

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA THÀNH PHỐ HỒ CHÍ MINH
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH



ĐỒ ÁN KỸ THUẬT MÁY TÍNH

**GIAO TIẾP VÀ ĐIỀU KHIỂN THIẾT BỊ
DỰA TRÊN OPC-UA**

NGÀNH: KỸ THUẬT MÁY TÍNH

Hội đồng:

CBHD: TS. LÊ TRỌNG NHÂN

CBPB:

Sinh viên 1: Trần Ngọc Cát 1912750

Sinh viên 2: Diệp Trần Nam 1914213

Sinh viên 3: Nguyễn Văn Trọng 1915677

TP. Hồ Chí Minh, tháng 9 năm 2022

LỜI CAM KẾT

Chúng tôi cam kết rằng Đồ Án này dựa trên ý tưởng và kiến thức của những người giám sát của chúng tôi. Tất cả các nghiên cứu và dữ liệu chưa được công bố. Các tài liệu tham khảo, các số liệu và thống kê là đáng tin cậy và trung thực. Nhóm đã hoàn thành các yêu cầu của Đồ án Kỹ thuật Máy tính do khoa Khoa học và Kỹ thuật Máy tính đề ra.

Trân trọng,

Trần Ngọc Cát
Diệp Trần Nam
Nguyễn Văn Trọng

LỜI CẢM ƠN

TÓM LUỢC

Trong những năm gần đây, sự phát triển của giao tiếp Machine-to-Machine (M2M) và Internet of Things (IoT) trong Công nghiệp đã mở đường cho việc phát triển các máy móc thông minh và tự điều khiển trong các doanh nghiệp sản xuất. Các máy tự động này có khả năng thu thập dữ liệu từ các thiết bị cảm biến khác nhau và các hệ thống máy được kết nối khác bằng cách sử dụng các giao thức truyền thông. Và để việc giao tiếp giữa các thiết bị được hiệu quả, nhanh chóng và đảm bảo về thời gian, cần phải chọn một giao thức kết nối phù hợp.

Hiện tại, những máy này có khả năng gửi và nhận dữ liệu đủ để đưa ra quyết định một cách tự chủ. Tuy nhiên, giao tiếp M2M ở cấp độ thấp như hiện tại thì chỉ dùng cho việc xử lý dữ liệu, và các hành động ra quyết định ở cấp độ cao hơn bị giới hạn trong một hệ thống máy riêng lẻ thay vì giao tiếp giữa các máy được kết nối với nhau.

Từ nhu cầu cấp thiết về một loại giao thức truyền thông phù hợp cho môi trường Công nghiệp, đáp ứng sự đồng bộ cao, tốc độ nhanh và giải quyết được những nhược điểm đã nêu trên thì OPC-UA - một giao thức truyền tải dữ liệu chuẩn công nghiệp đã được đề xuất. Những hạn chế trong giao tiếp Machine-to-Machine sẽ được cải thiện bằng cách sử dụng giao thức OPC UA, giao thức liên quan đến việc đưa ra quyết định trong việc phản hồi dữ liệu nhận được và ra lệnh cho các máy được kết nối theo chiều dọc (máy cấp cao hơn) và chiều ngang (máy cùng cấp). Khi làm như vậy, các nhà máy sẽ trở nên thông minh hơn thông qua giao tiếp M2M chủ động hơn. Vì thế, Đồ án này sẽ tập trung vào việc nghiên cứu và hiện thực **Giao tiếp và điều khiển thiết bị dựa trên OPC-UA**. Ngoài ra, để kiểm chứng được ưu thế của OPC-UA trong môi trường Công nghiệp cũng như có được một cái nhìn tổng quát hơn, thì Đồ án này cũng sẽ trình bày quá trình so sánh hiệu suất về thời gian gửi và nhận tin nhắn của hai giao thức nổi tiếng trong lĩnh vực IoT và IoT công nghiệp (IIoT): MQTT và OPC-UA.

Các từ khóa : OPC-UA, MQTT, M2M, IoT, Công nghiệp 4.0, So sánh giao thức, Cánh tay Robot.

Mục lục

Chapter 1. GIỚI THIỆU TỔNG QUAN	2
1.1 Tổng quan về Công nghiệp 4.0	2
1.2 Các ứng dụng công nghệ Công nghiệp 4.0	6
1.3 Smart factory	12
1.3.1 Định nghĩa và lợi ích của Smart Factory	13
1.3.2 Kiến trúc của Smart Factory	14
1.3.3 Thách thức của lớp kết nối	16
1.4 Giới thiệu về Dồ Án	18
Chapter 2. TỔNG QUAN VỀ CÁC GIAO THỨC M2M TRONG CÔNG NGHIỆP 4.0	20
2.1 Machine-to-Machine (M2M)	20
2.1.1 M2M là gì?	21
2.1.2 M2M hoạt động như thế nào?	21
2.1.3 Ứng dụng của M2M	22
2.2 MQTT	23
2.2.1 MQTT là gì?	23
2.2.2 MQTT hoạt động như thế nào?	24
2.2.3 Ưu và Nhược điểm của MQTT	27
2.2.4 Ứng dụng của MQTT	28
2.3 OPC UA	30
2.3.1 OPC UA là gì?	30
2.3.2 Điểm khác nhau giữa OPC UA và OPC Classic	31
2.3.3 OPC UA hoạt động như thế nào?	32
2.3.4 Ưu và nhược điểm của OPC UA	35
2.3.5 Ứng dụng của OPC UA	37
2.3.6 OPC UA trong Công nghiệp 4.0	37

Chapter 3. THỰC NGHIỆM SO SÁNH OPC-UA VÀ MQTT 39

3.1	Phương pháp thực nghiệm	39
3.1.1	MQTT	39
3.1.2	OPC-UA	41
3.1.3	Lựa chọn công nghệ hiện thực thực nghiệm	42
3.2	Kết quả thực nghiệm và đánh giá	49
3.2.1	Thực nghiệm MQTT	49
3.2.2	Thực nghiệm OPC-UA	52
3.2.3	So sánh kết quả thực nghiệm MQTT và OPC-UA . . .	53

Chapter 4. ĐIỀU KHIỂN CÁNH TAY ROBOT

DỰA TRÊN OPC-UA	57	
4.1	Cánh tay robot công nghiệp	57
4.1.1	Khái niệm	57
4.1.2	Cấu tạo của cánh tay robot	58
4.1.3	Ứng dụng của cánh tay robot trong công nghiệp . . .	58
4.2	Hiện thực điều khiển cánh tay robot với giao thức OPCUA .	59
4.2.1	Giới thiệu thiết bị	60
4.2.2	Kiến trúc hệ thống	62
4.2.3	Hiện thực module Control	63
4.2.4	Hiện thực module DataCenter	69
4.2.5	Hiện thực Mobile App điều khiển thiết bị	72

Chapter 5. TỔNG KẾT VÀ HƯỚNG PHÁT TRIỂN CHO

ĐỒ ÁN TỐT NGHIỆP	97	
5.1	Kết quả đạt được trong quá trình thực hiện	97
5.1.1	Tìm hiểu và hiện thực giao thức OPC-UA	97
5.1.2	Thực nghiệm và đưa ra những đánh giá nhận xét giữa OPC-UA và MQTT	98
5.1.3	Đề xuất kiến trúc và hiện thực điều khiển cánh tay robot	98
5.2	Hướng phát triển và mở rộng cho luận văn	100
5.2.1	Tích hợp thêm cảm biến cho cánh tay	100
5.2.2	Mô phỏng cánh tay trong 3D (Digital Twin)	100

5.2.3	Gắn thêm camera vận hành các tính năng nhận diện vật phẩm	101
5.2.4	Hoàn thiện DataCenter kết nối đồng bộ dữ liệu nhiều thiết bị	101
	Tham khảo	102

Danh sách hình ảnh

1.1	Các cuộc cách mạng khoa học công nghệ của nhân loại	3
1.2	Kiến trúc chung cho các ứng dụng thời đại 4.0	4
1.3	Các công nghệ của Công nghiệp 4.0	7
1.4	Lợi ích của các công nghệ Công nghiệp 4.0	11
1.5	Mạng cảm biến không dây công nghiệp	16
2.1	Ứng dụng của M2M	22
2.2	Kiến thức chung của MQTT sử dụng MQTT Broker	24
2.3	Ba mức Chất lượng Dịch vụ (QoS)	26
2.4	Cấu trúc của topic	26
2.5	Ứng dụng của MQTT	29
2.6	OPC Classic vs. OPC UA	31
2.7	Năm level trong tổ chức công nghiệp	33
2.8	OPC UA server	34
2.9	OPC UA client	34
2.10	OPC UA and Industry 4.0	38
3.1	Mô hình hệ thống đo đạc RTT sử dụng MQTT	40
3.2	Mô hình hệ thống đo đạc RTT sử dụng OPC-UA	41
3.3	Tải về bản cài đặt của Mosquitto trên trang chủ	43
3.4	Tiến hành cài đặt mosquitto	43
3.5	Vị trí của file mosquitto.conf trong thư mục cài đặt	44
3.6	Chỉnh sửa thông số "allow anonymous"	44
3.7	Chỉnh sửa thông số listener	45
3.8	Tải về packet sender từ trang chủ	47
3.9	Tiến hành cài đặt	47
3.10	Giao diện packet sender	48

