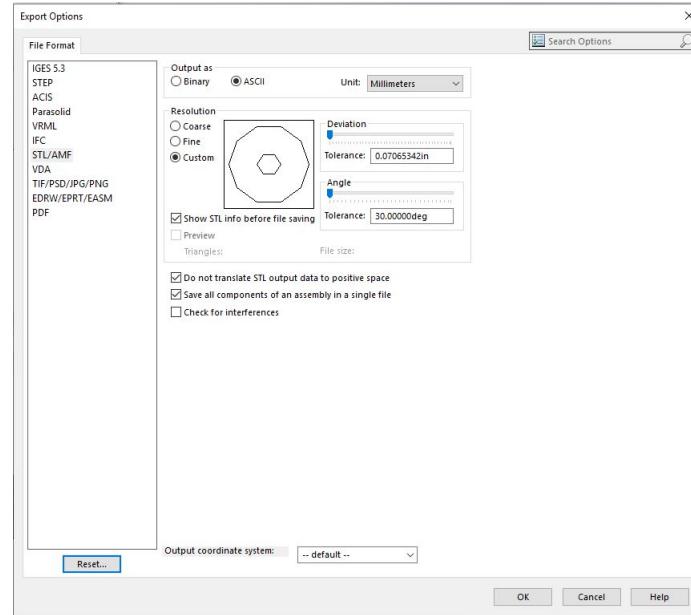
Hình 5.23: Khâu 4 robot SCARA

- Tay kẹp:



Hình 5.24: Tay kẹp robot SCARA

Sau khi thiết kế xong thì các chi tiết được lưu dưới dạng tập tin \*.STL. Tùy chỉnh để file STL sau khi chuyển có dạng mã ASCII. File STL sau khi chuyển sang phải được cài đặt như trong hình để hệ tọa độ khi vẽ trong SolidWork và trong tập tin định dạng STL không bị dịch chuyển.



Hình 5.25: Thiết lập tùy chọn cho file STL

Như vậy, sau khi thiết kế và thực hiện lưu lại theo định dạng STL **thì ta đã có file STL để làm vật liệu cho quá trình xây dựng mô hình robot trong OpenGL.**

### 5.4.3 Xây dựng module đọc tập tin định dạng STL

- Cấu trúc của file định dạng STL

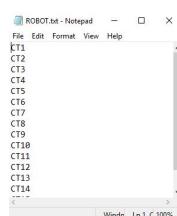
```
solid CT1
facet normal 0.00000e+000 0.00000e+000 1.00000e+000
    outer loop
        vertex 9.925000e+001 2.587010e+002 2.000000e+000
        vertex 5.299038e+000 2.607500e+002 2.000000e+000
        vertex 5.500000e+000 2.600000e+002 2.000000e+000
    endloop
endfacet
facet normal 0.00000e+000 0.00000e+000 1.00000e+000
    outer loop
        vertex 9.870097e+001 3.250000e+000 2.000000e+000
        vertex 9.850000e+001 4.000000e+000 2.000000e+000
        vertex 5.500000e+000 4.000000e+000 2.000000e+000
    endloop
endfacet
facet normal 0.00000e+000 0.00000e+000 1.00000e+000
    outer loop
        vertex 9.870097e+001 3.250000e+000 2.000000e+000
        vertex 5.500000e+000 4.000000e+000 2.000000e+000
        vertex 9.925000e+001 2.700962e+000 2.000000e+000
    endloop
endfacet
facet normal 0.00000e+000 0.00000e+000 1.00000e+000
    outer loop
        vertex 9.925000e+001 5.299038e+000 2.000000e+000
        vertex 1.000000e+002 5.500000e+000 2.000000e+000
        vertex 1.007500e+002 2.587010e+002 2.000000e+000
    endloop
endfacet
```

Hình 5.26: Cấu trúc file định dạng STL

Nhận xét: File định dạng STL có cấu trúc cứ sau 7 dòng thì cấu trúc lặp lại như ban đầu. Dòng đầu tiên của file là tên chi tiết được tên trong **solidworks**. Sau đó đến các vector pháp tuyến bề mặt cần vẽ và tọa độ 3 đỉnh của tam giác cơ sở. Như vậy sau khi lưu chi **tiết** trong Solidworks ở định dạng \*.STL thì chi tiết đã được số hóa thành nhiều bề mặt tam giác nhỏ hơn và các đỉnh tương ứng của nó. Đây chính là cơ sở để xây dựng module đọc file định dạng \*.STL để mô hình hóa các đối tượng bằng OpenGL.

- Xây dựng module đọc file \*.STL

Giải thích thuật toán: Ban đầu, chương trình sẽ đọc file "ROBOT.txt" là file text chứa tên các chi tiết đã được chuyển thành file định dạng \*.STL.



Hình 5.27: File ROBOT.txt chứa tên file STL

Chương trình kiểm tra sự tồn tại của file. Nếu file không tồn tại thì sẽ thông báo và kết thúc quá trình đọc file \*.STL. Ngược lại, nếu mở file thành công, chương trình lần lượt đọc từng dòng trong file "ROBOT.txt" và lưu vào mảng strPartFile với định dạng "CTx.STL". Kết thúc quá trình đọc file thì biến số chi tiết sẽ được gán giá trị cho tổng số file cần nạp.

Sau khi có được số chi tiết và mảng chứa tên tập tin cần đọc, khởi tạo một danh sách list có số phần tử bằng số chi tiết bằng lệnh:

```
list = glGenLists( nNumberOfParts );
```

Mục đích là tạo ra một danh sách list để thực hiện các thao tác vẽ và lưu lại trong mảng danh sách. Trong quá trình, chỉ cần gọi ra mà không cần vẽ lại giúp tối ưu cho quá trình mô phỏng, sau đó chương trình sẽ thực hiện mở từng file số liệu đã được lưu trong mảng strPartFile. Nếu quá trình mở thành công, chương trình sẽ tiếp tục nạp và vẽ lại chi tiết bằng câu lệnh:

```
glNewList( list + i , GL_COMPILE );
```

Mục đích câu lệnh nhằm tạo ra 1 list mới để lưu quá trình vẽ của chi tiết đang cần nạp. Tiếp theo là đọc tọa độ đỉnh tam giác theo cấu trúc đã phân tích ở trên rồi vẽ mặt tam giác cơ sở.



Hình 5.28: Sơ đồ giải thuật đọc file STL

Nội dung code module đọc file STL:

```
void COpenGLControl::ReadFileSTL()
{
    FILE *file;
    int nLoop, nNumberOfPart, list;
    char cPartName[20];
    char strPartFile[100][216];
    char strRead[200];
    char catbo[20];
    float fd1, fd2, fd3;
    float fdinh[1000][3];
    float fi1, fi2, fi3;
    file = fopen("ROBOT.txt", "r");
    if (file == NULL)
    {
        MessageBox(_T("Cannot find the file !"));
        ExitThread(0);
    }
    else
    {
        nLoop = 0;
        while (!feof(file))
        {
            fscanf(file, "%s", cPartName);
            sprintf(strPartFile[nLoop], "%s.STL", cPartName);
            nLoop++;
        }
        fclose(file);
        nNumberOfPart = nLoop;
    }

    list = glGenLists(nNumberOfPart);

    for (int i = 0; i < nNumberOfPart; i++)
    {
        file = fopen(strPartFile[i], "r");
        glNewList(list + i, GL_COMPILE);
        glBegin(GL_TRIANGLES);
        fgets(strRead, 200, file);
        while (!feof(file))
        {
            fgets(strRead, 200, file);
            sscanf(strRead, "%16c%f%f", &catbo, &fi1, &fi2, &fi3);
            if (!feof(file))
            {

```

```

fgets(strRead, 200, file);
for(int j = 1; j < 4; j++)
{
    fgets(strRead, 200, file);
    sscanf(strRead, "%16c%f%f%f", &catbo, &fd1, &fd2, &fd3);
    fdinh[j][0] = fd1;
    fdinh[j][1] = fd2;
    fdinh[j][2] = fd3;
}
glNormal3f(fi1, fi2, fi3);
Paint(fdinh[1], fdinh[2], fdinh[3]);
fgets(strRead, 200, file);
fgets(strRead, 200, file);
}
}
fclose(file);
glEnd();
glEndList();
}
}

```

#### 5.4.4 Mô hình hóa robot SCARA bằng OpenGL

Sau khi thực hiện quá trình đọc các tập tin định dạng STL trên, các chi tiết được lưu trong List. Do đó để vẽ bất kì chi tiết nào, chỉ cần gọi lệnh glCallList(i) với i là số thứ tự của chi tiết.

Để lắp những chi tiết riêng biệt thành mô hình robot hoàn chỉnh, ta có thể sử dụng các phép tịnh tiến và phép quay để di chuyển vị trí của các chi tiết robot. 2 lệnh thường dùng để di chuyển các chi tiết:

- glTranslatef(GLfloat x, GLfloat y, GLfloat z): dịch chuyển hệ tọa độ tịnh tiến một đoạn (x,y,z).
- glRotatef(GLfloat angle, GLfloat x, GLfloat y, GLfloat z): quay hệ trục tọa độ theo vector (x,y,z) một góc angle.

Các thông số về vector tịnh tiến cũng như là góc quay tương tự các hệ số trong bảng D-H nhưng hệ tọa độ có sự thay đổi: chọn y là trục **của các động cơ**.

Doan code trong chương trình để vẽ robot SCARA:

```

void CSimulationDlg::DrawRobot(double theta1, double theta2, double theta3)
{
    glPushMatrix();
    glPushMatrix();
    glRotatef(90.0, 0.0, 1.0, 0.0);

    glPushMatrix();

```

```

glTranslatef(-52.0, 20.0, -82.0);
glCallList(1);
glPopMatrix();

glPushMatrix();
glRotatef(90.0, 0.0, 1.0, 0.0);
glCallList(2);
glPopMatrix();

glPopMatrix();

glTranslatef(0.0, 290.0, 0.0);
glCallList(3);

glTranslatef(0.0, 74.0, 0.0);
glCallList(4);

glPushMatrix();
glTranslatef(0.0, -29.5, 0.0);

glRotatef(theta1 - 90.0, 0.0, 1.0, 0.0);
glCallList(5);

glTranslatef(420.0, 29.5, 0.0);
glCallList(6);

glTranslatef(0.0, 20.0, 0.0);

glPushMatrix();
glRotatef(180, 1.0, 0.0, 0.0);
glRotatef(-theta2, 0.0, 1.0, 0.0);
glCallList(7);

glPushMatrix();
glRotatef(-90.0, 0.0, 1.0, 0.0);
glTranslatef(0.0, 0.0, -235.0);
glCallList(8);
glPopMatrix();

glRotatef(180.0, 1.0, 0.0, 0.0);
glTranslatef(0.0, 45.0, 0.0);
glCallList(9);

glPushMatrix();
glRotatef(180.0, 0.0, 0.0, 1.0);

```

```

glTranslatef(80.0, 0.0, 0.0);
glCallList(10);
glPopMatrix();

glTranslatef(235.0, -78 + theta3, 0.0);
glRotatef(theta4, 0.0, 1.0, 0.0);
glCallList(11);

glTranslatef(0.0, -75.0, 0.0);
glCallList(12);

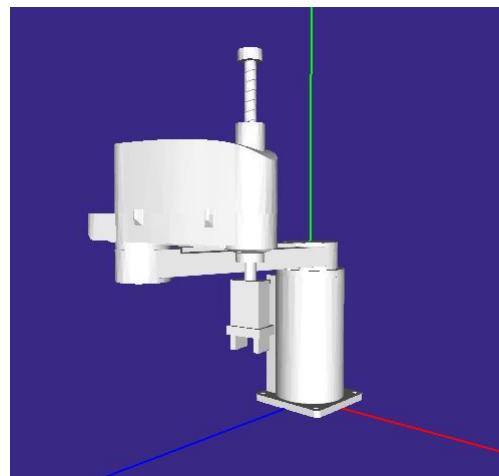
glPushMatrix();

glPopMatrix();
glPopMatrix();
glPopMatrix();
glPopMatrix();
}

}

```

Kết quả xây dựng mô hình robot SCARA bằng OpenGL:



Hình 5.29: Mô hình robot SCARA xây dựng bằng OpenGL

#### 5.4.5 Tạo chuyển động cho robot

Có 2 cách để tạo chuyển động cho mô hình robot: tạo vòng lặp để cho robot đi qua các điểm đã quy định hoặc dùng timer vẽ lại robot sau mỗi chu kỳ ngắn với giá trị góc khớp tương ứng. Trong lập trình ứng dụng bằng MFC, môi trường hoạt động là Windows, với cơ chế xử lý và hoạt động theo các thông báo (Message). Do vậy, việc sử dụng vòng lặp có thể làm treo chương trình hoặc phát sinh lỗi. Cho nên, cách sử dụng Timer để tạo chuyển động robot được sử dụng trong luận văn.

Để khởi tạo Timer, dùng lệnh:

```

UINT_PTR SetTimer(UINT_PTR nIDEvent, UINT nElapse,
void (CALLBACK* lpfnTimer)(HWND, UINT, UINT_PTR, DWORD))

```

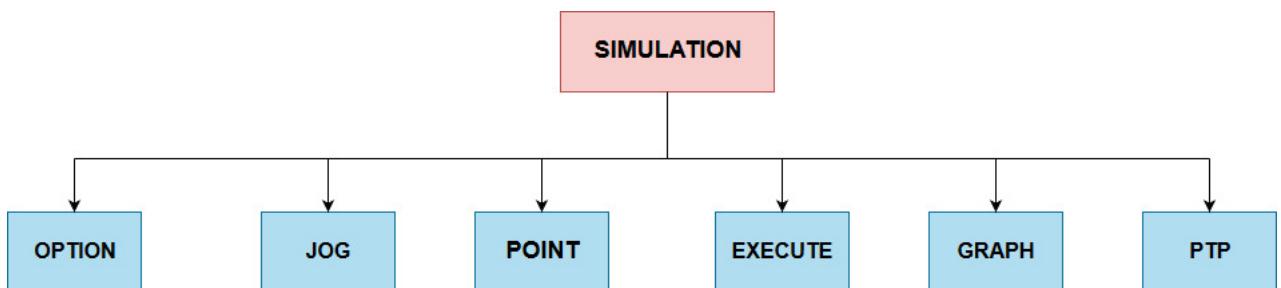
Ngược lại, để ngắt Timer, dùng lệnh:

```
KillTimer(UINT_PTR nIDEvent)
```

Mỗi Timer khi thực thi sẽ sinh ra một hàm ngắt OnTimer bằng cách gửi đi thông báo WM\_TIMER. Để tạo chuyển động của robot, trong hàm ngắt timer, thực hiện xóa các hình ảnh đã vẽ, sau đó vẽ lại hình ảnh robot ở vị trí mới, thực hiện lệnh SwapBuffer để hiển thị hình ảnh mới lên màn hình.