3.11	Chọn Intense traffic (tạo tải nặng). Chức năng này sẽ gửi một loạt nhiều gói tin giống nhau tới một địa chỉ và số port cho trước	48
3.12	Diễn IP và số port, nhấn start để bắt đầu tạo tải mạng bằng cách gửi hàng loạt các gói tin. Thông số thống kê được thể hiện trong hình.	49
3.13	Biểu đồ RTT của MQTT trong điều kiện mạng bình thường và quá tải	51
3.14	Biểu đồ RTT của OPCUA trong điều kiện mạng bình thường và quá tải	53
3.15	Biểu đồ RTT trung bình của OPC-UA và MQTT trong điều kiện mạng bình thường	54
3.16	Biểu đồ RTT trung bình của OPC-UA và MQTT trong điều kiện mạng tải cao	54
3.17	Biểu đồ Boxplot của OPC-UA và MQTT trong điều kiện mạng bình thường	55
3.18	Biểu đồ Boxplot của OPC-UA và MQTT trong điều kiện mạng tải cao	55
4.1	Hình ảnh mạch Yolo:bit	60
4.2	Hình ảnh mô phỏng kiến trúc của cánh tay cùng Yolo:bit . . .	61
4.3	Kiến trúc các module giao tiếp qua OPCUA	63
4.4	Cắm chấn Servo	64
4.5	Cáp nguồn mạch mở rộng	65
4.6	Kết nối UART vào chân P14-P15	65
4.7	Cắm chấn Servo	66
4.8	Unity	72
4.9	Trang web cài đặt Unity	74
4.10	Nhấn Nút Install Editor	74
4.11	Chọn phiên bản Unity Editor	75
4.12	Chọn module cho Unity	75
4.13	Chọn vị trí cài đặt Unity Editor	76
4.14	Khi Install thành công, Unity editor sẽ hiện ra như hình . . .	76
4.15	Tạo project Unity mới	77
4.16	Chọn template phù hợp với ứng dụng	77

4.17 Giao diện của Unity	78
4.18 Tạo Canvas để làm giao diện trong Unity	78
4.19 Thay đổi góc camera	79
4.20 Background của ứng dụng	80
4.21 Game Object Login Holder	80
4.22 Game Object Control holder	81
4.23 Game Object BlurBG	82
4.24 Game Object Setting Holder	83
4.25 Thành phần trong .NET Framework	85
4.26 Packet Opc.UaFx.Client	86
4.27 Tải về packet và xem danh sách cách packet liên quan	86
4.28 Danh sách các file .dll đầy đủ để sử dụng thư viện Opc.UaFx.Client	88
4.29 Giao diện ứng dụng khi vừa khởi động, chưa kết nối	95
4.30 Giao diện ứng dụng khi đã kết nối	95
4.31 Giao diện ứng dụng khi bật hộp thoại setting	96
5.1 Kiến trúc đơn giản với 1 Client - 1 Server	98
5.2 Kiến trúc đề xuất	99
5.3 Giao diện app mobile	99

Tables

1.1	Ứng dụng I4.0	7
3.1	thực nghiệm đo RTT của MQTT	50
3.2	thực nghiệm đo RTT của OPCUA	52

CHAPTER 1

GIỚI THIỆU TỔNG QUAN

Chương này cung cấp một cái nhìn tổng quan về nền Công nghiệp 4.0 (Industrie 4.0) - Về kiến trúc của một ứng dụng Công nghiệp 4.0 cũng như các ứng dụng mà các công nghệ trong nền Công nghiệp 4.0 mang lại. Với sự phát triển nhanh chóng của công nghệ điện - điện tử, công nghệ thông tin và công nghệ sản xuất tiên tiến, phương thức sản xuất của các doanh nghiệp sản xuất đang được chuyển từ kỹ thuật số sang thông minh. Và Nhà máy thông minh (Smart Factory) là một giải pháp sản xuất linh hoạt và hiệu quả để đáp ứng được nhu cầu của thị trường ngày nay. Những thuận lợi và thách thức của Nhà máy thông minh cũng sẽ được phân tích trong chương này để chúng ta có thể thấy được tầm quan trọng của nó cũng như những khuyết điểm cần được nghiên cứu và khắc phục.

1.1 Tổng quan về Công nghiệp 4.0

Quá trình công nghiệp hóa bắt đầu với sự ra đời của các thiết bị sản xuất cơ khí vào cuối thế kỷ 18. Sự phát triển động cơ hơi nước của James Watt đã dẫn đến cách mạng hóa cách sản xuất sản phẩm bằng máy móc và động cơ. Bên cạnh đó, xã hội nông nghiệp cũng chuyển thành xã hội công nghiệp. Cho đến nay, bốn cuộc cách mạng công nghiệp đã xảy ra như Hình 1.1. Cuộc cách mạng công nghiệp đầu tiên đã tạo ra những cải tiến vượt bậc nhờ cơ giới hóa, năng lượng hơi nước và khung dệt. Sự chuyển đổi này được sau bởi cuộc cách mạng công nghiệp lần thứ hai vào đầu thế kỷ 20 và lần chuyển đổi này liên quan đến sản xuất hàng loạt, dây chuyền lắp ráp và năng lượng điện. Cuộc cách mạng này ngụ ý những thay đổi về tổ chức, chẳng hạn như

việc triển khai dây chuyền lắp ráp của Henry Ford và các quy trình quản lý khoa học dựa trên *Frederic W. Taylor*, hay còn được gọi là *Chủ nghĩa Taylor*. Trong cuộc cách mạng này, sản xuất hàng loạt như sản xuất công nghiệp quy mô lớn tăng lên. Do đó, ngành hóa chất và điện tử, cũng như ngành cơ khí và ô tô đã tận dụng những cuộc cách mạng này và bắt đầu phát triển.

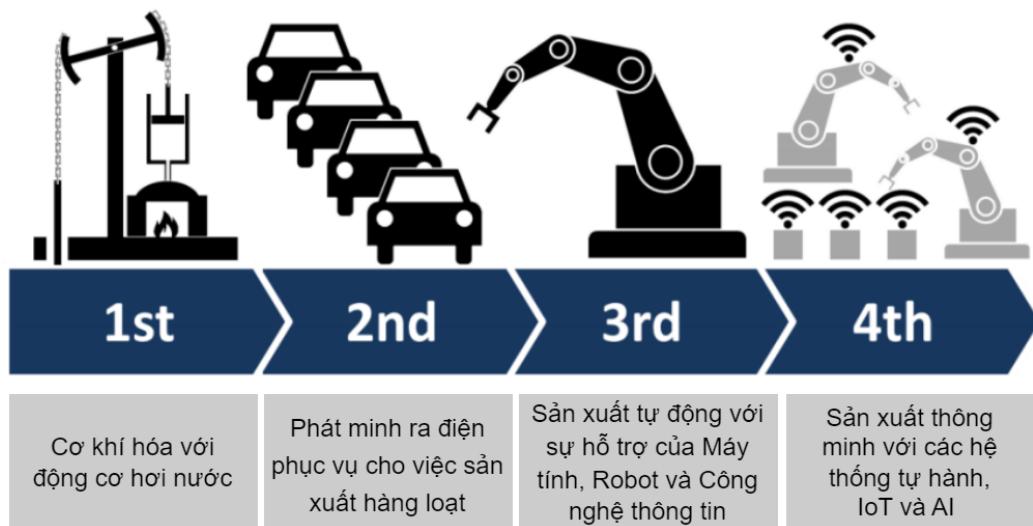


Figure 1.1: Các cuộc cách mạng khoa học công nghệ của nhân loại

Vào đầu những năm 1970, cuộc cách mạng công nghiệp lần thứ ba đã xuất hiện và nó vẫn tiếp tục cho đến ngày nay. Ứng dụng điện tử và công nghệ thông tin là những đặc điểm nổi bật nhất của cuộc cách mạng này giúp tăng cường tự động hóa các quy trình sản xuất, cũng như thay thế máy móc thay cho người lao động. Do đó, nó tạo ra nhiều hiệu quả về kinh tế-xã hội và văn hóa-xã hội. Hơn nữa, sản xuất hàng loạt linh hoạt đã tăng năng suất của các quy trình sản xuất. Ngày nay, cuộc cách mạng lần thứ ba vẫn hiện diện nhưng đang chuyển mình uyển chuyển sang một thời đại công nghiệp hóa mới hay còn gọi là cuộc cách mạng công nghiệp lần thứ tư (Industry 4.0). Cuộc cách mạng công nghiệp lần thứ tư đặc trưng bởi việc đưa Internet vạn vật (IoT) vào sản xuất, cho phép các nhà máy thông minh có hệ thống sản xuất tích hợp theo chiều dọc và chiều ngang. Đó là một sự chuyển đổi lớn của toàn bộ ngành sản xuất công nghiệp bằng cách kết hợp các công nghệ kỹ thuật số và internet với ngành công nghiệp truyền thống.