## 5.5 Giao diện mô phỏng

### 5.5.1 Sơ đồ khái niệm phần mô phỏng



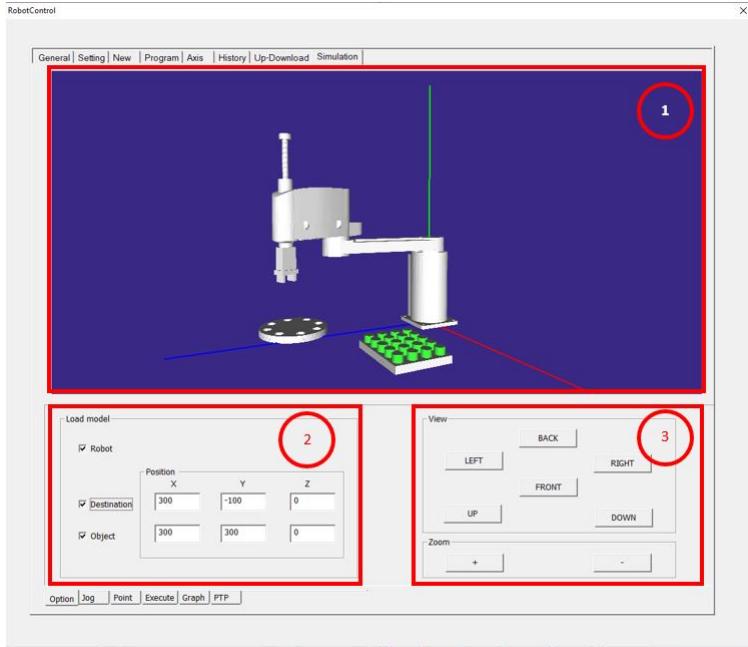
Cấu trúc mô phỏng bao gồm 6 phần:

- Option: Lựa chọn model cần hiển thị, tùy chọn về camera.
- Jog: Robot hoạt động ở chế độ Jog Mode và có thể dạy điểm cho robot.
- Point: Lưu lại những điểm đã được dạy.
- Execute: Thực thi tác vụ.
- Graph: Đồ thị trong quá trình mô phỏng.
- PTP: Robot chuyển động giữa 2 điểm theo đường thẳng hoặc đường tròn.

### 5.5.2 Giao diện Option

Chức năng: Dùng để hiển thị robot và các vật thể lên màn hình hiển thị, đồng thời người dùng có thể tùy chỉnh các tùy chọn về camera như góc nhìn và độ phóng vật thể.

- Vùng 1: vùng hiển thị robot và các vật thể trong không gian làm việc.
- Vùng 2: các tùy chọn về tải các model như robot, vật thể.
- Vùng 3: tùy chọn về camera.



Hình 5.31: Giao diện Option

### 5.5.3 Giao diện Jog

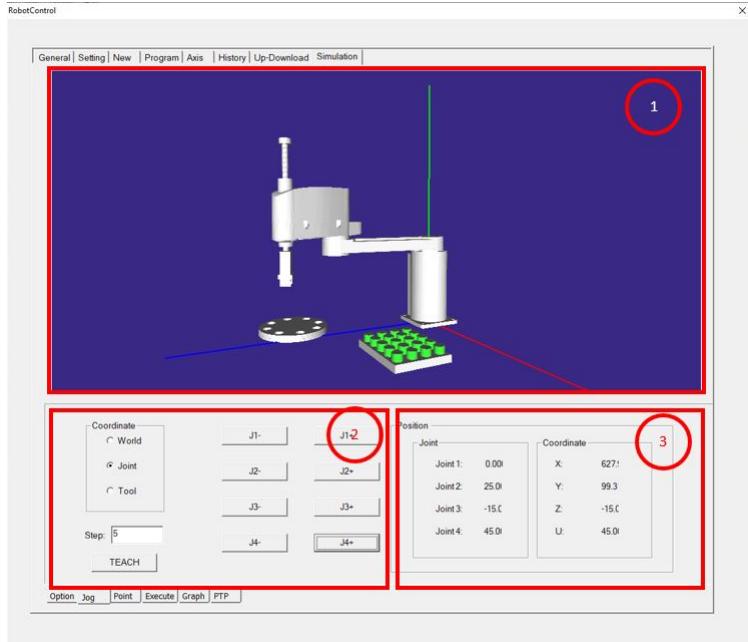
Chức năng: robot hoạt động ở chế độ Jog mode. Người dùng có thể tiến hành cho robot hoạt động theo chế độ từng trục của robot, từng trục của hệ tọa độ gốc và theo hệ tọa độ gắn trên đầu công tác. Vị trí của robot trong quá trình Jog sẽ được hiển thị ở 2 dạng là vị trí trong không gian biến khớp và vị trí trong không gian làm việc. Người dùng có thể dạy điểm này cho robot bằng cách nhấn nút TEACH để lưu vào danh sách các điểm robot được dạy. Đồng thời, điểm được dạy sẽ hiển thị lên màn hình.

- Vùng 1: vùng hiển thị robot và các vật thể trong không gian làm việc.
- Vùng 2: lựa chọn chế độ jog và nhập bước trong quá trình thực hiện, các nút nhấn tăng giảm cho từng trục
- Vùng 3: vị trí hiện tại của robot.

Trường hợp robot hoạt động ở Jog theo chế độ Joint: Khi có sự kiện nhấn nút xảy ra thì giá trị góc khớp tương ứng sẽ thay đổi tùy thuộc vào chiều của nút nhấn. Từ đó có thể tính được góc khớp tại điểm đích và đưa giá trị góc khớp vào timer của lớp CSimulationDlg để robot có thể đi tới điểm đích trên màn hình hiển thị. Trong quá trình thực hiện, **Jog mode**, các góc khớp được giới hạn trong tầm làm việc để robot không đi ra khỏi vùng làm việc. Để lấy con trỏ của class mẹ từ class con CJogDlg, ta sử dụng phương thức HWND GetParent() như sau:

```
CSimulationDlg*p = (CSimulationDlg*)GetParent();
```

Sau đó dùng con trỏ p để truy cập đến các giá trị biến khớp của robot khi mô phỏng.



Hình 5.32: Giao diện Jog

Trường hợp robot hoạt động ở chế độ World: Khi có sự kiện nhấn nút xảy ra, kiểm tra điều kiện về vị trí có nằm ngoài không gian làm việc hay ở vị trí biên giới không gian làm việc. Tùy vào chiều của nút nhấn mà tăng hoặc giảm giá trị tọa độ của trực tương ứng. Sau đó giải bài toán động học ngược, tìm ra giá trị góc khớp điểm đích và thực hiện di chuyển robot như ở trường hợp jog theo hệ tọa độ Joint.

Trường hợp robot hoặc động ở chế độ Tool: Gọi H là ma trận chuyển đổi tương ứng với nút nhấn, T là ma trận chuyển đổi từ hệ tọa độ gốc đến end-effector ban đầu của robot. Ma trận chuyển đổi  ${}^0T_4$  của robot khi thực hiện chuyển đổi theo ma trận H được tính theo công thức:  ${}^0T_4 = H*T$ .

Giải bài toán động học ngược, tìm được điểm cuối và thực hiện như trên để truyền thông số biến khớp vào timer của class CSimulationDlg.

Nút nhấn TEACH dùng để lưu tọa độ điểm hiện tại vào bảng, lưu lại vị trí cần di qua của robot. Để ghi giá trị tọa độ từ mục Jog sang mục Point, trong chương trình ngắt nút nhấn TEACH, thực hiện như sau:

```

void CJogDlg::OnBnClickedTeachingButton()
{
CSimulationDlg* p = (CSimulationDlg*)GetParent();

CString Str;
Str.Format("%d", p->m_PointArray.size()/4);

p->m_PointArray.resize(p->m_PointArray.size() + 4);

p->m_PointListDlg.m_PointListCtrl.InsertItem(p->m_nPointCount, Str);

GetDlgItem(IDC_X)->GetWindowTextA(Str);
p->m_PointListDlg.m_PointListCtrl.SetItemText(p->m_nPointCount, 2, Str)
p->m_PointArray[4*p->m_nPointCount] = atof(Str);

GetDlgItem(IDC_Y)->GetWindowTextA(Str);
p->m_PointListDlg.m_PointListCtrl.SetItemText(p->m_nPointCount, 3, Str)
p->m_PointArray[4*p->m_nPointCount + 1] = atof(Str);

GetDlgItem(IDC_Z)->GetWindowTextA(Str);
p->m_PointListDlg.m_PointListCtrl.SetItemText(p->m_nPointCount, 4, Str)
p->m_PointArray[4*p->m_nPointCount + 2] = atof(Str);

GetDlgItem(IDC_U)->GetWindowTextA(Str);
p->m_PointListDlg.m_PointListCtrl.SetItemText(p->m_nPointCount, 5, Str)
p->m_PointArray[4*p->m_nPointCount + 3] = atof(Str);

p->m_nPointCount++;
}

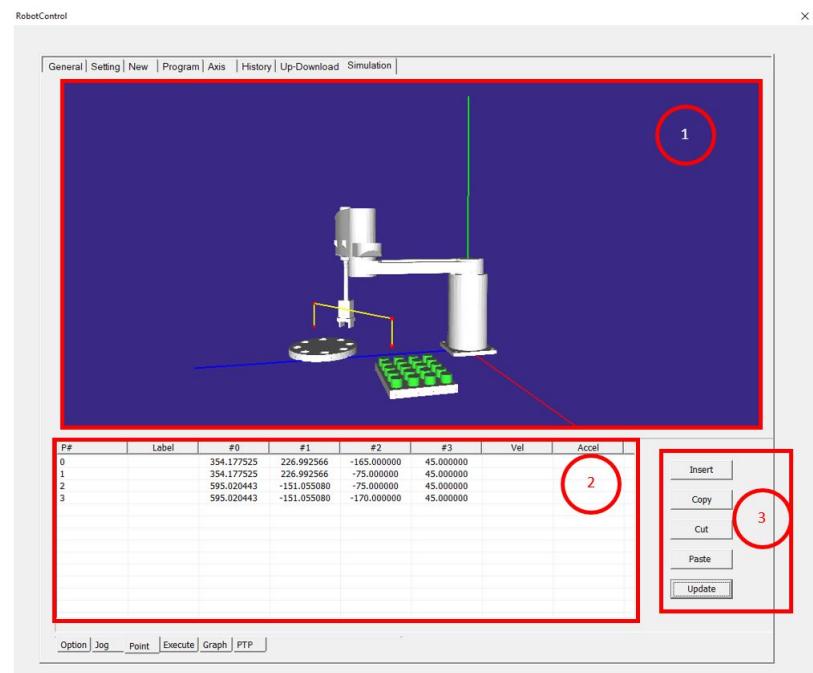
```

#### 5.5.4 Giao diện Point

Chức năng: giao diện có dạng bảng để lưu giá trị vị trí các điểm được dạy. Người dùng có thể sử dụng các điểm này để lập trình robot theo kiểu dạy học. Các nút nhấn INSERT, COPY, CUT, PASTE dùng để điều chỉnh lại bảng theo ý muốn. Ngoài ra, người dùng có thể sử dụng Menu hiện lên khi nhấn chuột phải vào bảng.

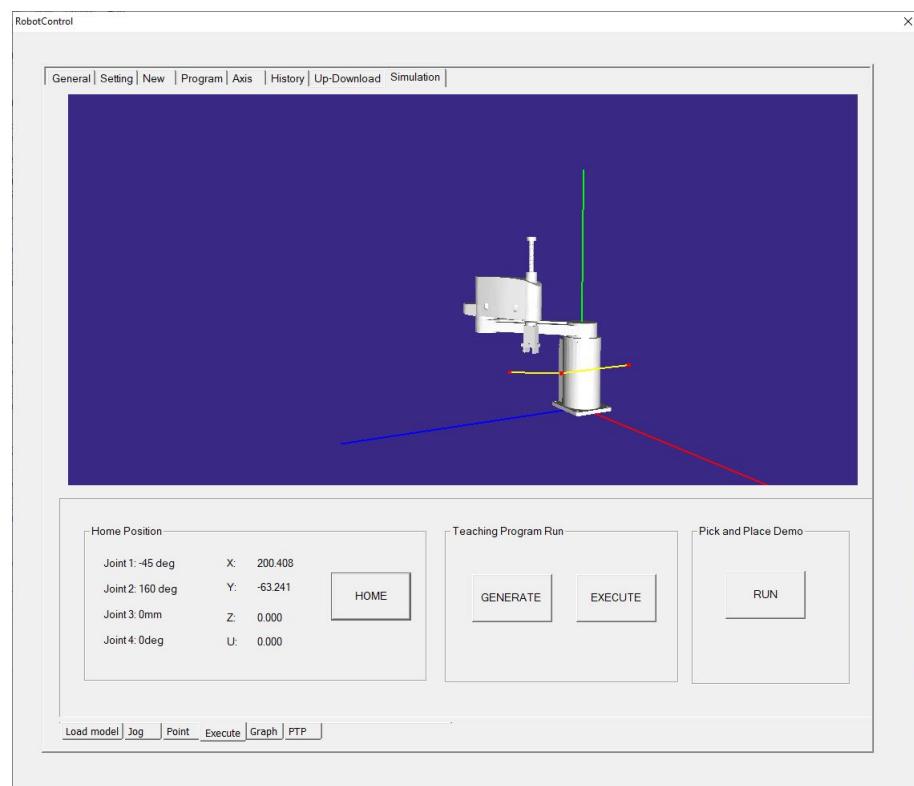
- Vùng 1: vùng hiển thị robot và các vật thể trong không gian làm việc.
- Vùng 2: bảng hiển thị điểm.
- Vùng 3: các nút nhấn để điều chỉnh sửa bảng.

Nút nhấn UPDATE dùng để cập nhật bảng vị trí các điểm sau đó vẽ lại dạng đường đi trên màn hình hiển thị.



Hình 5.33: Giao diện Point

### 5.5.5 Giao diện Execute

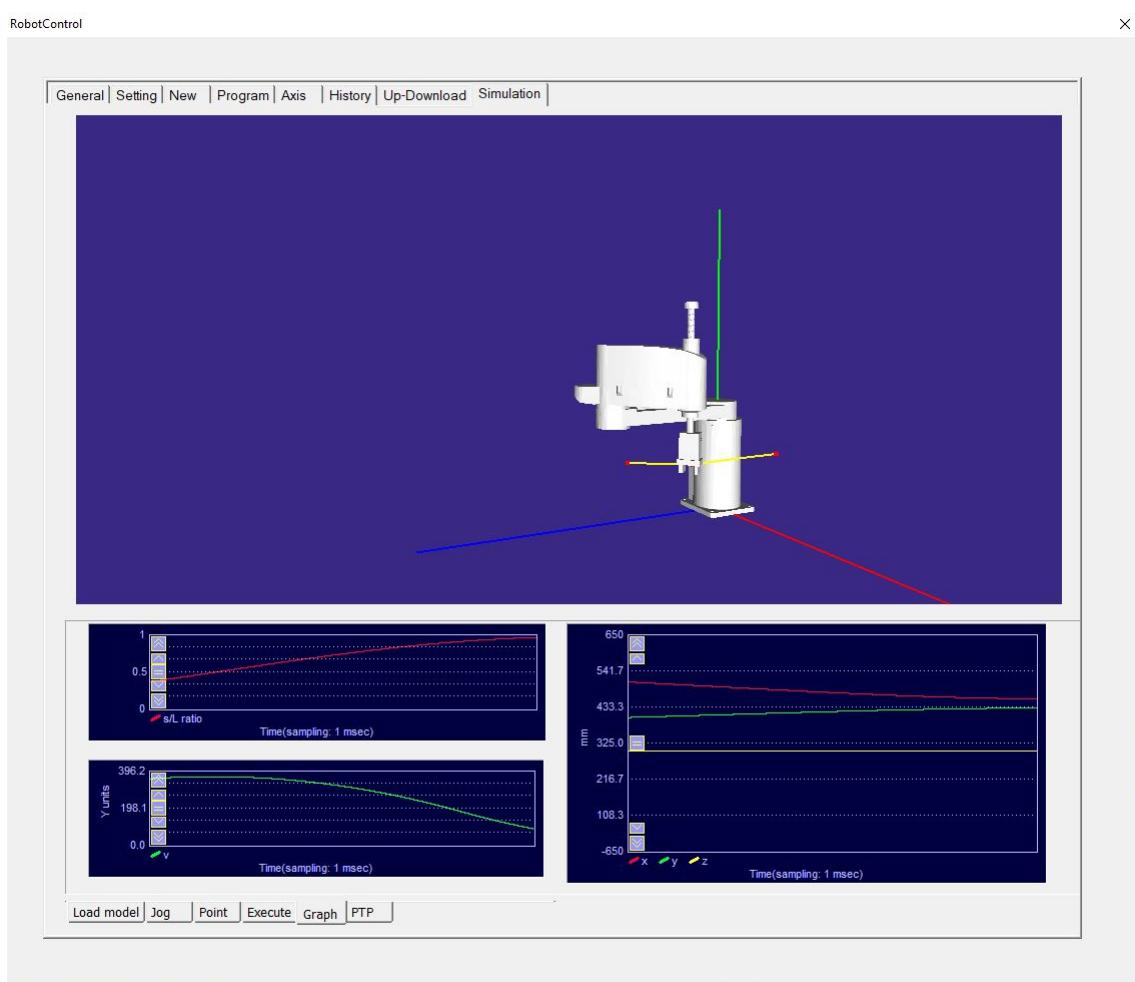


Hình 5.34: Giao diện thực thi chương trình

Chức năng: thực hiện đưa robot về vị trí Home được cài đặt. Để robot có thể thực hiện mô phỏng, nhấn nút GENERATE để tạo ra tập giá trị biến khớp để robot có thể mô phỏng được. Sau đó nhấn nút RUN để thực thi chương trình được dạy. Trong chương trình mô phỏng, phần mềm xây dựng một tác vụ được lập trình sẵn để mô phỏng công việc gấp thả vật.