Internet vạn vật, hay còn gọi là Internet of Things – IoTs, là một cuộc cách mạng trong việc kết nối giữa các thiết bị không dây với nhau. Ban đầu, chúng ta có mạng Internet, một thành tựu của cuộc cách mạng khoa

học công nghệ lần thứ 3, cho phép các máy tính có thể kết nối và trao đổi thông tin toàn cầu. Tuy nhiên, với sự phát triển nhanh chóng của ngành vi cơ điện tử (Micro Electro Mechanical System), không chỉ máy tính, giờ đây rất nhiều các thiết bị có khả năng kết nối vào mạng Internet. Thông dụng nhất trong cuộc sống mà chúng ta có thể kể đến như các điện thoại thông minh, máy tính bảng, các loại thẻ thông minh (Smart cards) hay như các nốt trong mạng cảm biến không dây (Wireless Sensor Networks). Với những đặc tính đó, một thế hệ mạng mới đã được hình thành, và là sản phẩm đặc trưng cho cuộc cách mạng khoa học công nghệ lần thứ 4, mạng Internet vạn vật, hay còn gọi là IoT - Internet of Things.

Dựa trên mạng Internet vạn vật, các ứng dụng không còn ở khái niệm thông minh nữa, mà sẽ tiến lên một bước cao hơn, gọi là tự hành (autonomous), chẳng hạn như các ứng dụng giám sát và tự động thích nghi trong việc điều khiển như các dịch vụ trong nhà, bãi giữ xe, hay các hệ thống quan trắc trong nông nghiệp, thủy hải sản. Theo diễn giả nổi tiếng Timothy Chou, kiến trúc của các ứng dụng thời đại cách mạng công nghiệp 4.0 dựa trên Internet vạn vật nói chung và sản xuất thông minh trong công nghiệp nói riêng, được chia thành mô hình 5 lớp, như mô tả ở hình bên dưới.

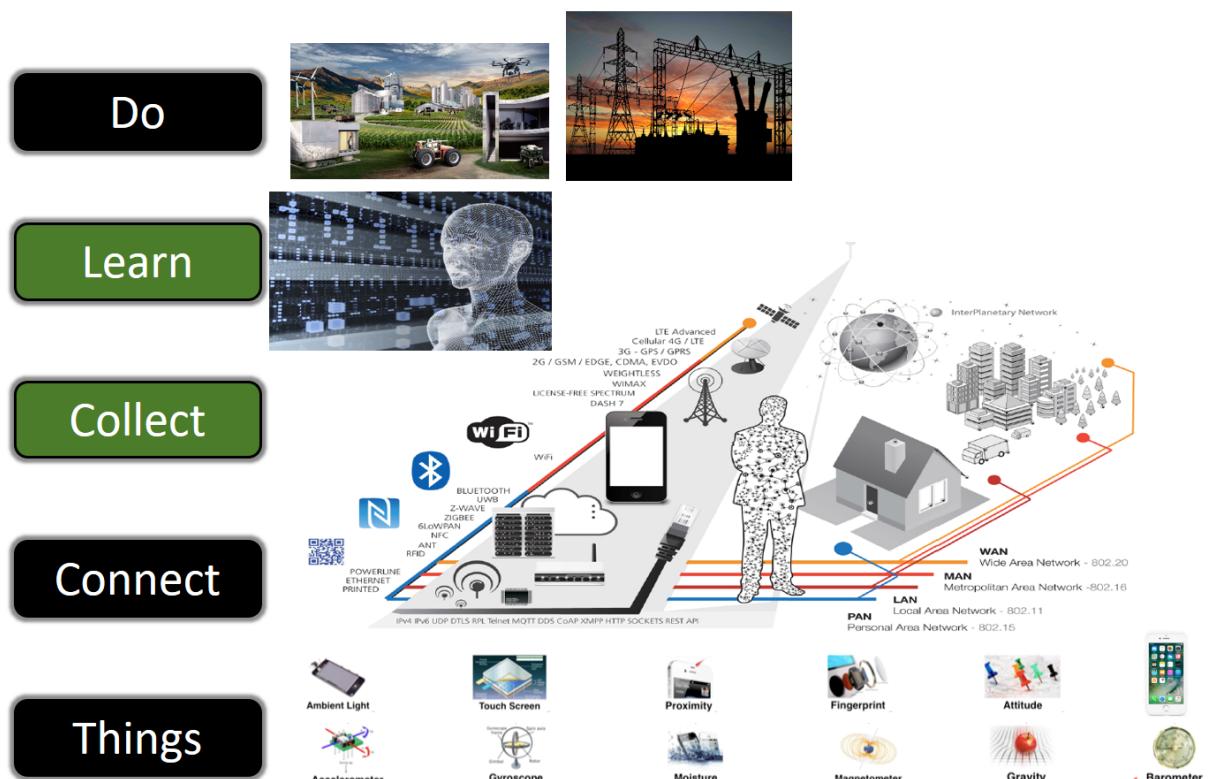


Figure 1.2: Kiến trúc chung cho các ứng dụng thời đại 4.0

Lớp đầu tiên bao gồm các Things. Things ở đây chính các máy móc và chúng được kết nối với Internet theo nhiều cách khác nhau. Sau khi được kết nối, Lớp Collect ở đây đề cập đến các công nghệ được thiết kế để thu thập dữ liệu - dữ liệu chuỗi thời gian được gửi đi mỗi giờ, phút hoặc giây. Lớp thứ tư là Learn. Không giống như trong thế giới của các ứng dụng IoP (Internet of People), nơi chúng ta phải nhập những nội dung input nào đó, thì các ứng dụng IoT sẽ lấy dữ liệu một cách liên tục. Ví dụ, chúng ta có thể sử dụng máy móc để học hỏi (learn) từ những Things của mình tại bệnh viện, hầm mỏ hoặc trang trại. Và cuối cùng, chúng ta sẽ có những câu hỏi, Những công nghệ này để làm gì? Kết quả kinh doanh là gì? Thì khi đó, Lớp Do mô tả cả công nghệ ứng dụng phần mềm và mô hình kinh doanh bị ảnh hưởng bởi các công ty sản xuất ra Things, cũng như những người sử dụng chúng để cung cấp dịch vụ chăm sóc sức khỏe, vận chuyển hoặc xây dựng chẳng hạn. Và chức năng chính của từng lớp trong kiến trúc này được khái quát như sau:

- **Things:** Các thiết bị trong ứng dụng giám sát. Chúng ta có thể thấy, đây là lớp rất phong phú về mặt số lượng và đa dạng về chức năng. Rất nhiều các loại cảm biến sẽ được dùng, tùy vào các ứng dụng giám sát. Bên cạnh đó, các nốt cảm biến sẽ chủ yếu dựa vào giao tiếp không dây.
- **Connect:** Thu thập dữ liệu từ các nốt cảm biến. Do có rất nhiều tiêu chuẩn kết nối tùy theo từng loại ứng dụng, lớp này phải hỗ trợ nhiều loại kết nối, từ giao tiếp Zigbee và Wifi trong các ứng dụng nhà thông minh, với khoảng cách giao tiếp ngắn cho đến các giao trên không gian rộng như LoRa hay 3G/4G.
- **Collect:** Sau khi dữ liệu được thu thập, chúng sẽ được gửi lên các server tập trung để lưu trữ dữ liệu. Tại đây, một lượng lớn dữ liệu sẽ được đẩy về, tạo ra một thách thức không nhỏ cho các server và phải ứng dụng các công nghệ về Big Data (dữ liệu lớn) để xử lý.
- **Learn:** Nhiệm vụ của lớp này là lọc ra các thông tin đặc trưng, có ngữ nghĩa đặc thù cho từng loại ứng dụng. Các công nghệ về Học Máy và hiện tại là Học Sâu (Deep Learning) sẽ được áp dụng ở đây.
- **Do:** Dựa vào các thông tin đặc trưng, hệ thống sẽ xây dựng nên những quy luật thích nghi theo ngoại cảnh, và đề xuất các quyết định cho hệ thống. Với mỗi quyết định, việc thực thi sẽ được đo đạc một cách tự

động, và sai lệnh của quyết định đó so với mục tiêu tối ưu sẽ được xem xét lại cho lần sau. Theo cách này, hệ thống sẽ tự động tích lũy “kinh nghiệm” trong một thời gian dài, để ngày càng trở nên thông minh và hoàn thiện hơn.

Sự phát triển mạnh mẽ của khoa học công nghệ ở những năm đầu thế kỷ 21 đã đưa nhân loại bước vào cách mạng công nghiệp 4.0, khi mà "hàng tỉ" thiết bị có thể giao tiếp và chia sẻ dữ liệu với nhau thông qua mạng kết nối vạn vật (Internet of Things). Theo ước tính của cộng đồng khoa học, đến năm 2025 sẽ có 75 tỉ thiết bị có thể kết nối mạng Internet với nhau. Và với sự bùng nổ về số lượng thiết bị kết nối đó, vô tình mang lại một trọng trách lớn đối với Lớp Connect - trong đó, các giao thức truyền thông đóng một vai trò đặc biệt quan trọng để có thể giúp các thiết bị kết nối và giao tiếp với nhau một cách hiệu quả.

1.2 Các ứng dụng công nghệ Công nghiệp 4.0

Ứng dụng công nghệ Công nghiệp 4.0 bao gồm các công nghệ khác nhau như Internet vạn vật (IoT), điện toán đám mây, additive manufacturing, an ninh mạng với blockchain, augmented reality với trí tuệ nhân tạo (AI), big data, tích hợp hệ thống (system integration), mô phỏng (simulation) và robot tự hành. Nhiều công nghệ trong số này đã được sử dụng trong sản xuất, nhưng với ngành công nghiệp 4.0, chúng sẽ chuyển đổi thành quy trình sản xuất dưới dạng các cell được tối ưu hóa sẽ kết hợp với nhau thành một quy trình sản xuất được tích hợp hoàn toàn, tự động hóa và tối ưu hóa. Chúng làm tăng hiệu quả và thay đổi hệ thống chuỗi cung ứng truyền thống, cũng như mối quan hệ giữa con người và máy móc. Các kỹ thuật của Công nghiệp 4.0 có khả năng cải thiện việc sử dụng năng lượng, thiết bị và nguồn nhân lực. Công nghiệp 4.0 là một cấu trúc tương lai nuôi dưỡng sự phát triển của các hệ thống sản xuất tự trị với ứng dụng IoT, CPS và AI. Các công nghệ dựa trên cảm biến mới giúp các doanh nghiệp vừa và nhỏ theo dõi liên tục việc sử dụng máy móc, nhu cầu năng lượng và đào tạo nhân viên. Bằng cách phân tích kỹ lưỡng các công nghệ Công nghiệp 4.0 khác nhau, dữ liệu từ các

thiết bị IoT khác nhau có thể được xử lý để cải thiện tính bền vững của hoạt động sản xuất.

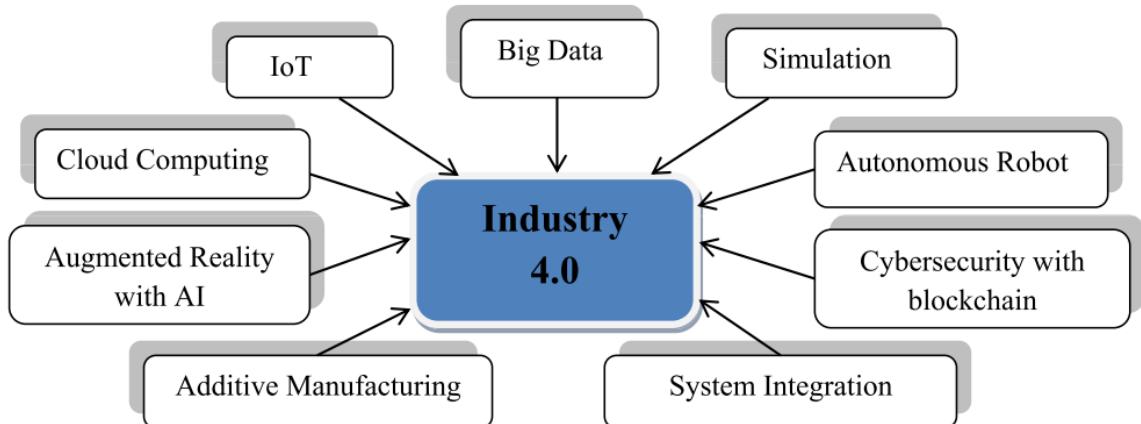


Figure 1.3: Các công nghệ của Công nghiệp 4.0

Các công nghệ này có thể đóng góp đáng kể cho sự bền vững bằng cách giảm lượng khí thải carbon, sử dụng năng lượng tái tạo và các giải pháp công nghệ phù hợp cho cả cá nhân và xã hội. Sự phát triển của Công nghiệp 4.0 giúp sử dụng tối ưu các nguồn tài nguyên theo cách minh bạch hơn. Bằng cách thực hiện các phương pháp của Công nghiệp 4.0, hiệu quả sản xuất và đổi mới có thể được cải thiện, ảnh hưởng đến tính bền vững về xã hội và môi trường. Có nhiều lợi ích được đóng góp riêng bởi các công nghệ khác nhau của Công nghiệp 4.0. Các ứng dụng quan trọng của các công nghệ Công nghiệp 4.0 này được tóm tắt trong Bảng 1.1

Table 1.1: Ứng dụng của các công nghệ Công nghiệp 4.0

Các công nghệ của Công nghiệp 4.0	Ứng dụng
IoTs	IoT tích hợp các cảm biến vào hệ thống sản xuất. Các cảm biến và máy móc của nhà sản xuất được kết nối mạng và sử dụng điện toán nhúng. Nó tạo ra khả năng thu thập và phân tích dữ liệu theo cách phi tập trung.

Continued on next page

Table 1.1: Ứng dụng của các công nghệ Công nghiệp 4.0 (Continued)

Diện toán đám mây (Cloud computing)	Truy cập dữ liệu có sẵn mọi lúc mọi nơi; tính minh bạch và khả năng đáp ứng của chuỗi cung ứng; dễ dàng chia sẻ dữ liệu quan trọng; hỗ trợ từ các đối tác trong chuỗi cung ứng trong việc nâng cấp công nghệ, dữ liệu về vòng đời sản phẩm có thể được lưu trữ và truy xuất theo triết lý CE.
Augmented Reality	Các hệ thống dựa trên Augmented Reality hỗ trợ nhiều dịch vụ khác nhau, chẳng hạn như chọn các bộ phận trong kho và gửi hướng dẫn sửa chữa qua thiết bị di động. Nó cung cấp thông tin theo thời gian thực để giúp các nhà sản xuất đưa ra quyết định theo thời gian thực và cải thiện quy trình làm việc.
Sản xuất phụ gia (Additive manufacturing - AM)	Sản xuất phụ gia hoặc in 3D được sử dụng để sản xuất các lô sản xuất nhỏ theo yêu cầu. Nó tạo ra những lợi thế như là: sản xuất các sản phẩm phức tạp và cấu trúc nhẹ. Hệ thống AM phi tập trung và hiệu suất cao này sẽ giảm khoảng cách vận chuyển và lượng hàng tồn kho. Do đó, nó sửa đổi hệ thống chuỗi cung ứng.

Continued on next page

Table 1.1: Ứng dụng của các công nghệ Công nghiệp 4.0 (Continued)

Tích hợp hệ thống (System integration)	Trong ngành công nghiệp 4.0, các công ty, bộ phận, phòng ban, chức năng sẽ trở nên gắn kết hơn nhiều, với tư cách là cross-company. Do đó, cross-company này, cũng như các mạng tích hợp dữ liệu toàn cầu phát triển và cho phép các chuỗi giá trị thực sự tự động. Do đó, nó tạo ra sự kết nối trong chuỗi cung ứng, giữa nhà cung cấp và khách hàng thông qua tích hợp theo chiều ngang và dọc.
An ninh mạng (Cyber-security)	Công nghiệp 4.0 làm tăng khả năng kết nối và sử dụng các giao thức truyền thông tiêu chuẩn. Khi đó, cần phải bảo vệ các hệ thống thông tin, hệ thống công nghiệp, dây chuyền sản xuất và thiết bị khỏi các mối đe dọa an ninh mạng với tần suất ngày càng tăng cao. Điều cần thiết là tạo ra thông tin liên lạc an toàn, đáng tin cậy, cũng như quản lý truy cập và nhận dạng tinh vi của máy móc và người dùng.
Robot tự hành	Ngày nay, robot được sử dụng để giải quyết các nhiệm vụ phức tạp và cộng tác với nhau và với con người. Những robot này tự chủ, linh hoạt và hợp tác. Hơn nữa, chúng có chi phí thấp hơn và có nhiều khả năng hơn so trước đây.

Continued on next page

Table 1.1: Ứng dụng của các công nghệ Công nghiệp 4.0 (Continued)

Mô phỏng (Simulation)	Mô phỏng 2D hoặc 3D của quá trình phát triển sản phẩm, phát triển vật liệu và sản xuất đã được sử dụng trong thế giới công nghiệp. Ngày nay, mô phỏng phải được sử dụng rộng rãi hơn trong các hoạt động của nhà máy. Những mô phỏng này cung cấp dữ liệu thời gian thực để phản ánh thế giới thực trong một mô hình ảo, bao gồm máy móc, sản phẩm và con người. Nó giúp làm giảm thời gian thiết lập máy và tăng chất lượng sản xuất.
Phân tích dữ liệu lớn (Big Data Analytics)	Dữ liệu được thu thập từ nhiều thiết bị dựa trên IoT có thể được phân tích để lấy thông tin & xu hướng; dữ liệu có thể được sử dụng để lập trình các thiết bị AI; việc sử dụng máy móc và nguồn nhân lực có thể được tối ưu hóa; khả năng truy xuất nguồn gốc của sản phẩm sẽ cải thiện.