### 5.5.6 Giao diện Graph

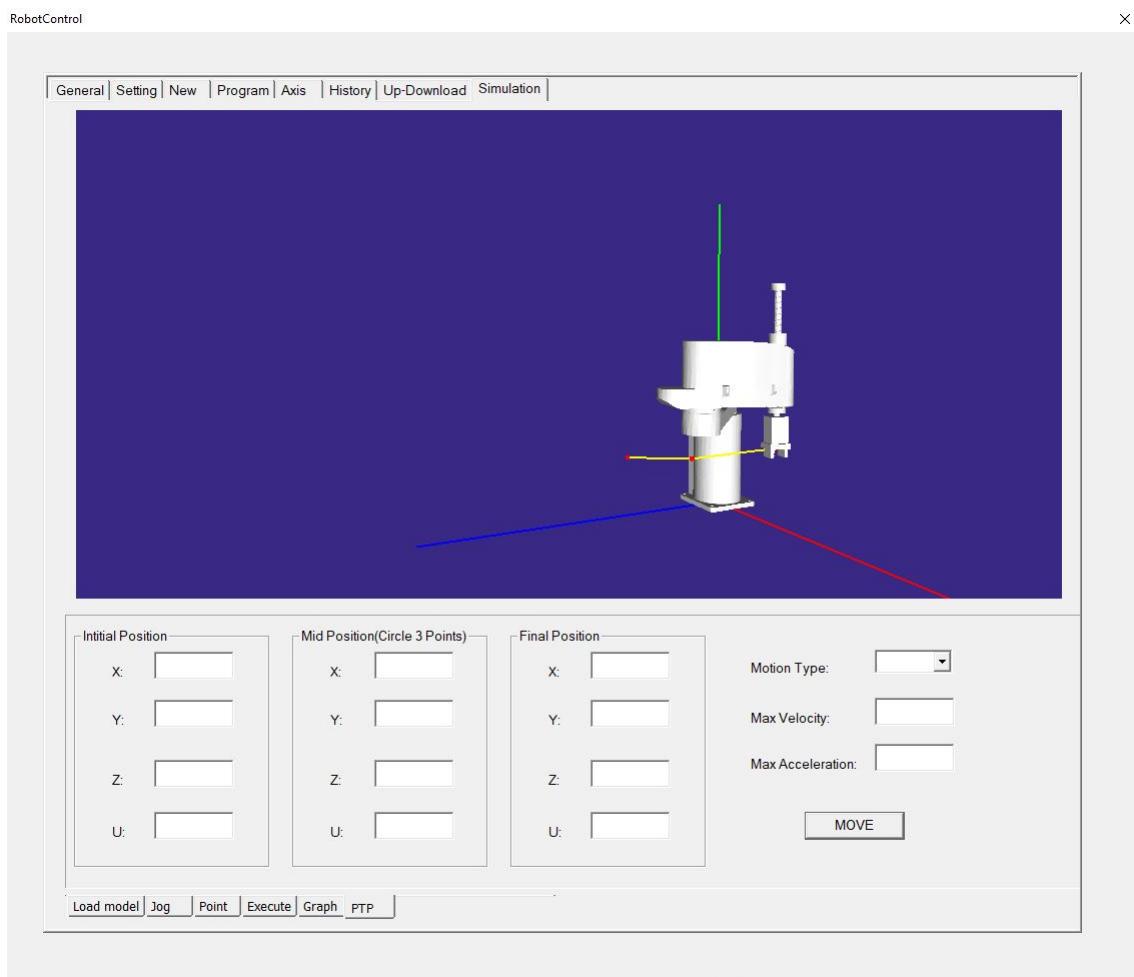
Chức năng: giúp người dùng quan sát được các profile vận tốc, quãng đường trong quá trình di chuyển, vị trí của điểm cuối trên 3 trục tọa độ.



Hình 5.35: Giao diện graph

### 5.5.7 Giao diện PTP

Chức năng: Giao diện PTP: để người dùng thực hiện các chuyển động Point to Point với quỹ đạo đường thẳng hoặc đường tròn đi qua 3 điểm cho trước.

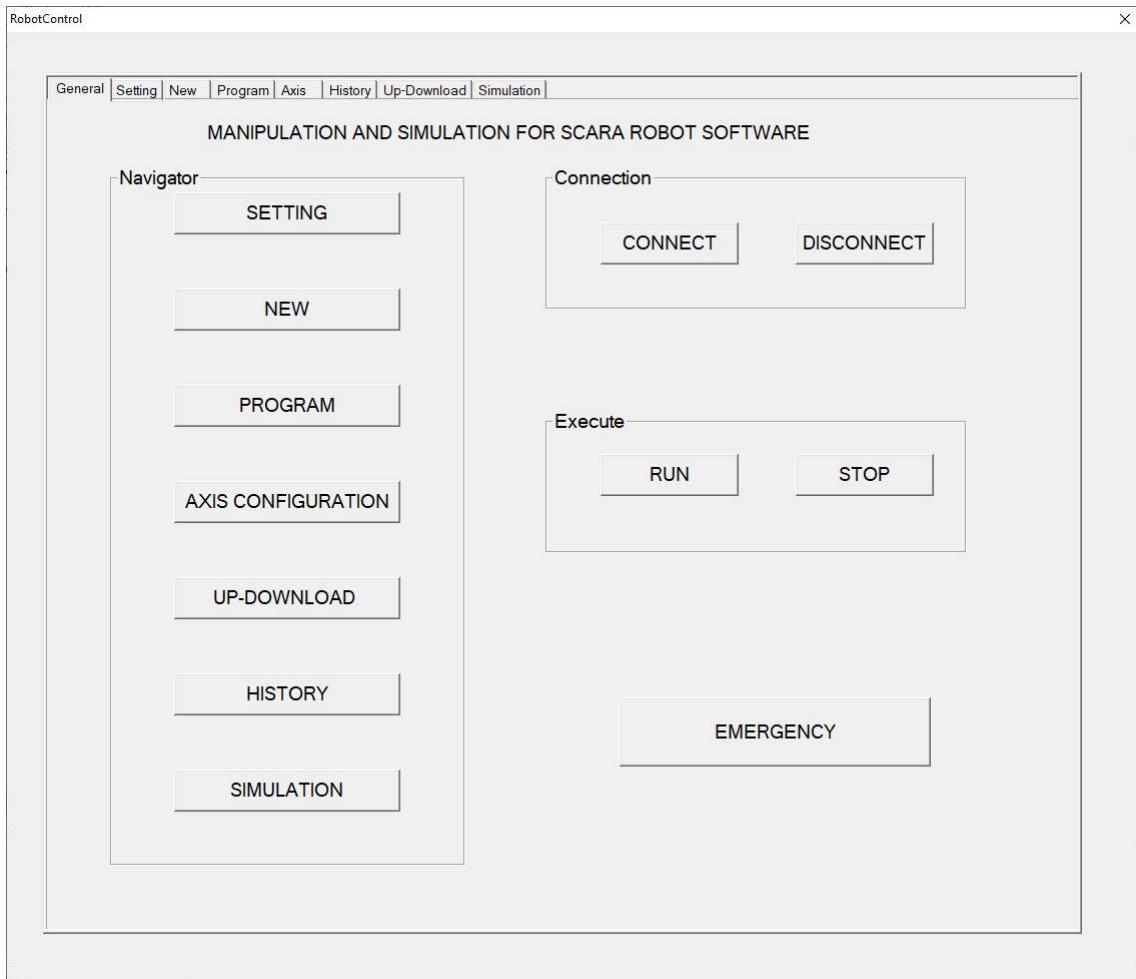


Hình 5.36: Giao diện PTP

## 5.6 Chức năng điều khiển của phần mềm

**Hình trên** là giao diện chính của chương trình khi khởi động, giao diện chính bao gồm các Tab chức năng:

- General: tạo đường dẫn nhanh để chuyển sang Tab chức năng khác cũng như thực hiện việc kết nối phần mềm với board điều khiển và cho phép hoạt động thực thi và dừng chương trình của robot.
- Setting: Thiết lập các thông số về truyền thông kết nối giữa board điều khiển, các thông số của bộ điều khiển.



Hình 5.37: Giao diện chính của chương trình

- **New:** Cho phép người dùng tạo các dự án mới cũng như các file trong một dự án.
- **Program:** Giao diện cho phép người dùng lập trình các dự án. Trong Tab này, người dùng có thể lập trình, chỉnh sửa các file JOB Script, file PNT là file chứa tọa độ các điểm được người dùng dạy cho robot. Dự án sau khi được lập trình xong được biên dịch thành mã ASM.
- **Axis:** Nhiệm vụ chính của Tab là tạo ra môi trường để người dùng có thể thiết lập một số thông số về Motion cơ bản cho từng trục của robot cũng như chạy thử robot ở chế độ Jog Mode để kiểm tra hoạt động của robot.
- **History:** Các hoạt động của người dùng từ lúc khởi động chương trình được lưu lại và hiển thị trong giao diện giúp người dùng nắm được các hoạt động cũng như kết quả thực hiện chương trình.
- **Up-Download:** Sau khi lập trình dự án cho robot xong, cần thực hiện tải mã code xuống Flash để board điều khiển **có thể tính toán và thực hiện**.

### 5.6.1 General Tab

Dây là Tab chức năng được khởi động đầu tiên khi chạy chương trình. Từ giao diện chính này, người dùng có thể chuyển sang các Tab chức năng khác bằng cách click chuột. Ngoài ra, để chuyển đổi sang Tab chức năng khác, người dùng có thể nhấn trực tiếp bằng nút nhấn trong mục Navigator trong giao diện chính.

Trong vùng Connection, có 2 nút nhấn là CONNECT và DISCONNECT cho phép người dùng thực hiện kết nối hoặc ngắt kết nối giữa máy tính và board điều khiển.

- Khi nhấn nút CONNECT, máy tính kiểm tra tên của cổng COM được cấu hình có giống với cổng COM của board khi kết nối với máy tính. Sau đó thực hiện việc cấu hình baudrate cho giao tiếp giữa **board và bộ điều khiển**. Máy tính tiếp tục thực hiện công việc đọc các trạng thái Mechanical, các mức tích cực của tín hiệu Mechanical, lấy giá trị cài đặt về Pulse Out Method và giá trị UnitperPulse của từng trục. Các công việc này để kiểm tra xem trạng thái của robot cũng như là kiểm tra kết nối của board và máy tính đã thành công hay chưa.
- Khi nhấn nút DISCONNECT, kết nối giữa board điều khiển và máy tính được ngắt.

Trong vùng Execute, 2 nút nhấn START và STOP có chức năng liên quan đến việc thực thi chương trình và dừng thực hiện chương trình.

- Khi nhấn nút RUN, máy tính gửi frame đến board điều khiển tương đương với lệnh thực thi chương trình với định dạng chuỗi là ("%cTRUN,0;%d%c", STX, CRC, ETX). Robot sẽ thực thi chương trình chứa trong Flash.
- Khi nhấn nút DISCONNECT, máy tính gửi frame đến board điều khiển tương đương với lệnh dừng thực hiện chương trình với định dạng chuỗi là ("%cTSTOP,0;%d%c", STX, CRC, ETX). Sau khi nhận được lệnh, robot sẽ dừng thực thi chương trình đang chạy.

Nút nhấn Emergency dùng để kết thúc khẩn cấp chương trình đang chạy. Khi nhấn nút, frame lệnh tương đương với lệnh dừng khẩn cấp được gửi xuống board từ máy tính với định dạng ("%cAXET;%d%c", STX, CRC, ETX). Khi nhận được lệnh, robot lập tức dừng tất cả các hoạt động đang diễn ra. Nút nhấn này được dùng khi robot vận hành không theo ý muốn của người lập trình hoặc là board điều khiển vận hành không chính xác.

### 5.6.2 Setting Tab

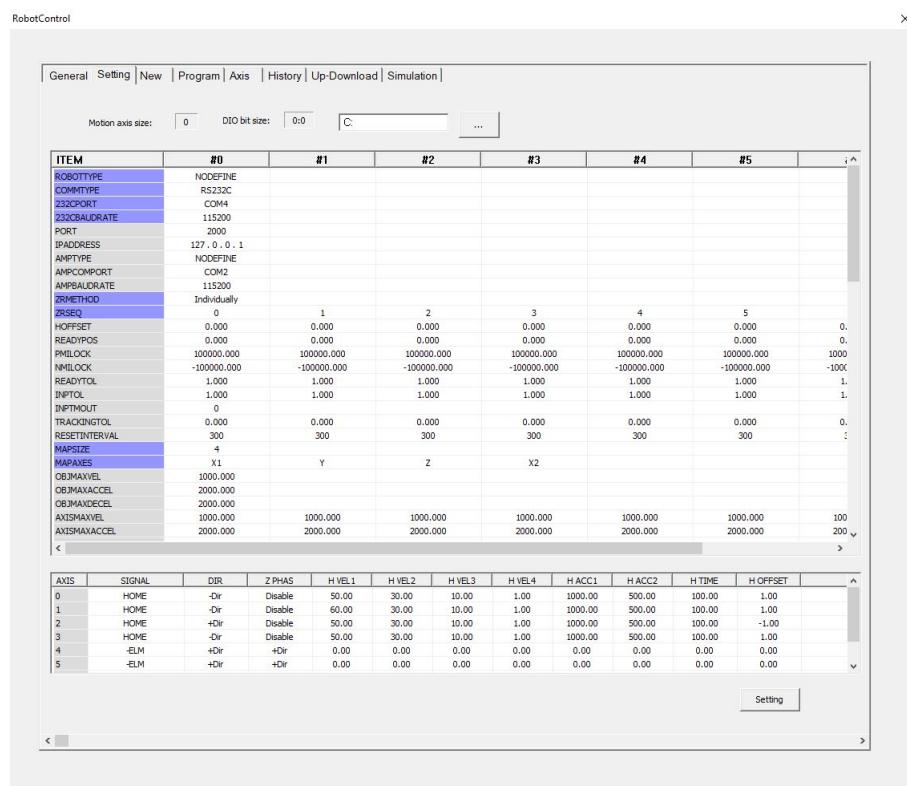
Để thiết lập kết nối cho board điều khiển và máy tính, trước tiên người sử dụng cần cài đặt các thông số liên quan truyền thông như: tên cổng COM, tốc độ baud.

Robot muốn hoạt động chính xác thì trước khi vận hành robot, người sử dụng cần cài đặt các thông số liên quan đến chuyển động của từng trục cũng như board điều khiển.