Công nghiệp 4.0 và 9 công nghệ chủ chốt của nó mang lại nhiều lợi ích cho thế giới công nghiệp

- **Năng suất:** Con người và máy móc có thể thiết lập mối quan hệ làm việc thông minh, do đó cho phép các doanh nghiệp tăng năng lực sản xuất, giảm sai sót của con người và cung cấp khả năng tùy chỉnh hàng loạt để đáp ứng nhu cầu đa dạng trong thời gian ngắn.
- **Tính linh hoạt:** Tính linh hoạt được cải thiện giúp tổ chức thay thế một sản phẩm hiện có bằng một sản phẩm dựa trên khách hàng và tăng tốc độ đổi mới sản phẩm.
- **Đổi mới:** Tüm nhìn từ nguồn cấp dữ liệu IoT tại các sản phẩm và thiết bị thông minh để cho phép hiểu rõ hơn về việc làm cái gì cho cả thiết kế sản phẩm và quy trình.
- **Trải nghiệm khách hàng:** Dữ liệu từ Hệ thống điều hành sản xuất

(MES) có thể là cơ sở để giải quyết ngay các vấn đề giữa khách hàng và nhà sản xuất.

- **Chi phí:** Trong khi ngành công nghiệp 4.0 yêu cầu đầu tư ban đầu, một khi trí thông minh được tích hợp vào các sản phẩm và quy trình, chi phí sẽ giảm mạnh. Ít vấn đề về chất lượng hơn dẫn đến ít lãng phí vật liệu hơn, ít nhân viên hơn và chi phí vận hành thấp hơn.
- **Doanh thu:** Với chất lượng tốt hơn, chi phí thấp hơn, tỷ lệ chất lượng trên giá cao hơn và khả năng phục vụ khách hàng tốt hơn, ngành công nghiệp 4.0 đặt các nhà sản xuất vào con đường trở thành nhà cung cấp ưa thích của khách hàng hiện tại đồng thời mở ra các thị trường lớn hơn.



Figure 1.4: Lợi ích của các công nghệ Công nghiệp 4.0

Với sự phát triển nhanh chóng của những công nghệ của nền Công nghiệp 4.0 đã được nêu ở trên và những ứng dụng mà nó mang lại, phương thức sản xuất của các doanh nghiệp sản xuất đang được chuyển từ kỹ thuật số sang thông minh (digital to intelligent). Kỷ nguyên mới của sự kết hợp công nghệ thực tế ảo dựa trên Hệ thống vật lý mạng (Cyber-Physical System) đang đến. Trước những thách thức mới, lợi thế của các ngành sản xuất truyền thống đang dần bị mai một. Do đó, công nghệ sản xuất thông minh (intelligent manufacturing technology) là một trong những lĩnh vực công nghệ cao được các nước công nghiệp phát triển đặc biệt chú trọng. Chiến lược Châu Âu 2020 (Europe 2020 strategy), Chiến lược Công nghiệp 4.0 (Industry 4.0 strategy) và Sản xuất Trung Quốc 2025 (China manufacturing 2025) đã được đề xuất. Hoa Kỳ đã từng bước đẩy nhanh tốc độ tái công nghiệp hóa và tái sản xuất. Sự chuyển đổi của sản xuất thông minh đã gây tò mò về tác động sâu sắc và lâu dài của nó đối với ngành sản xuất trong tương lai trên toàn thế giới.

Đặc biệt, trong bối cảnh sản xuất thông minh, các nhà máy đã trở nên linh hoạt hơn bao giờ hết trước sự hỗn loạn phức tạp của thị trường hiện đại. Các khái niệm hiện đại về hệ thống sản xuất đòi hỏi sự tích hợp theo chiều dọc và chiều ngang của tất cả những người tham gia vào quá trình sản xuất. Do đó, điều quan trọng nhất là thiết lập nhà máy thông minh (smart factory) để có thể đạt được sản xuất tiên tiến dựa trên công nghệ mạng (network technologie) và dữ liệu sản xuất (manufacturing data). Nhà máy thông minh là một giải pháp sản xuất linh hoạt và hiệu quả để đáp ứng nhu cầu của thị trường ngày nay, đồng thời nó đạt được sự tích hợp giữa các đối tác công nghiệp và phi công nghiệp khác nhau, những người xây dựng các tổ chức năng động và các tổ chức ảo.

1.3 Smart factory

Dưới sự ảnh hưởng cuộc cách mạng công nghiệp 4.0 và sự ra đời, phát triển của vô số các công nghệ mới như IoT, IIoT, Cloud computing, Cyber-security, Big Data Analytics... như trình bày ở trên, đã dẫn đến sự thay đổi lớn cho xu hướng công nghiệp thời đại mới, chú trọng hơn vào dữ liệu, về sự kết nối và tính thông minh, các mô hình sản xuất thuộc nhiều lĩnh vực khác nhau đã có những bước thay đổi, phát triển để phù hợp hơn, mà trong đó không thể kể đến là mô hình nhà máy thông minh (Smart Factory). Smart Factory là một mô hình sản xuất công nghiệp đánh dấu bước trưởng thành đột phá của sản xuất trong thời đại công nghiệp 4.0 so với thời kì 3.0 trước đó, đem lại định nghĩa mới về sản xuất tại nhà máy không chỉ đơn thuần là "tự động", mà còn phải "tự hành" và "thông minh". Để có thể xây dựng một mô hình Smart Factory, ta có thể áp dụng và kết hợp các công nghệ mới của thời đại 4.0 trên, và đảm bảo chúng cần được liên kết với nhau để tận dụng tối đa sức mạnh mà các công nghệ mang lại. Trong phạm vi đề tài, nhóm chỉ giới hạn tìm hiểu về ứng dụng của công nghệ IoT, cụ thể là IIoT (Industrial Internet of Things) trong bối cảnh nhà máy thông minh.

Phần tiếp theo, nhóm sẽ trình bày ba ý chính của smart factory: Khái niệm và lợi ích của smart factory, kiến trúc đề xuất của nhóm cho các ứng dụng Smart Factory dựa trên mô hình năm lớp của diễn giả Timothy Chou, và cuối cùng là những thách thức của ứng dụng smart factory trong lớp

Connect - Một trong năm lớp của kiến trúc ứng dụng thời đại 4.0 thuộc phạm vi nghiên cứu của đồ án, qua đó nêu ra được những giải pháp đã và đang được áp dụng để xử lý vấn đề đó trong môi trường công nghiệp ngày nay.

1.3.1 Định nghĩa và lợi ích của Smart Factory

Smart Factory (Nhà máy thông minh) là một cơ sở sản xuất được "số hóa" sử dụng các thiết bị, máy móc và hệ thống sản xuất kết nối với nhau nhằm liên tục thu thập thông tin và chia sẻ dữ liệu. Dữ liệu này được sử dụng để tạo ra những dự đoán nhằm cải thiện quá trình sản xuất và chỉ ra các vấn đề đang gặp phải trong tiến trình này. Đọc định nghĩa trên, ta có thể chỉ ra hai ý chính làm nên một Smart Factory, đó chính là "sự kết nối của tất cả thiết bị" và "dự đoán". Thật vậy, nói đến smart factory, không phải ý chỉ là nhà máy chỉ trang bị những máy móc hiện đại và tự động là đủ, bởi những cỗ máy "tự động" này đã xuất hiện từ lâu ở thời đại công nghiệp 3.0. Sự khác biệt giữa Smart factory trong thời đại công nghiệp 4.0 này chính là việc các máy móc thay vì độc lập với nhau, chúng sẽ thông minh hơn, biết kết nối, giao tiếp và chia sẻ thông tin, tạo thành một mạng lưới thông tin khổng lồ giữa các thiết bị trong nhà máy. Hơn nữa, với lượng dữ liệu khổng lồ như vậy, sẽ là cơ hội để ta thu thập và phân tích chúng - sử dụng trí tuệ nhân tạo để có thể thấy rõ những đặc trưng của sản xuất một cách thực tế, đồng thời giúp ta dự đoán những sự kiện mới, thậm chí là tạo ra những cải tiến cho mô hình sản xuất và truyền lại để các máy móc áp dụng. Smart Factory đã đề ra một nhà máy "kết nối" và "thông minh" hơn so với sự tự động hóa đơn thuần của các nhà máy thời đại công nghiệp 3.0

Khi áp dụng mô hình Smart Factory, doanh nghiệp sẽ nhận được nhiều lợi ích và ưu thế, đồng thời cải thiện hiệu năng sản xuất của mình. Một số lợi ích từ Smart Factory được liệt kê như sau:

- **Tăng cường năng suất và hiệu quả:** Trong suốt lịch sử, sự thành bại của sản xuất công nghiệp là từ sự phản ứng nhanh nhạy trước những sự kiện hay xu hướng đang xảy ra và cố gắng để dịch chuyển phương hướng sản xuất theo xu hướng đó. Nhưng với Smart Factory, ta có thể giảm bớt cơ chế phản ứng lỗi thời, tạo ra khả năng quản lý nguồn cung một cách linh hoạt và hợp lý hơn. Thật vậy, nhờ vào khả năng dự đoán

phân tích (predictive analytics) và dữ liệu lớn (Big Data) khi áp dụng vào smart factory, giúp nhà máy có thể tìm ra được phương hướng sản xuất tối ưu và phù hợp với xu hướng hiện tại. Quản lý tài nguyên kịp thời (Just-in-time), dự đoán trước nhu cầu chính xác và tăng cường tốc độ sản xuất ra thị trường là các lợi ích có được từ smart factory, giúp tiết kiệm chi phí đồng thời tăng khả năng đáp ứng nhanh của doanh nghiệp trước sự thay đổi chóng mặt của thị trường thời đại mới. Chưa hết, nhờ vào việc thu thập dữ liệu của các máy móc và sự giao tiếp giữa chúng trong một mạng nhà máy thông minh, doanh nghiệp có thể phân tích và dự đoán trạng thái của máy móc, qua đó kịp thời sửa chữa khi cần, giảm thời gian chết trong sản xuất và cải thiện năng suất nhà máy.