Các thông số chính mà người sử dụng cần lập trình liên quan đến các cài đặt về **chuyển động** của các trục như: UnitperPulse, Pulse Out Method, Encoder Method, các mức tín cực của tín hiệu Mechanical. Ngoài ra, người dùng có thể cài đặt về phương thức xác định vị trí Home.

Nút nhấn Setting dùng để lưu các giá trị đã thiết lập thành file .txt để sử dụng cho lần sau và đồng thời tạo ra file cài đặt Motion Parameter để người dùng tải xuống Flash cho board điều khiển.

Giao diện của Setting Tab:

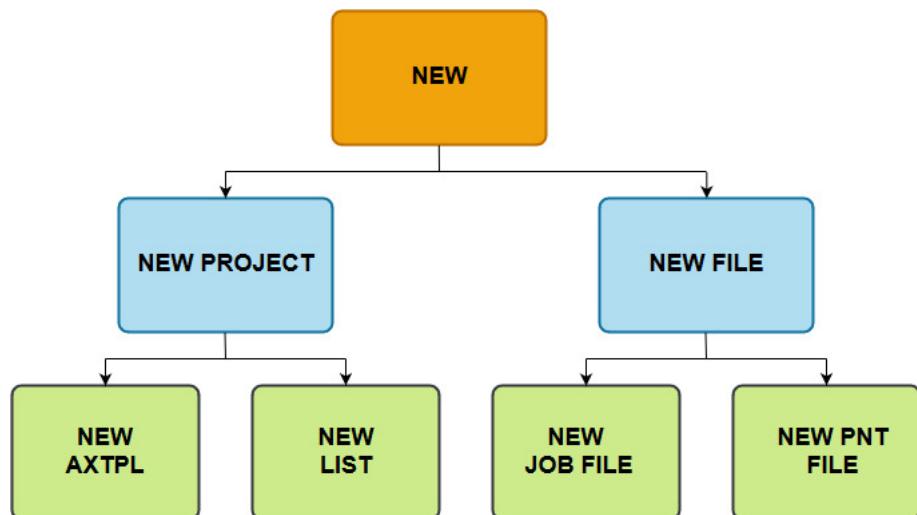


Hình 5.38: Giao diện cài đặt Setting

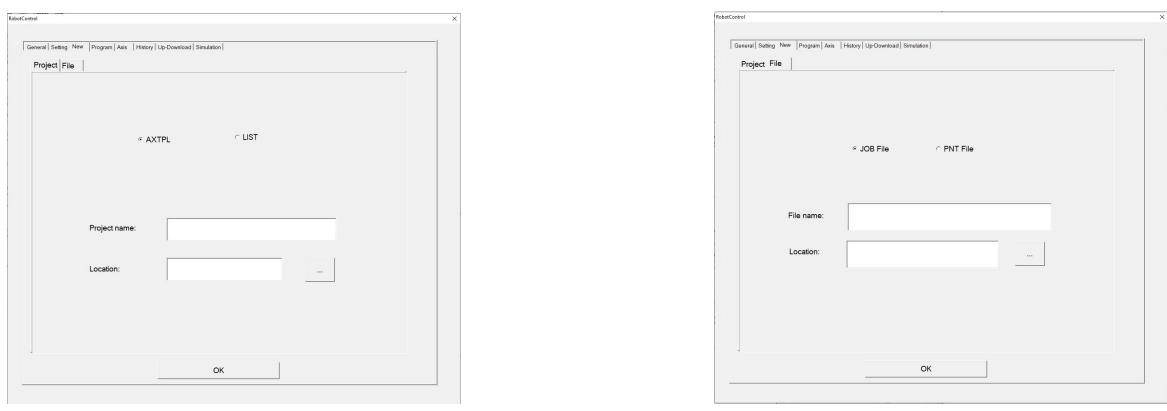
### 5.6.3 New Tab

New Tab có nhiệm vụ giúp người dùng tạo ra các dự án hoặc từng file riêng lẻ trong dự án. Có 2 loại dự án là lập trình sử dụng ngôn ngữ lập trình robot AXTPL và lập trình theo kiểu Wizard. Các file trong một dự án bao gồm: JOB file và PNT file.

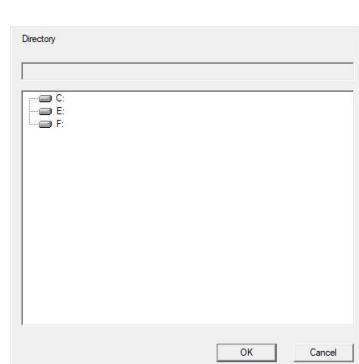
Trong mỗi mục nhỏ của New Tab đều có Edit Control để người dùng có thể nhập tên cho dự án hoặc tên file và đường dẫn để hiện vị trí cần lưu file vào. Đường dẫn có thể chọn bằng cách nhấn vào nút nhấn bên cạnh ô Browse. Khi nhấn nút Browse, một hộp thoại xuất hiện để người dùng chọn đường dẫn. Nút nhấn OK để chọn đường dẫn đã chọn, nút Cancel để quay lại giao diện trước đó.



Hình 5.39: Sơ đồ tổ chức của New Tab



Hình 5.40: Giao diện tạo project mới và các file trong một project

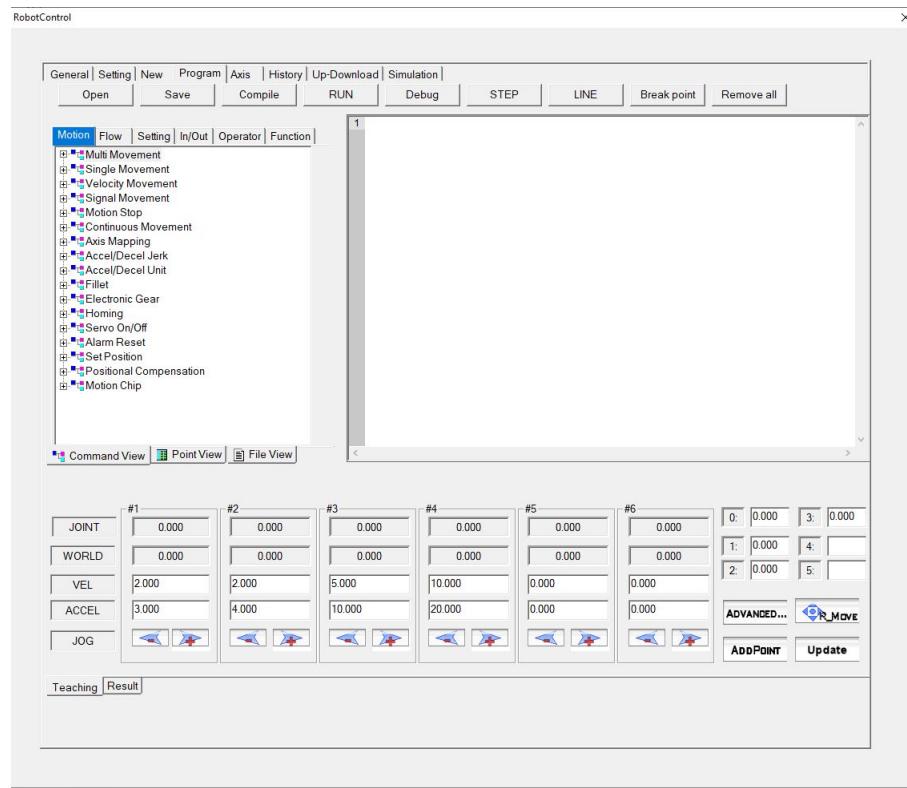


Hình 5.41: Hộp thoại chỉ dẫn vị trí lưu file

Nút nhấn OK dùng để xác nhận hoàn tất quá trình tạo file mới.

## 5.6.4 Program Tab

### 1. Giao diện lập trình robot



Hình 5.42: Giao diện Program Tab

Vùng soạn thảo chương trình robot nằm chính giữa giao diện. Giao diện chính bao gồm:

- Command View: Danh sách lệnh liên quan đến chuyển động, I/O, các loại biến, toán hạng và câu lệnh rẽ nhánh,..



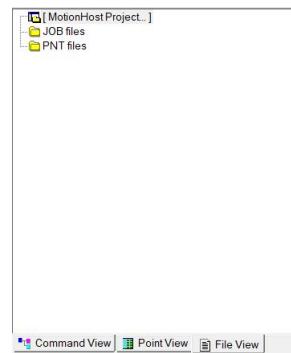
Hình 5.43: Giao diện Command View

- Point View: chứa các điểm đã được dạy trong quá trình lập trình robot.



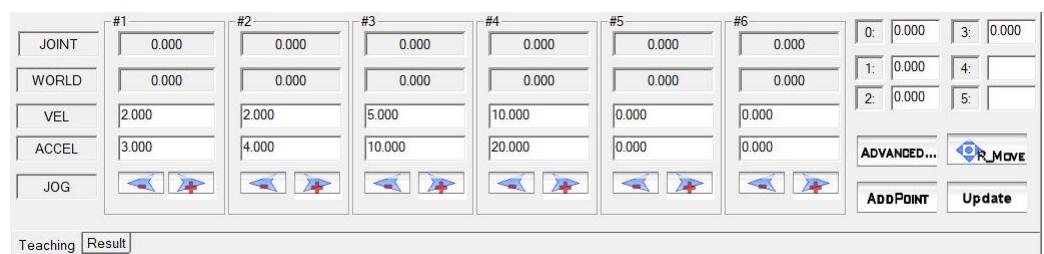
Hình 5.44: Giao diện Point View

- File View: thể hiện các file và dự án hiện tại đang thực hiện.



Hình 5.45: Giao diện File View

- Teaching: giúp người dùng có thể di chuyển robot đến điểm mong muốn và dạy cho robot.



Hình 5.46: Giao diện Teaching

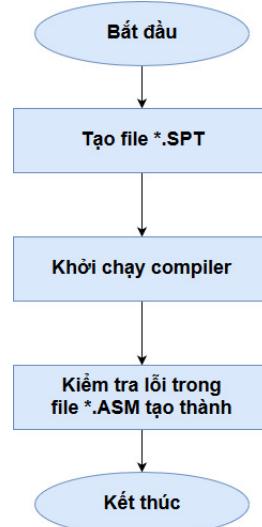
- Result: hiển thị kết quả biên dịch chương trình.



Hình 5.47: Giao diện Result

- Các nút nhán có chức năng:
  - OPEN: mở một file hoặc dự án đã được lưu trong máy tính.
  - SAVE: lưu lại file đang lập trình vào máy tính.
  - COMPILE: biên dịch chương trình đang soạn thảo.
  - RUN: thực thi chương trình chứa trong Flash của board điều khiển.
  - DEBUG: thực thi chương trình chứa trong Flash của board ở chế độ Debug.
  - STEP: chế độ debug theo từng bước của chương trình.
  - LINE: chế độ debug theo từng dòng lệnh.
  - BREAKPOINT: đặt breakpoint vào dòng hiện tại của con trỏ soạn thảo.
  - REMOVE ALL: bỏ tất cả các breakpoint đã đặt.

#### 5.6.4.1 Quá trình biên dịch một dự án

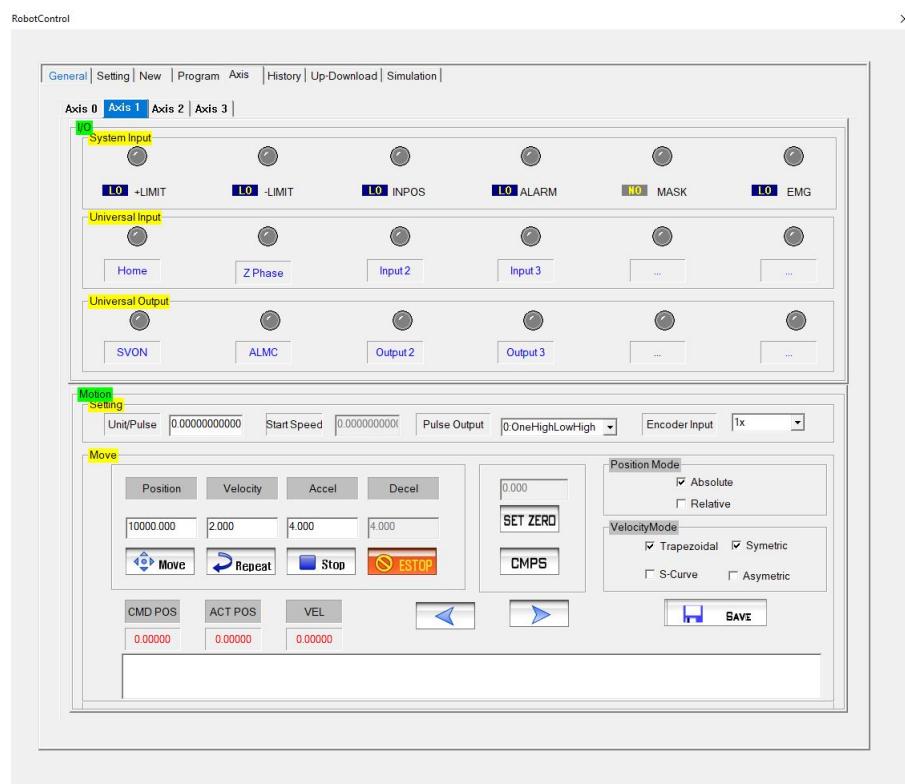


Hình 5.48: Quá trình biên dịch chương trình robot

Dể có thẻ biên dịch file JOB Script đang soạn thảo thành file mã ASM, chương trình thực hiện các bước sau:

1. Tạo file \*.SPT: đọc nội dung từ trình soạn thảo chương trình, sau đó loại bỏ bớt các comment có trong chương trình, đồng thời những dòng trống thì được thay bằng dấu "://" để trình biên dịch hiểu đó là một dòng commnet
2. Cho chương trình khởi chạy trình biên dịch để tạo ra file \*.ASM, \*.LIN, \*.DBG, \*.STR.
3. Lần lượt kiểm tra xem trong file \*.ASM có lỗi không và đưa ra thông báo. Nếu có lỗi thì sẽ thông báo vào mục Result, người dùng có thể nhấn chuột vào dòng lỗi để biết đang bị lỗi ở dòng nào

### 5.6.5 Axis Tab



Hình 5.49: Giao diện trực 1

Chức năng của Axis Tab:

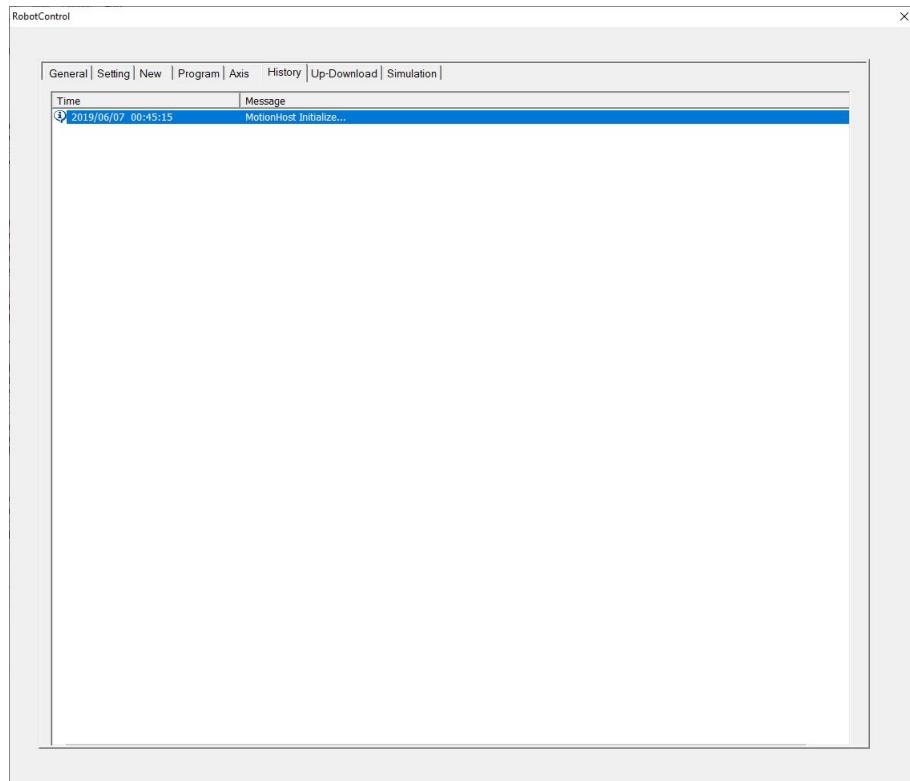
- Cài đặt lại các thông số liên quan đến chuyển động của từng trục.
- Tiến hành chạy thử robot ở chế độ Jog Mode.
- Quản lý các I/O của từng trục cũng như bật/tắt các tín hiệu output của trục như Servo On, Alarm Clear,...

- Các đèn hiển thị ở vùng System Input biểu thị các trạng thái của input từng trực trong quá trình di chuyển và mức kích hoạt của từng tín hiệu được hiển thị thành các nhãn ngay dưới đèn hiển thị.
- Các đèn hiển thị ở vùng Universal Input biểu thị trạng thái ngõ vào chung của các trực.
- Các đèn hiển thị cũng có tác dụng như nút nhấn ở vùng Universal Output, khi nhấn nút thì trạng thái của tín hiệu output bị đảo ngược lại và đèn hiển thị trạng thái tương ứng.
- Vùng Motion cho phép người dùng cài đặt lại các thông số chuyển động của từng trực như UnitperPulse, PulseOutMethod, Encoder Input Method, Position Mode và Velocity Mode. Để board điều khiển có thể nhận được cài đặt mới nhất thì nhấn nút Save để lưu lại giá trị cài đặt mới vào trong Flash của board điều khiển.
- Vùng Move cho phép người dùng thực hiện các chuyển động cơ bản.
  - Nút nhấn Move dùng để thực hiện chuyển động trực với giá trị vị trí được nhập trong ô Position, vận tốc là giá trị được nhập trong ô Velocity, gia tốc là giá trị được nhập trong ô Accel.
  - Nút nhấn Repeat có tác dụng thực hiện chuyển động trực như nút nhấn Move nhưng nút nhấn Repeat sẽ thực hiện mãi cho đến khi người dùng ra lệnh dừng chuyển động.
  - Nút nhấn Stop dùng để dừng chuyển động với gia tốc trong ô Decel.
  - Nút nhấn EStop giúp người dùng thực hiện dừng khẩn cấp chuyển động của trực.
  - Nút nhấn Set Zero để đặt các giá trị CMD POS, ACTPOS, VEL về giá trị 0.
  - 2 nút nhấn mũi tên chiều ngược nhau tương ứng với chế độ Jog Mode theo 2 chiều ngược nhau. Vận tốc chuyển động là vận tốc được nhập trong ô Velocity và gia tốc được nhập trong ô Accel.
  - Vùng hiển thị bên dưới để hiển thị trạng thái của trực khi có sự cố xảy ra hoặc có lỗi trên trực.

### 5.6.6 History Tab

Trong quá trình sử dụng phần mềm, có rất nhiều hoạt động được diễn ra. Do đó, người dùng khó có thể nhớ hết được các công việc đã thực hiện cũng như là kết quả. Mục đích của History Tab là tạo ra giao diện để có thể hiển thị các thông báo về hành động người dùng đã thực hiện và kết quả của quá trình.

Khi một hành động xảy ra, một thông báo sẽ được phát đi tới class của dialog History. Thông báo được phát đi mang mã số thông báo để hộp thoại History có thể biết được đó là thông báo về nội dung gì và hiển thị thông báo lên giao diện.

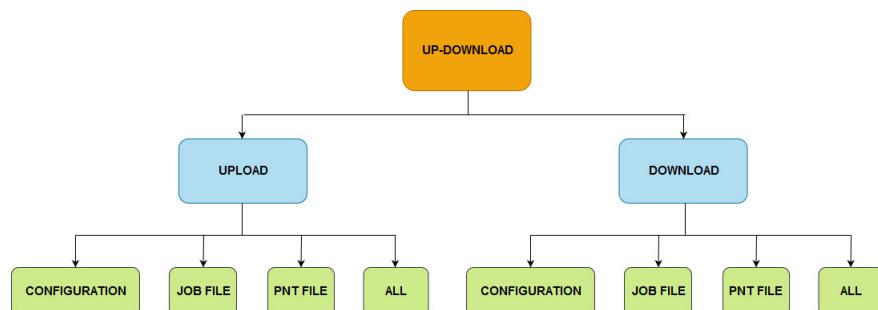


Hình 5.50: Giao diện History

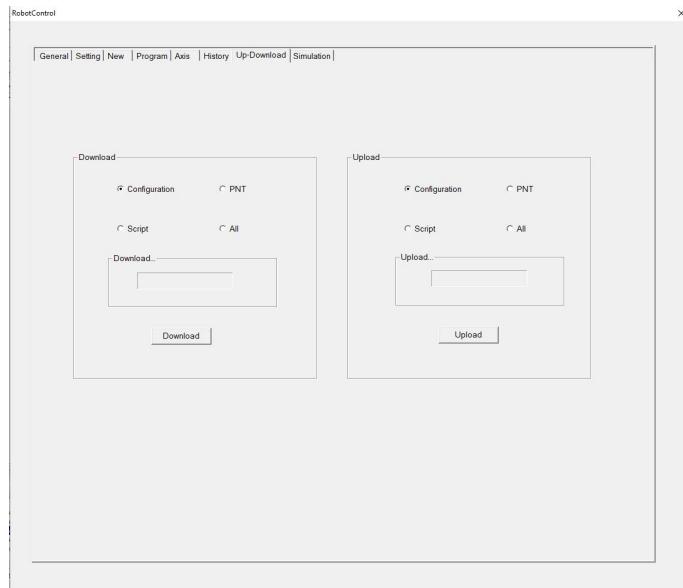
### 5.6.7 Up-Download Tab

Để thiết lập các giá trị thông số cho board điều khiển, sau khi khởi tạo, file lưu giữ giá trị thông số chuyển động cần được tải xuống Flash của board điều khiển. Một cách tương tự, sau khi lập trình robot, để robot có thể hoạt động được, chương trình được biên dịch thành công phải được tải xuống Flash của board điều khiển và sau đó thực thi nếu có tín hiệu điều khiển từ máy tính.

Ngược lại, nếu muốn lấy giá trị hiện tại của robot, chương trình đã được tải xuống ở robot, người dùng có thể sử dụng chức năng upload để tải lên máy tính các cài đặt cũng như chương trình từ board điều khiển. Người dùng có thể tải lên hoặc tải xuống một trong các file cài đặt, JOB File, PNT File hoặc tất cả các file.



Hình 5.51: Sơ đồ tổ chức và chức năng của Upload và Download Tab



Hình 5.52: Giao diện Upload và Download

## 5.7 Phương thức truyền thông giữa board điều khiển và máy tính

### 5.7.1 Chuẩn giao tiếp RS232C

Chuẩn giao tiếp RS232 là một trong những chuẩn giao tiếp được sử dụng rộng rãi hiện nay để nối ghép các thiết bị ngoại vi với máy tính. Nó là một chuẩn giao tiếp truyền nhận dữ liệu nối tiếp, kết nối nhiều nhất là 2 thiết bị.

RS232 sử dụng phương thức truyền thông không đối xứng, tức là sử dụng tín hiệu điện áp chênh lệch giữa một dây với đất. Mức điện áp của tiêu chuẩn RS232C được mô tả:

- Mức logic 0: +3V → +12V
- Mức logic 1: -12V → -3V

Các mức điện áp trong phạm vi từ -3V đến 3V là trạng thái chuyển tuyến. Chính vì phạm vi -3V đến 3V không được định nghĩa, trong trường hợp thay đổi giá trị logic từ mức thấp lên cao hoặc ngược lại, một tín hiệu phải vượt qua ngưỡng quá độ trong một thời gian ngắn hợp lý. Điều này dẫn đến hạn chế về điện dung của các thiết bị tham gia và cả của đường truyền. Tốc độ tối đa phụ thuộc vào chiều dài dây dẫn.

Chuẩn giao tiếp RS232 truyền dữ liệu theo khung dữ liệu. Việc truyền dữ liệu được thực hiện không đồng bộ nên trong một thời điểm chỉ có một bit được truyền. Khung truyền (frame) bao gồm 1 start bit, các bit dữ liệu, bit kiểm tra chẵn lẻ và cuối cùng là stop bit, có thể là 1, 1,5 hay 2 stop bit.

### 5.7.2 Khung truyền và nhận dữ liệu ở máy tính khi giao tiếp với board điều khiển

Board điều khiển giao tiếp với máy tính qua chuẩn RS-232C.

Khung dữ liệu truyền thông có dạng:

<b>STX(0x02)</b>	<b>Command</b>	,	<b>Argurment_1</b>	,	...	,	<b>Argurment_n</b>	;	<b>CRC</b>	<b>ETX(0x03)</b>
------------------	----------------	---	--------------------	---	-----	---	--------------------	---	------------	------------------

Hình 5.53: Khung dữ liệu truyền thông

Khung dữ liệu được bắt đầu bằng kí tự bắt đầu STX, tiếp theo là tên lệnh, các đối số được ngăn cách nhau bằng kí tự ",", kế tiếp là CRC để kiểm tra lỗi và kí tự kết thúc ETX.

Quy tắc tạo ra CRC để kiểm tra lỗi:

```
unsigned short MakeCrc(unsigned short* wPacketData, uint16 usPacketSize)
{
    unsigned short wCheckSum = 0;
    for (int i = 0; i < usPacketSize ; i++)
    {
        wCheckSum = wCheckSum ^ wPacketData[ i ];
    }
    return wCheckSum;
}
```

Dịnh dạng khung truyền một số lệnh dùng để giao tiếp giữa board và máy tính:

- Xóa bộ đệm truyền thông:
  - Truyền: Format("%cRXBC;%d%c", STX, CRC, ETX)
  - Nhận: Format("%cACK,OK;%d%c", STX, CRC, ETX)
- Bắt đầu thực hiện chương trình 0:
  - Truyền: Format("%cTRUN,0;%d%c", STX, CRC, ETX)
  - Nhận: Format("%cACK,OK;%d%c", STX, CRC, ETX)
- Dừng thực hiện chương trình 0:
  - Truyền: Format("%cTSTP,0;%d%c", STX, CRC, ETX)
  - Nhận: Format("%cACK,OK;%d%c", STX, CRC, ETX)
- Dừng thực hiện chương trình 0:
  - Truyền: Format("%cTSTP,0;%d%c", STX, CRC, ETX)
  - Nhận: Format("%cACK,OK;%d%c", STX, CRC, ETX)

- Đừng khẩn cấp:
  - Truyền: Format("%cAXET;%d%c", STX, CRC, ETX)
  - Nhận: Format("%cACK,OK;%d%c", STX, CRC, ETX)
- Đọc trạng thái của các Output Motion:
  - Truyền: Format("%cASTS;%d%c", STX, CRC, ETX)
  - Nhận: Format("%cASTS,%d;%d%c", STX, dwTotalAxisStatus, CRC, ETX)
- Đọc trạng thái của các tín hiệu Mechanical:
  - Truyền: Format("%cMSIG;%d%c", STX, CRC, ETX)
  - Nhận: Format("%cMSIG,%d;%d%c", STX, dwTotalMechanicalStatus, CRC, ETX)
  - dwTotalMechanicalStatus nhận được là trạng thái hiện tại của các tín hiệu Mechanical như: Limit, Alarm, Emergency,...
- Cài đặt mức tích cực cho các tín hiệu Mechanical:
  - Truyền: Format("%cMLVL;%d%c", STX, CRC, ETX)
  - Nhận: Format("%cMLVL,%d;%d%c", STX, dwTotalMechanicalLevel, CRC, ETX)
  - dwTotalMechanicalLevel là trạng thái tích cực cần cài đặt cho các tín hiệu Mechanical: Limit, Alarm, Emergency,...
- Kiểm tra trạng thái của hệ thống:
  - Truyền: Format("%cSSTS;%d%c", STX, CRC, ETX)
  - Nhận: Format("%cSSTS,%d;%d%c", STX, dwTotalSystemStatus, CRC, ETX)
  - dwTotalSystemStatus là trạng thái hiện tại của mỗi trục bao gồm: Servo On, Limit, Alarm Clear, In Position, Emergency,
- Lấy thông số về Encoder Method và Pulse Out Method:
  - Truyền: Format("%cENPL;%d%c", STX, CRC, ETX)
  - Nhận: Format("%cENPL,%d%d%d%d,%d%d%d%d,%d%d%d%d,%d%d%d%d;%d%c", STX, nMethod1[0], nMethod2[0], nAbsRelMode[0], nProfileMode[0], nMethod1[1], nMethod2[1], nAbsRelMode[1], nProfileMode[1], nMethod1[2], nMethod2[2], nAbsRelMode[2], nProfileMode[2], npMethod1[3], nMethod2[3], nAbsRelMode[3], nProfileMode[3], CRC, ETX)
- Cài đặt Encoder Method:
  - Truyền: Format("%cSENC,%d,%d;%d%c", STX, nAxis, nMethod, CRC, ETX)
 

nAxis là trục cần cài đặt, nMethod là phương thức đọc Encoder bao gồm: Up/Down(0), 1x(1), 2x(2), 4x(3), Reserve Up/Down(4), Reserve 1x(5), Reserve 2x(6), Reserve 4x(7).