- **Sự ổn định và an toàn:** Việc áp dụng smart factory sẽ giúp doanh nghiệp tìm ra được những phương pháp sản xuất "xanh", an toàn và có trách nhiệm với môi trường. Đồng thời, do dây chuyền sản xuất được tự động hóa một cách thông minh sẽ giúp tránh được lỗi sai của sản phẩm do con người, giúp tăng tính ổn định của sản phẩm và đảm bảo được chất lượng. Một sản phẩm được sản xuất từ việc áp dụng smart factory sẽ giúp khách hàng yên tâm sử dụng hơn và họ sẽ chấp nhận trả nhiều tiền hơn cho sản phẩm đó, góp phần vào việc tăng dần doanh thu của nhà máy.

1.3.2 Kiến trúc của Smart Factory

Trong thời đại công nghiệp 4.0, sản xuất thông minh (intelligent manufacturing) thu hút nhiều sự quan tâm từ nhà nước, các chuyên gia và các nhà nghiên cứu khoa học. Theo đó, sự xây dựng nền kiến trúc của Smart Factory cũng được nghiên cứu. Smart Factory là dựa trên phương hướng xây dựng nhà máy "số hóa" (digital) và "tự động" (automated), sử dụng công nghệ thông tin như nền tảng đám mây hoặc IIoT để cải thiện khả năng quản lý tài nguyên sản xuất và chất lượng dịch vụ. [1] Hiện tại, vẫn chưa có một hiện thực nào của Smart factory được chuẩn hóa và công bố, tuy nhiên, dựa theo nhiều nghiên cứu khác nhau, Chen et al. [1] cho rằng có ba lớp chính trong kiến trúc của Smart factory:

- **Lớp vật lý:** gồm các thiết bị vật lý cần sự hỗ trợ để lấy được dữ liệu

nhanh chóng trong thời gian thực, các thiết bị giao tiếp phải cung cấp khả năng truyền nhanh chóng, kể cả loại dữ liệu phức tạp.

- **Lớp mạng:** Lớp này quy định cách thức kết nối giữa các thiết bị vật lý và thu thập chúng tới các server edge để xử lý. Về phương thức kết nối, smart factory sử dụng Industrial Internet of Thing - IIoT. Với việc áp dụng IIoT, Các phương pháp để giao tiếp giữa các thiết bị trong nhà máy như Mạng cảm biến công nghiệp (Industrial Wireless Sensor Networks - IWSNs) cần được nghiên cứu.
- **Lớp dữ liệu** Dữ liệu thu thập được từ các thiết bị cần được tập hợp ở nền tảng đám mây, sau đó có khả năng phân tích tìm ra ngữ nghĩa của nhiều loại dữ liệu khác nhau, đặc biệt hơn hết là tạo ra khả năng tự tổ chức, tự học và tự áp dụng ở các thiết bị thông minh này nhằm cải thiện năng suất nhà máy. Đây là lớp áp dụng các công nghệ phân tích, trí tuệ nhân tạo nhằm tận dụng lượng dữ liệu khổng lồ thu thập được từ các thiết bị để đưa ra những dự đoán và tối ưu cho công tác sản xuất của smart factory

Có thể thấy, ba lớp kiến trúc được trình bày trên có những điểm tương đồng nhất định với kiến trúc năm lớp cho các ứng dụng thời đại 4.0 được đề xuất bởi diễn giả Timothy Chou: Lớp vật lý gồm "Things", lớp mạng gồm "Connect" và "Collect", và lớp dữ liệu bao gồm "Learn" và "Do". Mô hình năm lớp của diễn giả Chou có sự phân tách sâu hơn về chức năng của một ứng dụng 4.0, do đó sẽ được nhóm tập trung phân tích. Trong phạm vi của đề tài, nhóm sẽ chú trọng và lớp kết nối (Connect) trong ứng dụng thời đại 4.0 nói chung và trong kiến trúc ứng dụng smart factory nói riêng, nhằm có cái nhìn cụ thể hơn về các thức kết nối các thiết bị thông minh trong smart factory.

Ở phần tiếp theo, nhóm sẽ tập trung vào lớp kết nối này, nhằm nhận những thách thức trong việc hiện thực lớp kết nối cho ứng dụng smart factory và đề ra những giải pháp đang được áp dụng hiện nay để giải quyết những thách thức đó

1.3.3 Thách thức của lớp kết nối

Trong lớp kết nối này là lớp tập trung vào vấn đề kết nối giữa các thiết bị với nhau. Trong nhà máy, công nghệ kết nối thường dùng có thể là dây bus (field bus) hoặc mạng cảm biến (sensor network). Do sự phát triển nhanh chóng của điện toán đám mây, để truyền tải dữ liệu, nhà máy thông minh yêu cầu một công nghệ mạng có thể truyền tải thời gian thực (real-time) và đáng tin cậy, nhằm chia sẻ dữ liệu giữa các thiết bị thông minh với nhau và nền tảng đám mây. Ta có thể sử dụng các dây truyền dẫn trực tiếp nhằm đảm bảo về tốc độ truyền và sự toàn vẹn dữ liệu, tuy nhiên áp dụng nó trong tình huống nhà máy lớn với vô số các cảm biến và thiết bị sẽ tăng độ phức tạp, cản trở sự mở rộng và linh hoạt ứng biến của nhà máy. Do đó, lớp mạng cần được thiết kế để đảm bảo sự linh hoạt, tiện lợi, dễ sử dụng. Mạng cảm biến không dây công nghiệp (Industrial Wireless Sensor Networks - IWSNs) đang được áp dụng nhiều cho các nhà máy thông minh hiện nay để đảm bảo tính kết nối linh hoạt của nhà máy, tuy nhiên chính từ phương thức xây dựng lớp kết nối này cũng tạo ra nhiều thách thức không nhỏ.



Figure 1.5: Mạng cảm biến không dây công nghiệp

Một số thách thức từ việc xây dựng mạng cảm biến không dây công nghiệp cho lớp kết nối có thể được kể đến như:

- Mạng cảm biến không dây công nghiệp xuất hiện và phổ biến là nhờ sự

phát triển và mở rộng vượt bậc của các công nghệ giao tiếp không dây trong thời đại hiện nay, và các công nghệ dành riêng cho công nghiệp. Việc triển khai mạng cho công nghiệp đã dần trở nên linh hoạt, và ít chi phí hơn, tuy nhiên, vì ứng dụng mạng trong công nghiệp vẫn rất phức tạp, nên hiện chưa có một chuẩn giao tiếp không dây tổng quát nào.

- Yêu cầu của mạng không dây trong công nghiệp trong bối cảnh nhà máy thông minh bắt buộc phải có độ trễ thấp, độ tin cậy cao, độ đồng bộ chính xác khi xử lý các dịch vụ điều khiển, khả năng chịu tải truy cập tốt và tiêu tốn năng lượng thấp khi thu thập dữ liệu. Trong đó, độ tin cậy và độ trễ của dữ liệu là yêu cầu quan trọng nhất trong việc xây dựng mạng không dây cho nhà máy thông minh, bởi dữ liệu là nền tảng cho công việc sản xuất và vận hành của nhà máy. Đôi khi, truyền tải dữ liệu bị mất đồng bộ với xung clock có thể gây mất gói, trì hoãn.
- Yếu tố đáng quan tâm nữa dành cho mạng không dây công nghiệp là bảo mật. Vì mạng không dây sử dụng các sóng radio, nên chúng dễ dàng bị xâm phạm bởi cả những người dùng không được ủy quyền, dễ tổn hại hơn so với hệ thống mạng dùng dây. Những cuộc tấn công mạng không dây vào nhà máy có thể gây ra tổn hại nghiêm trọng tới khả năng sản xuất của nhà máy. Hơn nữa, mạng không dây cũng không thể áp dụng các giải thuật bảo mật đang được dùng rộng rãi bên ngoài do chịu giới hạn về vấn đề năng lượng và sức mạnh tính toán hay tài nguyên mạng. Việc xây dựng một cơ chế bảo mật cho mạng không dây công nghiệp vẫn là một vấn đề nan giải cho đến ngày nay.