- Nhận: Format("%cACK,OK;%d%c", STX, CRC, ETX)
- Cài đặt Pulse Out Method:
  - Truyền: Format("%cSPUL,%d,%d;%d%c", STX, nAxis, nMethod, CRC, ETX)
 

nAxis là trục cần cài đặt, nMethod là phương thức xuất xung bao gồm: One-HighLowHigh(0), OneHighHoghLow(1), OneLowLowHigh(2), OneLowHigh-Low(3), TwoCewCwHigh(4), TwoCcwCwLow(5), TowCwCewHigh(6), TowCwC-cwLow(7), TwoPhase(8), TwoPhaseReverse(9).
  - Nhận: Format("%cACK,OK;%d%c", STX, CRC, ETX)
- Cài đặt cách xác định vị trí Abs/Rel:
  - Truyền: Format("%cSABR,%d,%d;%d%c", STX, nAxis, nAbsRelMode, CRC, ETX)
 

nAxis là trục cần cài đặt, nAbsRelMode là kiểu vị trí tương ứng với 0 là Absolute và 1 là Relative.
  - Nhận: Format("%cACK,OK;%d%c", STX, CRC, ETX)
- Cài đặt profile vận tốc chuyển động:
  - Truyền: Format("%cSPRF,%d%d;%d%c", STX, nAxis, nProfileMode, CRC, ETX)
 

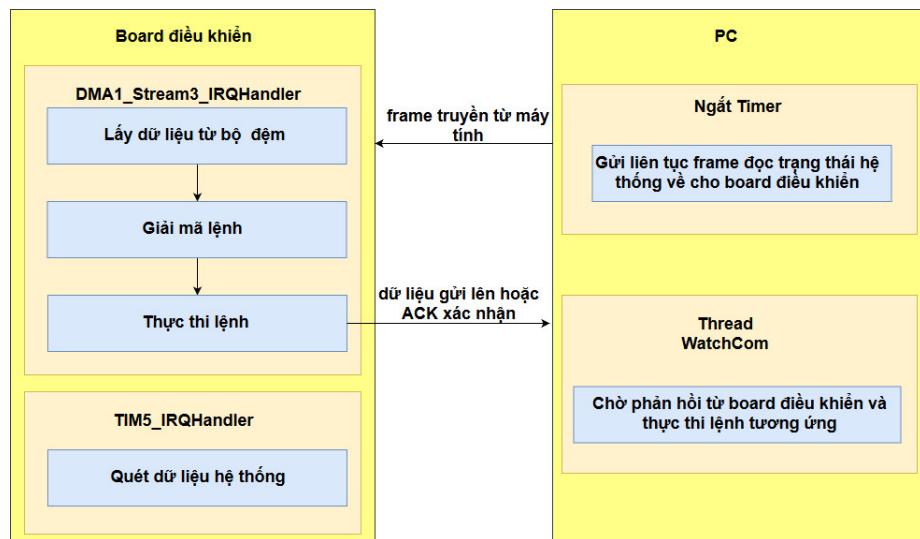
nAxis là trục cần cài đặt, nProfileMode là kiểu profile vận tốc tương ứng với 0 là vận tốc hình thang dạng đối xứng, 1 là dạng vận tốc hình thang không đối xứng, 3 là vận tốc hình chữ S đối xứng và 4 là dạng vận tốc hình chữ S không đối xứng.
  - Nhận: Format("%cACK,OK;%d%c", STX, CRC, ETX)
- Kiểm tra UnitperPulse hiện tại:
  - Truyền: Format("%cUTPL;%d%c", STX, CRC, ETX)
  - Nhận: Format("%cUTPL,%f,%f,%f,%f;%d%c", STX, dUnitPerPulse[0], dUnitPerPulse[1], dUnitPerPulse[2], dUnitPerPulse[3], CRC, ETX)
- Cài đặt UnitperPulse:
  - Truyền: Format("%cSUTP,%d,%f;%d%c", STX, nAxis, dUnitPulse, CRC, ETX)
 

nAxis là trục cần cài đặt, dUnitPulse là giá trị UnitperPulse.
  - Nhận: Format("%cACK,OK;%d%c", STX, CRC, ETX)
- Kiểm tra giá trị vị trí được tính toán và vận tốc:

- Truyền: Format("%cPSVL;%d%c", STX, CRC, ETX)  
nAxis là trục cần cài đặt, dUnitPulse là giá trị UnitperPulse.
  - Nhận: Format("%cPSVL,%.3f,.3f,.3f,.3f,.3f,.3f,.3f,.3f,.3f,.3f,.3f;.3f;%d%c", STX, dPos[0][0], dPos[1][0], dVel[0], dPos[0][1], dPos[1][1], dVel[1], dPos[0][2], dPos[1][2], dVel[2], dPos[0][3], dPos[1][3], dVel[3], CRC, ETX)
- Cài đặt giá trị vận tốc:
  - Truyền: Format("%cSVEL,%f,%f,%f,%f;%d%c", STX, dVel[0], dVel[1], dVel[2], dVel[3], CRC, ETX)
  - Nhận: Format("%cACK,OK;%d%c", STX, CRC, ETX)
- Cài đặt giá trị gia tốc tăng:
  - Truyền: Format("%cSACC,%f,%f,%f,%f;%d%c", STX, dAccel[0], dAccel[1], dAccel[2], dAccel[3], CRC, ETX)
  - Nhận: Format("%cACK,OK;%d%c", STX, CRC, ETX)
- Cài đặt giá trị gia tốc giảm:
  - Truyền: Format("%cSDEC,%f,%f,%f,%f;%d%c", STX, dDecel[0], dDecel[1], dDecel[2], dDecel[3], CRC, ETX)
  - Nhận: Format("%cACK,OK;%d%c", STX, CRC, ETX)
- Bật/tắt tín hiệu Servo On;
  - Truyền: Format("%cSVON,%d,%d;%d%c", STX, nAxis, nFlag, CRC, ETX)  
nAxis là thứ tự trục, nFlag là giá trị cần cài đặt.
  - Nhận: Format("%cACK,OK;%d%c", STX, CRC, ETX)
- Bật/tắt tín hiệu Alarm Clear:
  - Truyền: Format("%cALMC,%d,%d;%d%c", STX, nAxis, nFlag, CRC, ETX)  
nAxis là thứ tự trục, nFlag là trạng thái cần cài đặt.
  - Nhận: Format("%cACK,OK;%d%c", STX, CRC, ETX)
- Xóa giá trị vị trí đã xuất xung về 0:
  - Truyền: Format("%cSZRO,%d;%d%c", STX, nAxis, CRC, ETX)  
nAxis là thứ tự trục.
  - Nhận: Format("%cACK,OK;%d%c", STX, CRC, ETX)

### 5.7.3 Giao tiếp giữa board điều khiển và máy tính

Giao tiếp giữa máy tính và board điều khiển được kiểm soát bởi một Thread. Máy tính sau khi gửi lệnh xuống board điều khiển thì sẽ nhận lại ACK xác nhận đã hoàn thành hoặc là mã lệnh tương ứng và giá trị cần lấy. Trường hợp dữ liệu nhận được từ board điều khiển là mã lệnh thì các mã lệnh nhận từ board điều khiển gửi lên sẽ được kiểm tra để xác định lệnh và thực hiện công việc tương ứng trong thread.

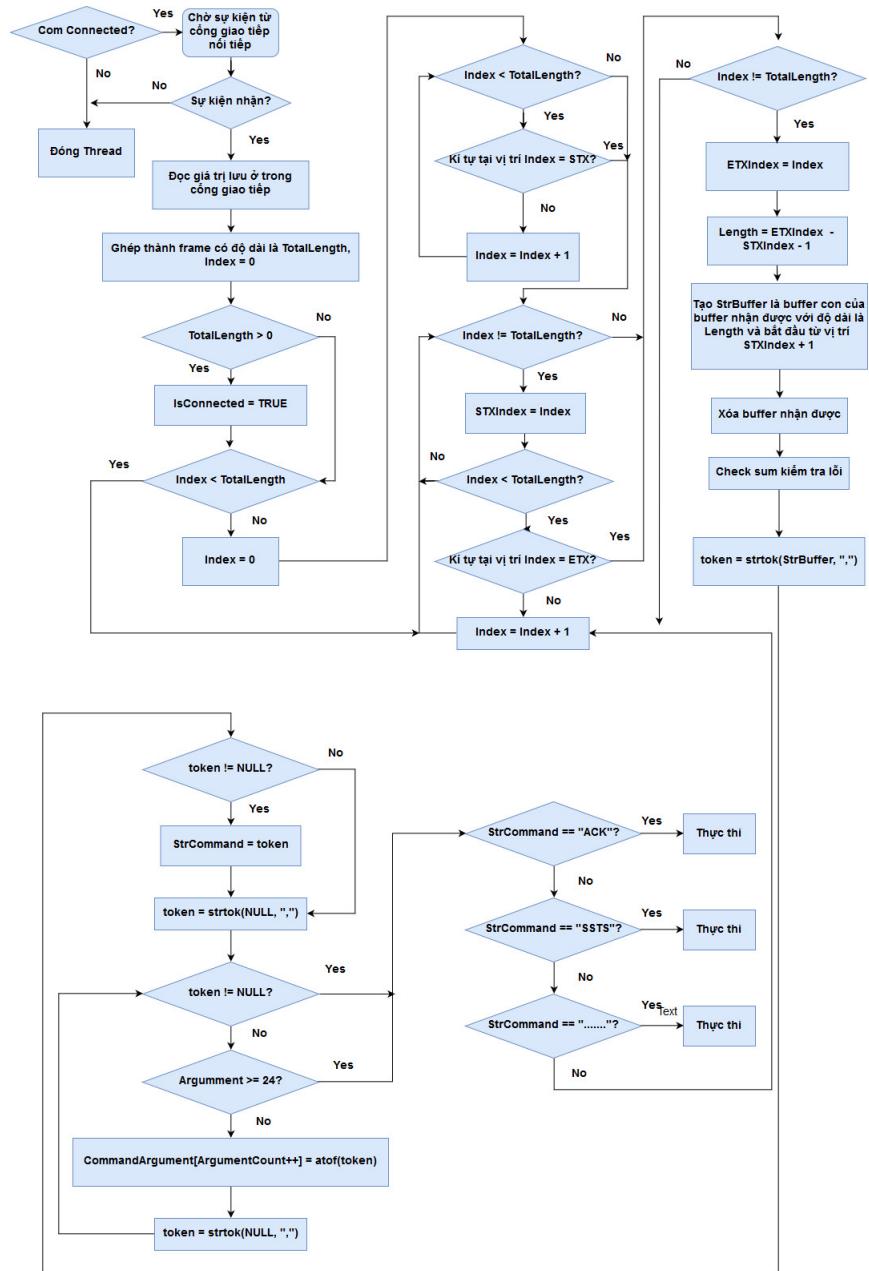


Hình 5.54: Phương thức giao tiếp giữa board và máy tính

Trong quá trình thực hiện chương trình, phần mềm ở máy tính thực hiện các lệnh quét trong hàm ngắt Timer để đọc các trạng thái hiện tại của hệ thống, các tín hiệu Mechanical. Sau một thời gian, nếu board điều khiển không phản hồi thì sẽ yêu cầu gửi lại.

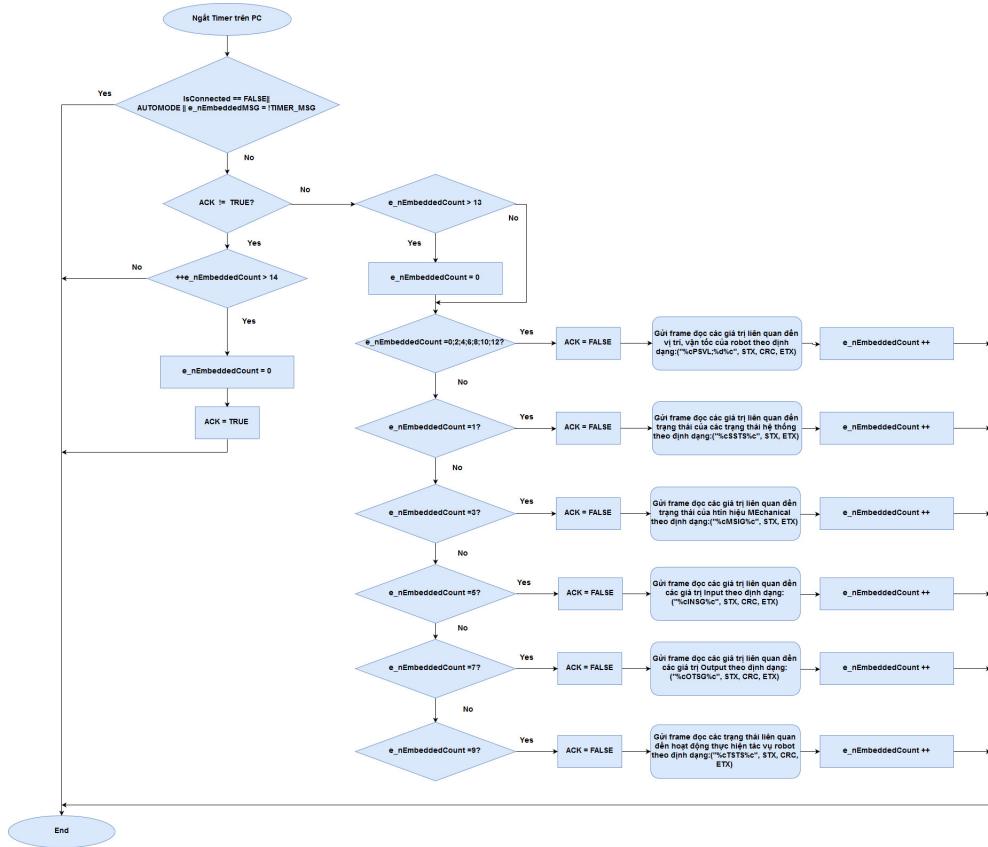
Board điều khiển có CPU là chip ARM Cortex-M3. Khi nhận được frame chứa lệnh mới thì ở board điều khiển sẽ xảy ra ngắt truyền thông. Sau đó, frame lệnh sẽ được giải mã và thực thi lệnh tương ứng. Trong board điều khiển, ngắt timer được sử dụng để lấy giá trị liên quan đến chuyển động hoặc trạng thái của board một cách liên tục sau đó gửi lên lại cho máy tính. **Trường hợp lệnh không có giá trị gửi lên, board sẽ gửi ACK lên máy tính để xác nhận đã thực thi lệnh thành công.** Ngược lại, trong trường hợp lệnh thực hiện không thành công, thì lần ngắt Timer sau ở máy tính, lệnh đó sẽ lại tiếp tục được gửi xuống board. Việc làm này đảm bảo cho việc giao tiếp giữa máy tính và board điều khiển đạt tốc độ cao và không xảy ra tình trạng bị treo.

Quá trình Thread Watchcom để đọc và xử lý giá trị nhận về ở máy tính:



Hình 5.55: Sơ đồ thực thi của Thread WatchCom

## Chương trình ngắt phục vụ timer trên máy tính:



Hình 5.56: Chương trình phục vụ ngắt Timer trên máy tính

## Chương 6

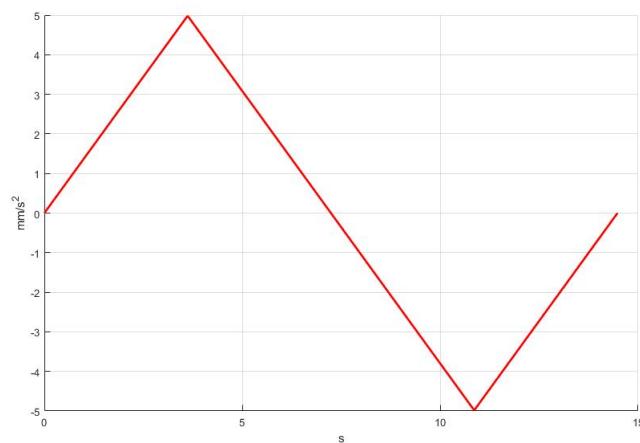
# KẾT QUẢ VÀ HƯỚNG PHÁT TRIỂN

### 6.1 Kết quả

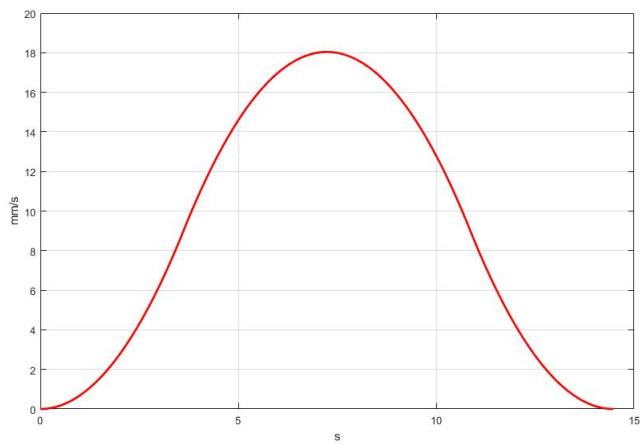
- Robot có thể di chuyển theo quỹ đạo: đoạn thẳng, đường tròn,... có độ chính xác về vị trí, chuyển động êm, không bị rung lắc khi chuyển động.
- Hệ thống hoạt động ổn định.
- Phần mềm có khả năng điều khiển robot và mô phỏng.