Một số công nghệ đã được sử dụng để giải quyết vấn đề trong lớp kết nối sử dụng mạng không dây tại nhà máy, các chuẩn không dây như NB-IoT, 5G, LTE, 3GPP cung cấp khả năng giao tiếp không dây với độ trễ thấp và độ tin cậy cao. Giao thức OPC-UA - một giao thức dạng M2M (Machine to Machine) cũng được dùng nhiều trong môi trường công nghiệp, như một giải pháp để truyền tải dữ liệu có cấu trúc với tốc độ cao và đáng tin cậy, được phát triển theo hướng tập trung vào dịch vụ (service oriented). OPC-UA là một giao thức đa nền tảng, phù hợp cho nhiều loại hệ điều hành, cộng với khả năng bảo mật tốt, cho phép giao tiếp hiệu quả giữa các khối thông minh với nhau trong môi trường nhà máy.

Trong đề tài này, nhóm sẽ chủ yếu tập trung vào giao thức được dùng rộng rãi trong ngành công nghiệp hiện nay, và được hỗ trợ bởi đa số các thiết bị phần cứng hiện đại trong nhà máy là giao thức OPC-UA, một giao thức chuẩn công nghiệp theo hướng dịch vụ, được phát triển với mong đợi giải quyết các vấn đề hóc búa về giao tiếp giữa các thiết bị thông minh trong nhà máy với tốc độ cao, độ ổn định tốt và đáng tin cậy về dữ liệu.

1.4 Giới thiệu về Đồ Án

Trong những năm gần đây, sự phát triển của giao tiếp Machine-to-Machine (M2M) và Internet of Things (IoT) trong Công nghiệp đã mở đường cho việc phát triển các máy móc thông minh và điều khiển tự động trong các doanh nghiệp sản xuất. Các máy tự động này có khả năng thu thập dữ liệu từ các thiết bị cảm biến khác nhau và giao tiếp với các hệ thống máy được kết nối khác bằng cách sử dụng các giao thức truyền thông. Điều này được thể hiện rõ nét nhất qua Lớp Connect theo như mô hình năm lớp của Timothy Chou đã được trình bày ở phần trên. Và để đảm bảo được việc các thiết bị có thể trao đổi thông tin với nhau có hiệu quả cao nhất về sự chính xác, và đồng bộ nhanh, đặc biệt là trong môi trường Công nghiệp - Smart Factory, thì việc lựa chọn ra một giao thức kết nối an toàn và phù hợp là điều vô cùng quan trọng. Vì thế, nội dung nghiên cứu của Đồ án này cũng sẽ tập trung vào Lớp Connect, lớp tạo nên kết nối vạn vật giữa các thiết bị - và cụ thể hơn là nghiên cứu và hiện thực một giao thức truyền thông chuyên dụng trong Công nghiệp hiện nay - **OPC-UA**.

Nội dung nghiên cứu tổng quan gồm những vấn đề chính sau:

- **Các giao thức chuyên dụng cho Smart Factory**

Trong nội dung nghiên cứu này, hai giao thức truyền thông thông dụng của M2M (MQTT và OPC-UA) sẽ được phân tích để phân biệt rõ ưu điểm và nhược điểm của chúng. Từ đó, các giao thức sẽ được lựa chọn và áp dụng phù hợp vào ứng dụng Smart Factory.

- **Thực nghiệm so sánh các giao thức OPC-UA và MQTT**

Trong nội dung nghiên cứu này, chúng em sẽ xây dựng hai kiến trúc tương đương nhau cho các giao thức OPC UA và MQTT rồi tiến tới so sánh hiệu năng của chúng dựa trên việc so sánh Round Trip Time

(RTT) của hai giao thức khi chúng được áp dụng vào các kiến trúc đã xây dựng, qua đó đánh giá được tốc độ gửi nhận của gói tin trong một số tình huống khi sử dụng giao thức OPC-UA hoặc MQTT. Từ đó, có thể nhận thấy được giao thức nào chiếm ưu thế và phù hợp hơn trong môi trường Công nghiệp.

- **Hiện thực điều khiển cánh tay Robot dựa trên OPC-UA**

Và cuối cùng, để kiểm nghiệm hiệu quả của giao thức OPC-UA, một số thiết bị đơn giản sẽ được sử dụng để thực nghiệm minh họa cho một số hoạt động cơ bản trong công nghiệp (Cánh tay Robot, Ứng dụng điện thoại điều khiển cánh tay...)

CHAPTER 2

TỔNG QUAN VỀ CÁC GIAO THỨC M2M TRONG CÔNG NGHIỆP 4.0

Chương này sẽ trình bày những kiến thức tổng quan về các loại giao thức M2M chuyên dụng trong môi trường Công nghiệp. Những kiến thức này cũng sẽ hỗ trợ cho quá trình nghiên cứu cũng như góp phần vào việc triển khai hệ thống của chúng em. Bao gồm một số thông tin cơ bản về công nghệ Machine-to-Machine (M2M) và các giao thức truyền thông thông dụng của M2M như MQTT và OPC UA. Với mỗi điểm kiến thức được nhắc tới, chúng em sẽ phân tích về khái niệm, nguyên lý hoạt động cũng như ứng dụng của nó trong thực tế, đời sống.

2.1 Machine-to-Machine (M2M)

Giao tiếp Machine-to-Machine (M2M) là một công nghệ đầy hứa hẹn cho các hệ thống giao tiếp thế hệ tiếp theo. Mô hình giao tiếp này tạo điều kiện cho các giao tiếp phổ biến với cơ chế tự động hóa hoàn toàn, trong đó một số lượng lớn thiết bị thông minh được kết nối bằng liên kết có dây/không dây, tương tác với nhau mà không cần sự can thiệp trực tiếp của con người. Do đó, giao tiếp M2M thường được ứng dụng trong các lĩnh vực rộng lớn như lưới điện thông minh, chăm sóc sức khỏe điện tử, mạng khu vực gia đình, hệ thống giao thông thông minh, giám sát môi trường, thành phố thông minh và tự động hóa công nghiệp. Để hiểu rõ hơn về công nghệ này, chúng ta sẽ cùng nhau tìm hiểu về Khái niệm, Cách thức hoạt động và Ứng dụng của nó trong đời sống.

2.1.1 M2M là gì?

M2M là viết tắt của “Machine to Machine.” Đó là một thuật ngữ mô tả bất kỳ công nghệ nào cho phép các thiết bị có kết nối mạng trao đổi thông tin và thực hiện các hành động mà không cần sự trợ giúp thủ công của con người. Nói cách khác, giao tiếp được thực hiện từ máy này sang máy khác.

Công nghệ M2M cho phép các thiết bị trên mạng đưa ra quyết định tự chủ mà không yêu cầu các thao tác thủ công. Mặc dù được sử dụng rộng rãi trong sản xuất, nhưng nó cũng được sử dụng trong các lĩnh vực khác. Các ứng dụng phổ biến bao gồm chăm sóc sức khỏe, bảo hiểm và Internet vạn vật (IoT).

Khi ở trong ngành sản xuất, nó thường được kết hợp với các công nghệ khác như SCADA. Một ví dụ về kết nối M2M trong các lĩnh vực khác là máy bán hàng tự động. Nó tự động gửi thông tin kiểm kê hàng trong kho của mình cho người điều phối. Một ví dụ điển hình khác là máy ATM tự gửi thông tin khi gần hết tiền mặt cho cơ quan chức năng của nó.

2.1.2 M2M hoạt động như thế nào?

Kết nối M2M có vẻ phức tạp, nhưng chúng hoạt động theo nguyên tắc đơn giản. Chúng lấy dữ liệu cảm biến tự động và chuyển dữ liệu đó qua mạng truy cập công cộng (public access network). Không giống như SCADA, các kết nối M2M sử dụng các mạng công cộng như mạng di động hoặc Ethernet. Đây là những gì làm cho các kết nối M2M trở nên hiệu quả về chi phí.

Một hệ thống M2M điển hình bao gồm nhiều cảm biến, RFID, Wi-Fi hoặc liên kết truyền thông di động (cellular communications link). Các cảm biến giao tiếp một số key condition với máy tính. Chẳng hạn, trong máy ATM, một cảm biến phát hiện mức tiền sẽ gửi tín hiệu khi tiền ở mức thấp.

Các hệ thống M2M cũng thường chứa phần mềm tính toán tự trị (autonomic computing software). Phần mềm này được thiết kế để giúp mạng diễn giải dữ liệu mà nó nhận được và hành động tương ứng.

Ngoài khả năng giám sát thiết bị và hệ thống từ xa, những lợi ích hàng đầu của M2M bao gồm:

- **giảm chi phí** bằng cách giảm thiểu thời gian bảo trì và ngừng hoạt động của thiết bị;

- **tăng doanh thu** bằng cách mở ra các cơ hội kinh doanh mới để phục vụ các sản phẩm trong lĩnh vực này; và
- **cải thiện dịch vụ khách hàng** bằng cách chủ động theo dõi và bảo dưỡng thiết bị trước khi thiết bị hỏng hoặc khi cần thiết.