Một số kết quả thu được trong quá trình thử nghiệm:

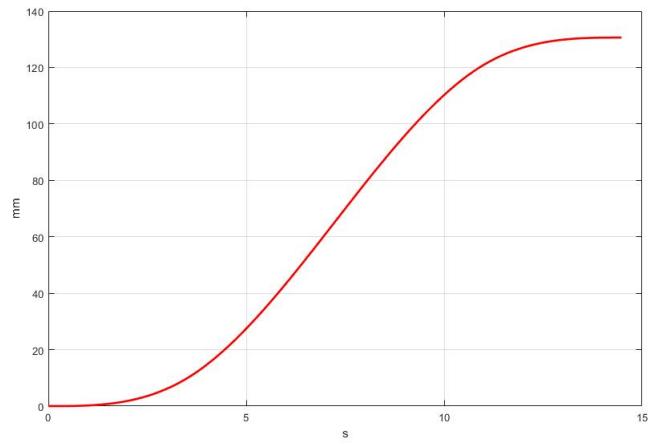
1. Robot di chuyển theo đường thẳng từ vị trí  $p_i = [194.134 \ 280.5 \ 0]$  đến vị trí  $p_f = [200 \ 150 \ 0]$ :



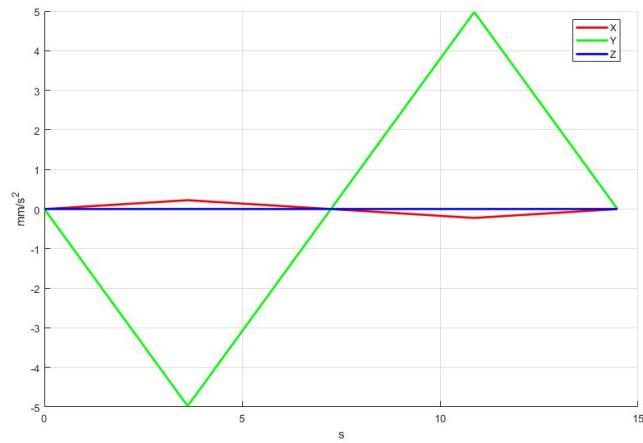
Hình 6.1: Đồ thị biểu diễn gia tốc end-effector trên quỹ đạo



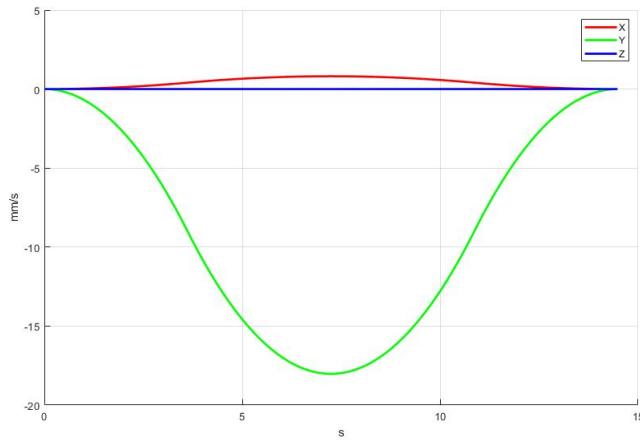
Hình 6.2: Đồ thị biểu diễn vận tốc end-effector trên quỹ đạo



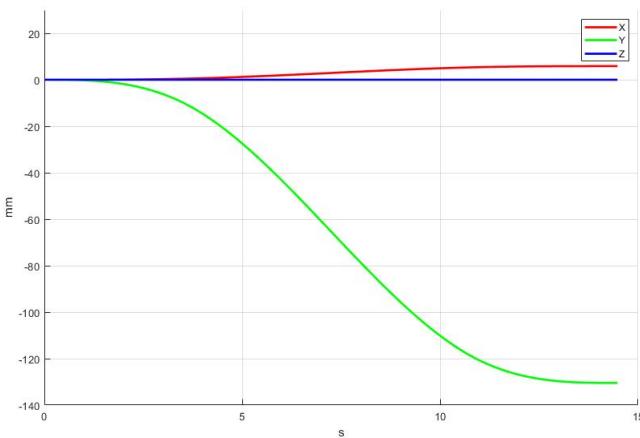
Hình 6.3: Đồ thị biểu diễn độ dời end-effector trên quỹ đạo



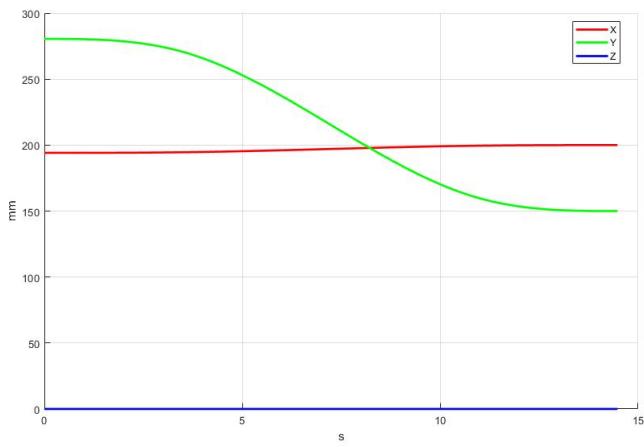
Hình 6.4: Đồ thị biểu diễn gia tốc end-effector trên mỗi trục tọa độ



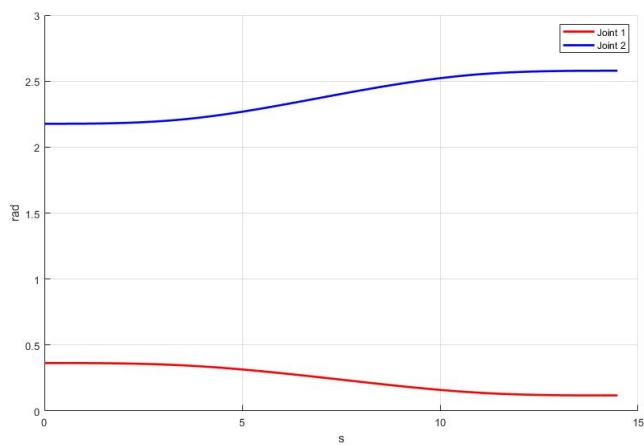
Hình 6.5: Đồ thị biểu diễn vận tốc end-effector trên mỗi trục tọa độ



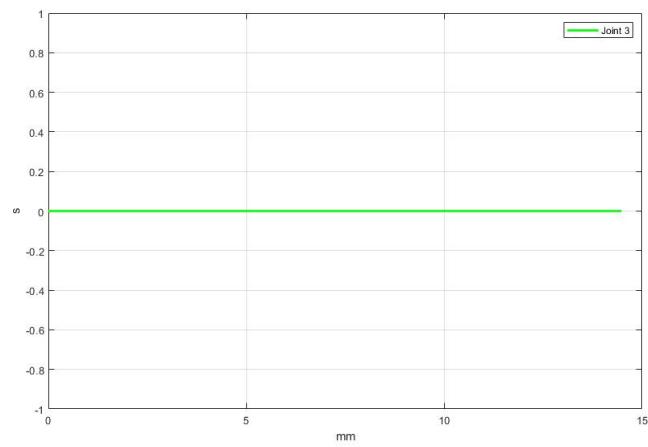
Hình 6.6: Đồ thị biểu diễn độ dời end-effector trên mỗi trục tọa độ



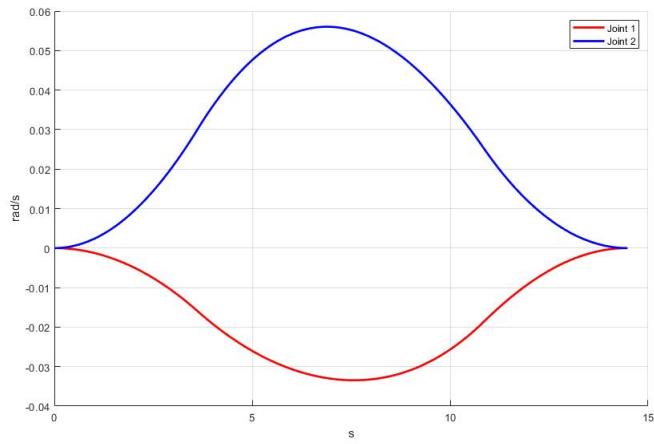
Hình 6.7: Đồ thị biểu diễn vị trí end-effector trên mỗi trục tọa độ



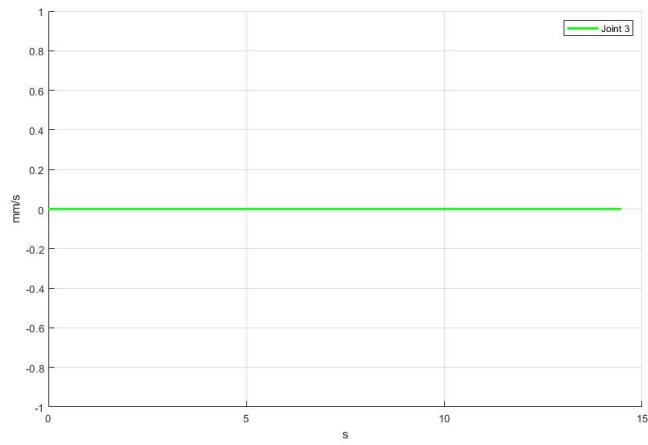
Hình 6.8: Giá trị biến khớp 1 và 2



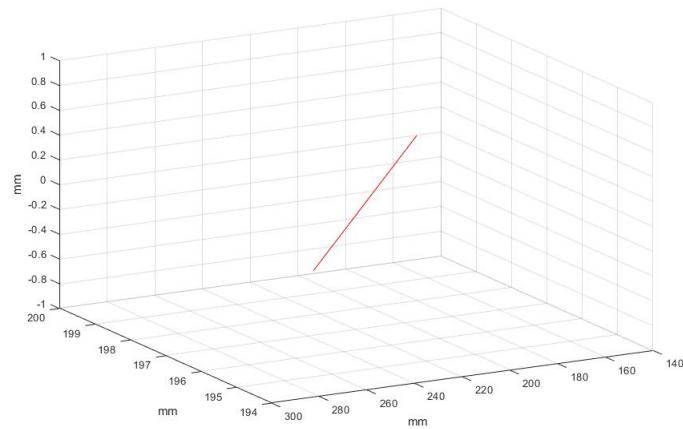
Hình 6.9: Đồ thị biểu diễn giá trị khớp 3



Hình 6.10: Đồ thị biểu diễn vận tốc biến khớp 1 và khớp 2

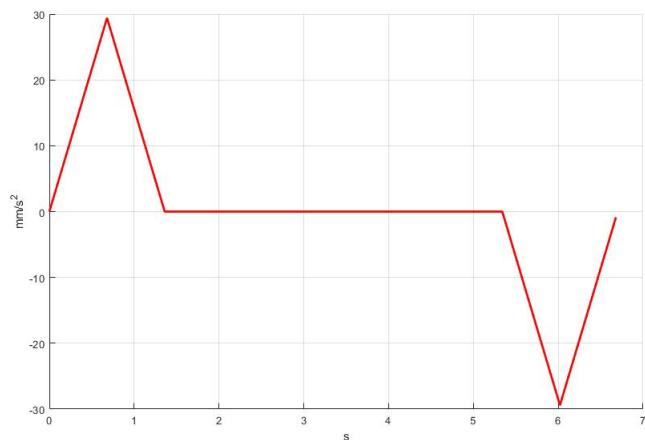


Hình 6.11: Đồ thị biểu diễn vận tốc biến khớp 3

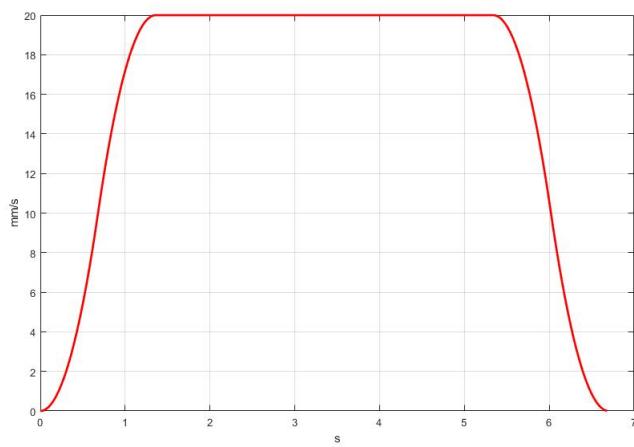


Hình 6.12: Đường đi trong không gian

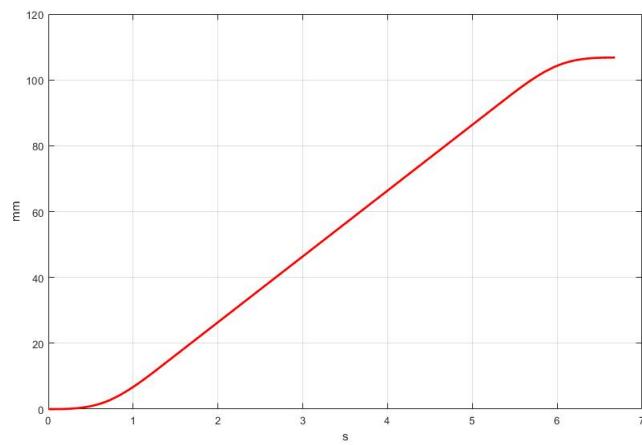
2. Robot di chuyển theo đường thẳng từ vị trí  $p_i = [194.134 \ -80.4669 \ 0]$  đến vị trí  $p_f = [270 \ -11.564 \ -30]$ :



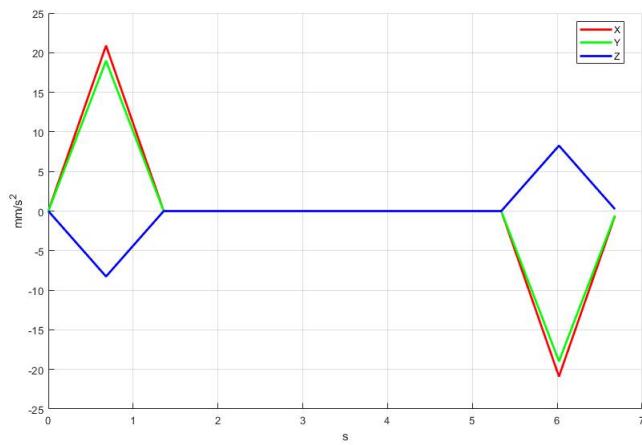
Hình 6.13: Đồ thị biểu diễn gia tốc end-effector trên quỹ đạo



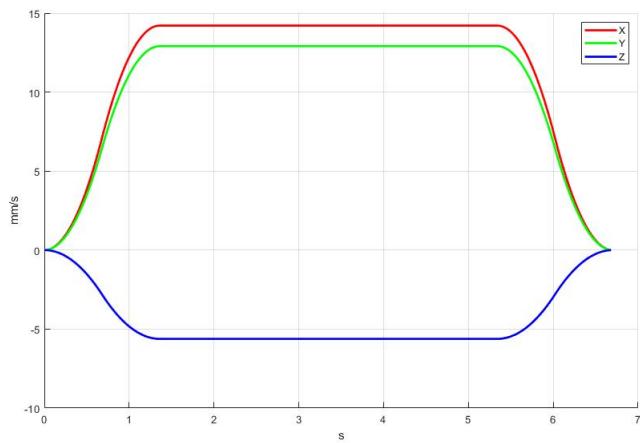
Hình 6.14: Đồ thị biểu diễn vận tốc end-effector trên quỹ đạo



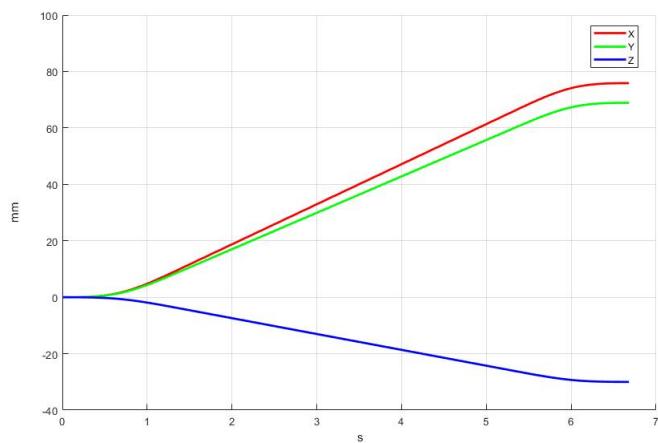
Hình 6.15: Đồ thị biểu diễn độ dời end-effector trên quỹ đạo



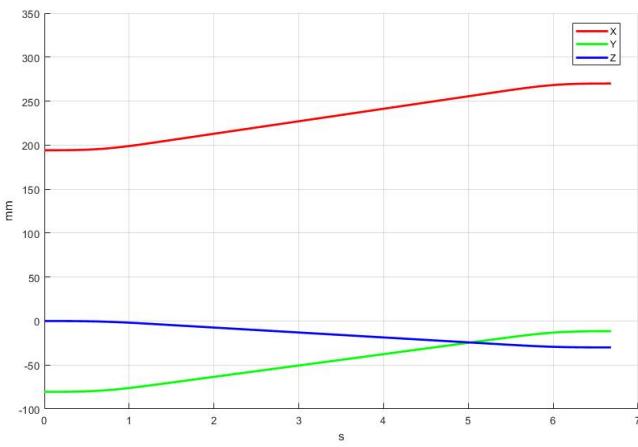
Hình 6.16: Đồ thị biểu diễn gia tốc end-effector trên mỗi trục tọa độ



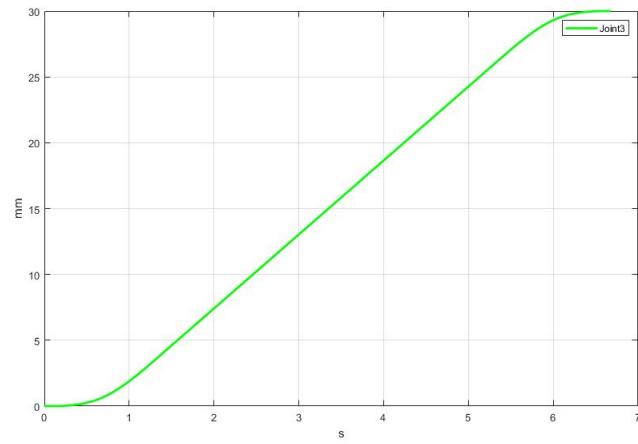
Hình 6.17: Đồ thị biểu diễn vận tốc end-effector trên mỗi trục tọa độ



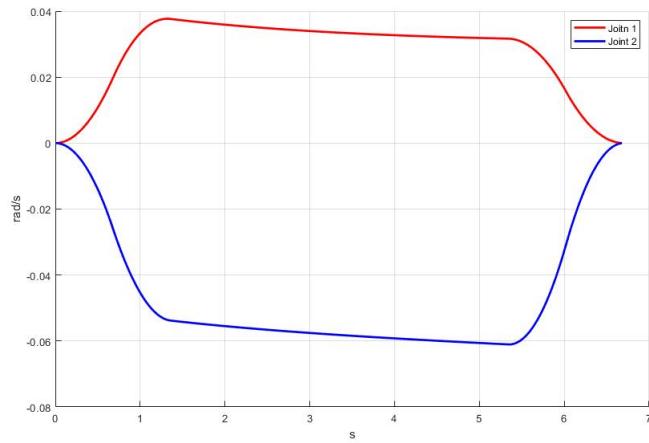
Hình 6.18: Đồ thị biểu diễn độ dời end-effector trên mỗi trục tọa độ



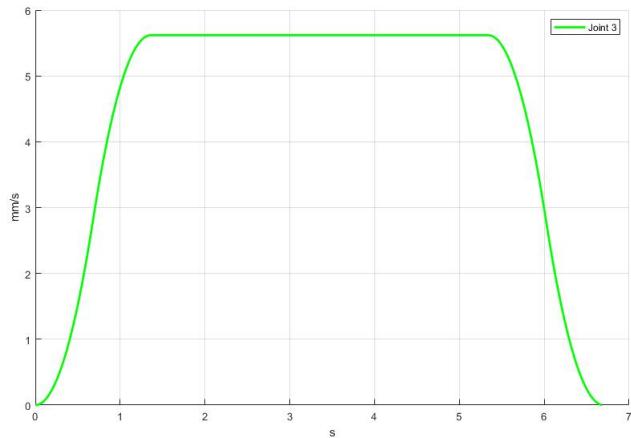
Hình 6.19: Đồ thị biểu diễn vị trí end-effector trên mỗi trục tọa độ



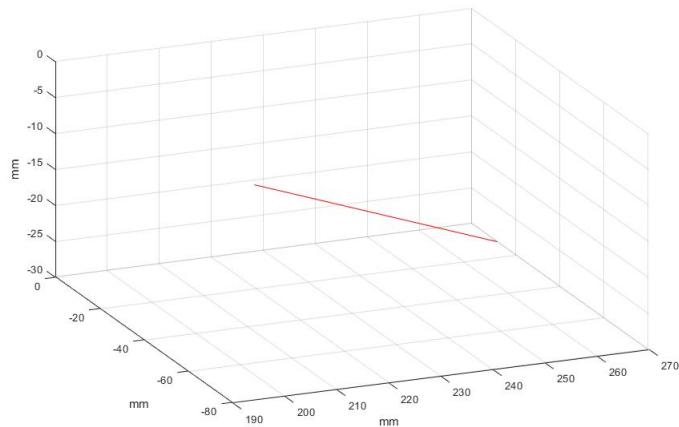
Hình 6.20: Đồ thị biểu diễn giá trị khớp 3



Hình 6.21: Đồ thị biểu diễn vận tốc biến khớp 1 và khớp 2

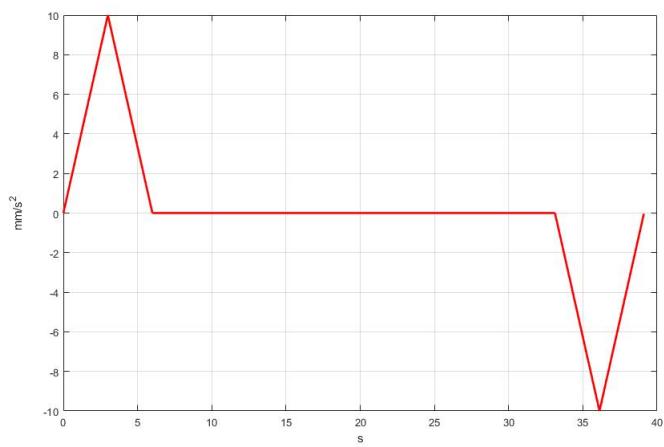


Hình 6.22: Đồ thị biểu diễn vận tốc biến khớp 3

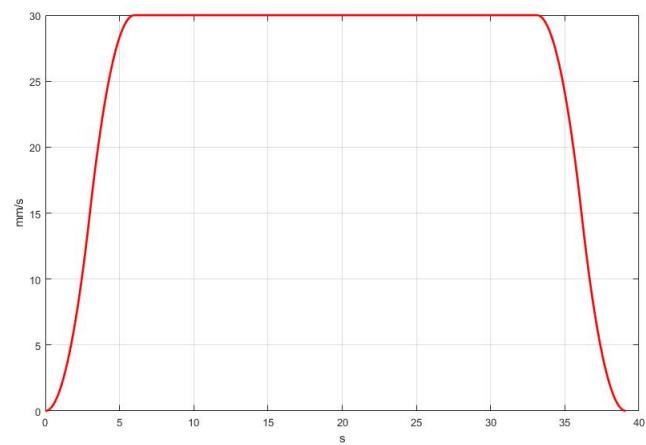


Hình 6.23: Đường đi trong không gian

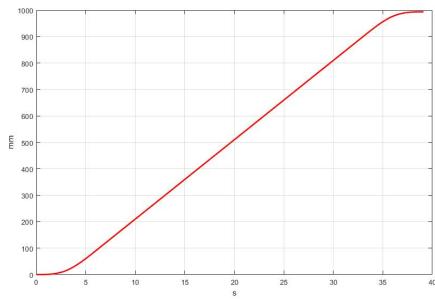
3. Robot di chuyển theo quỹ đạo đường tròn qua 3 điểm  $P_0 = [200 \ 100 \ 100]$ ,  $P_1 = [300 \ 200 \ 100]$   $P_2 = [500 \ 100 \ 100]$  với vận tốc tối đa là 30 mm/s và gia tốc tối đa là 10  $\text{mm}/\text{s}^2$  :



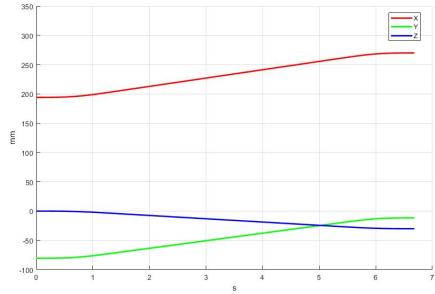
Hình 6.24: Đồ thị biểu diễn gia tốc end-effector trên quỹ đạo



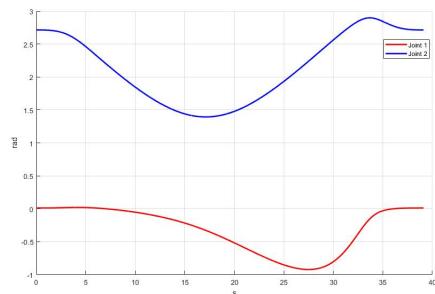
Hình 6.25: Đồ thị biểu diễn vận tốc end-effector trên quỹ đạo



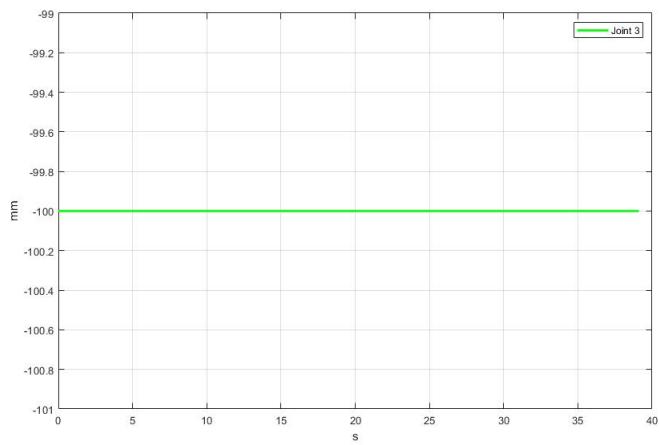
Hình 6.26: Đồ thị biểu diễn độ dời end-effector trên quỹ đạo



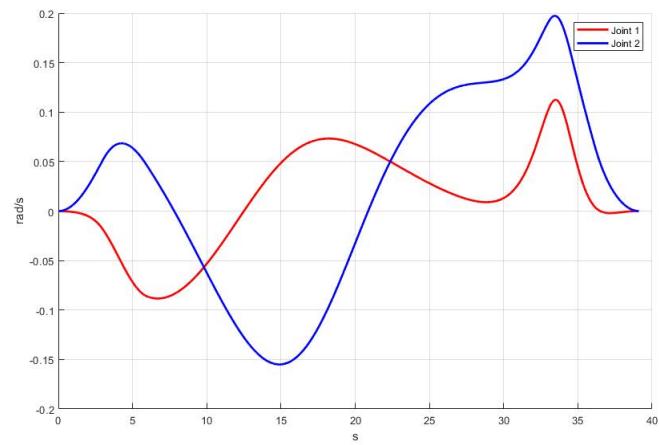
Hình 6.27: Đồ thị biểu diễn vị trí end-effector trên mỗi trục tọa độ



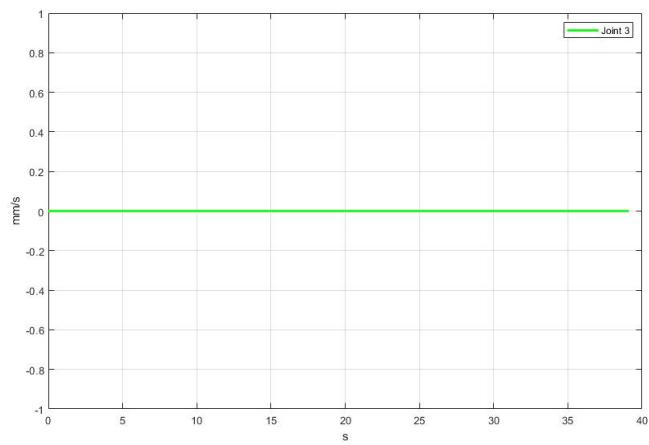
Hình 6.28: Đồ thị biểu diễn giá trị khớp 1 và khớp 2



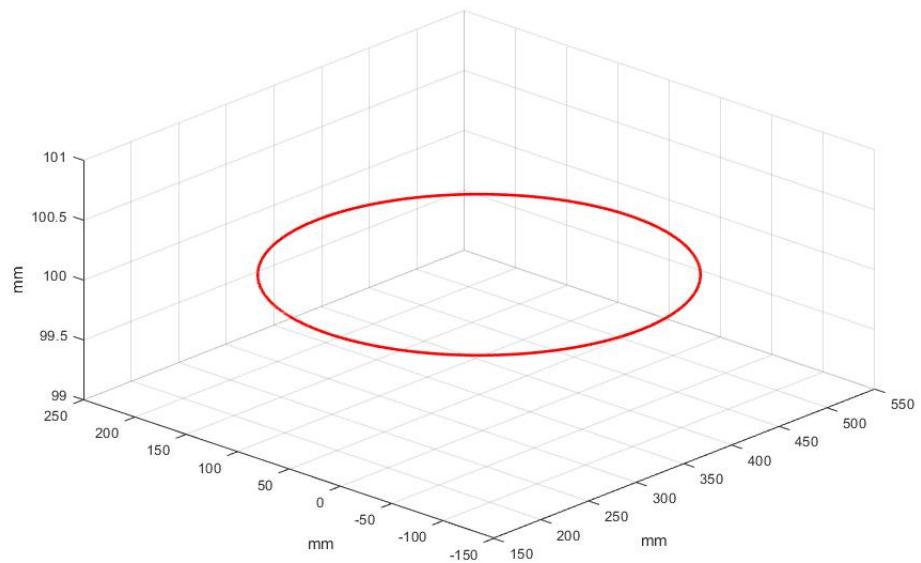
Hình 6.29: Đồ thị biểu diễn giá trị khớp 3



Hình 6.30: Đồ thị biểu diễn vận tốc biến khớp 1 và khớp 2



Hình 6.31: Đồ thị biểu diễn vận tốc biến khớp 3



Hình 6.32: Đường đi trong không gian

## 6.2 Hướng phát triển

- Nâng cao kĩ thuật đồ họa.
- Nâng cao hiệu suất trong việc truyền thông giữa máy tính và board điều khiển.
- Bổ sung thêm các tính năng tương tự như các phần mềm khác trong công nghiệp.

Tài liệu tham khảo:

# Tài liệu tham khảo

- [1] TS. Phạm Đăng Phước, *Robot công nghiệp*.
- [2] Prof. Alessandro De Luca, *Trajectory planning in Cartesian space*.
- [3] Zhang Yongde<sup>1</sup>, Wei Chung<sup>1</sup>, Jiang Jingang<sup>1</sup>, Jiang Jixiong<sup>1</sup>, Liu Yi<sup>2</sup> and Wang Yong, *Motion Planning for Archwire Bending Robot in Orthodontic Treatments*.
- [4] Haihua Mu<sup>1</sup>, Han Chen<sup>1</sup>, and Yu Zhu<sup>1</sup> 1Department of Mechanical Engineering Tsinghua University Beijing, China, *SIMPLE AND ADAPTIVE DISCRETE-TIME ERROR CORRECTION FOR THE ULTRA-PRECISION S-CURVE MOTION PROFILE GENERATION*.