2.1.3 Ứng dụng của M2M

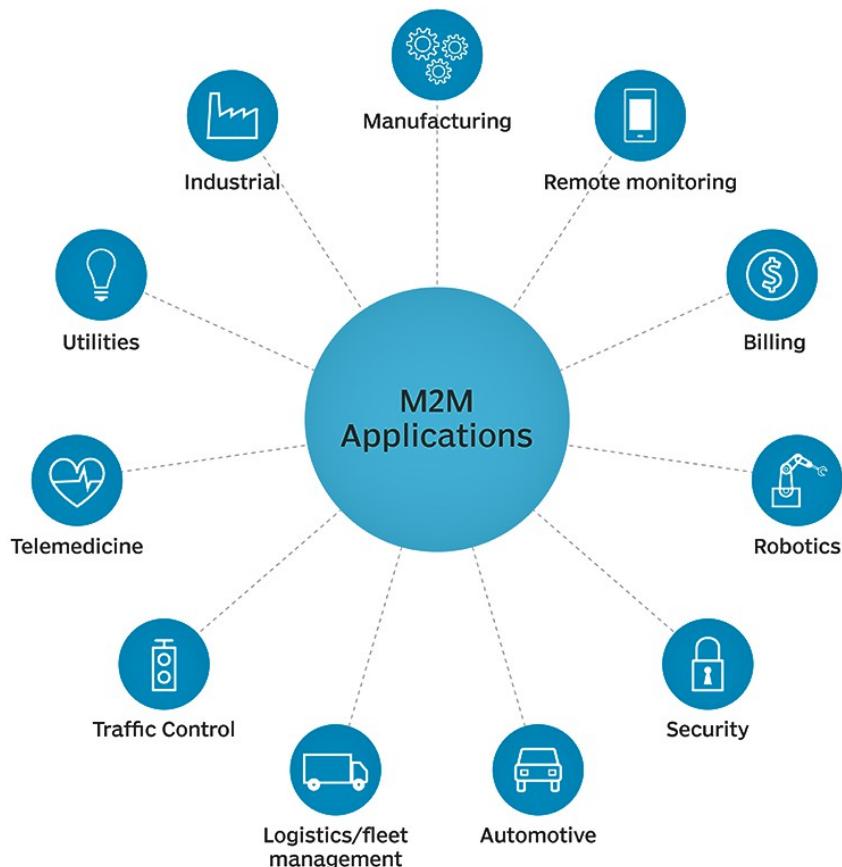


Figure 2.1: Ứng dụng của M2M

Kết nối M2M ở xung quanh chúng ta. Dưới đây là một vài ví dụ điển hình:

Máy bán hàng tự động Một cảm biến phát hiện khi một sản phẩm cuối cùng sắp hết trong máy bán hàng tự động. Thông tin này được truyền qua Wi-Fi hoặc mạng di động đến máy tính của người điều phối. Quá trình này hoàn toàn tự động.

Y tế từ xa Trong lĩnh vực y tế từ xa, kết nối M2M có thể sử dụng các cảm biến để theo dõi thông tin quan trọng của bệnh nhân. Thông tin này sau đó có thể được chuyển đến một máy cục bộ gửi thuốc hoặc bác sĩ ở xa.

Hệ thống nhà thông minh Có lẽ không lĩnh vực nào sử dụng kết nối M2M nhiều hơn hệ thống nhà thông minh. Có rất nhiều ví dụ về điều này. Điểm mấu chốt là nó cho phép một số cảm biến nhất định kết nối với các thiết bị khác để tự động hóa ngôi nhà của bạn.

2.2 MQTT

MQTT là giao thức truyền gói tin được sử dụng phổ biến nhất cho Internet of Things (IoT). Giao thức này là một bộ quy tắc xác định cách các thiết bị IoT có thể publish và subscribe dữ liệu qua Internet. MQTT được sử dụng để truyền tin và trao đổi dữ liệu giữa các thiết bị IoT và IoT công nghiệp (IIoT), chẳng hạn như thiết bị nhúng, cảm biến, PLC công nghiệp, v.v. Để tìm hiểu rõ hơn về giao thức MQTT, phần tiếp theo sẽ trình bày Khái niệm, Nguyên lý hoạt động, Ưu và nhược điểm của giao thức này cũng như Ứng dụng của nó trong đời sống.

2.2.1 MQTT là gì?

MQTT (Message Queueing Telemetry Transport) được công bố là một giao thức cho các ứng dụng machine-to-machine theo mô hình publish/subscribe cực kì nhẹ và là một giao thức kết nối Internet of Things. Đây là một giao thức truyền thông điệp (message) mở, chủ yếu tập trung vào việc sử dụng băng thông thấp, độ tin cậy cao và có khả năng hoạt động trong điều kiện đường truyền không ổn định. Do đó, liên lạc giữa các cảm biến thông qua liên kết vệ tinh là một trong những ứng dụng của MQTT. Kể từ năm 2013, MQTT được tiêu chuẩn hóa thành một giao thức cho Internet of Things bởi Tổ chức Organization for the Advancement of Structured Information Standards (OASIS).

Mặc dù MQTT bắt đầu như một giao thức độc quyền được sử dụng để giao tiếp với các hệ thống thu thập dữ liệu và kiểm soát giám sát (SCADA) trong ngành dầu khí, nhưng nó đã trở nên phổ biến trong lĩnh vực thiết bị thông minh và ngày nay là giao thức nguồn mở hàng đầu để kết nối internet vạn vật (IoT) và thiết bị IoT công nghiệp (IIoT).

2.2.2 MQTT hoạt động như thế nào?

Nhằm mục đích tối đa hóa băng thông có sẵn, mô hình giao tiếp publish/subscribe (pub/sub) của MQTT là một giải pháp thay thế cho kiến trúc client-server truyền thống - giao tiếp trực tiếp với điểm cuối. Ngược lại, trong mô hình pub/sub, ứng dụng khách gửi tin nhắn (message) (the publisher) được tách biệt khỏi các ứng dụng khách nhận tin nhắn (the subscribers). Bởi vì cả publisher và subscriber đều không liên hệ trực tiếp với nhau, nên bên thứ ba – các broker – sẽ chịu trách nhiệm cho các kết nối giữa chúng.

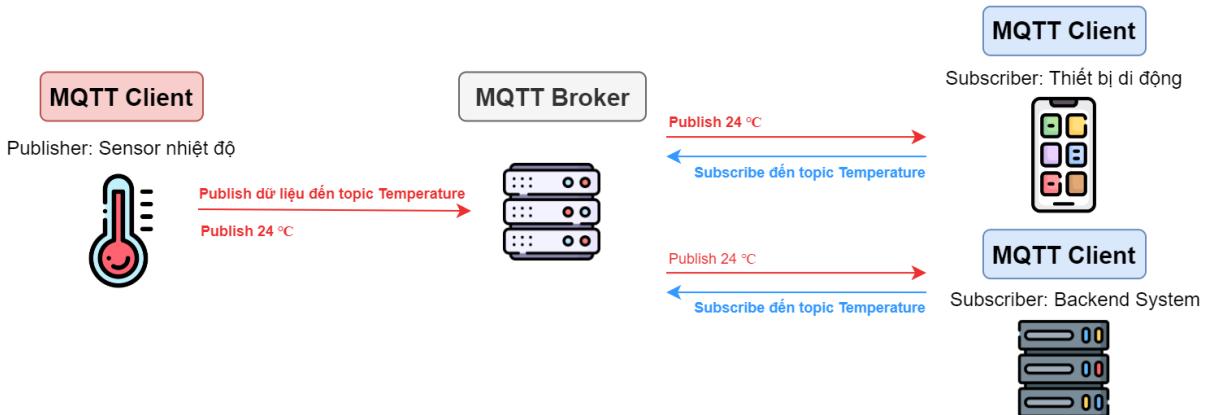


Figure 2.2: Kiến trúc chung của MQTT sử dụng MQTT Broker

Ứng dụng khách MQTT (MQTT client) bao gồm publishers và subscribers, thuật ngữ này đề cập đến việc ứng dụng khách đang xuất bản tin nhắn (message) hay đăng ký nhận tin nhắn. Hai chức năng này có thể được triển khai trong cùng một máy khách MQTT. Khi một thiết bị (hoặc máy khách) muốn gửi dữ liệu đến máy chủ (hoặc broker), nó được gọi là *publish*. Khi hoạt động được đảo ngược, nó được gọi là *subscribe*. Theo mô hình pub/sub, nhiều client có thể kết nối với một broker và subscribe các *topic* mà chúng quan tâm.

Một bài viết của IBM mô tả mô hình pub/sub: "Những Publisher gửi tin nhắn, những Subscriber nhận tin nhắn mà chúng quan tâm và Broker chuyển tin nhắn từ Publisher đến Subscriber. Publisher và Subscriber là ứng dụng khách MQTT (MQTT client), chỉ giao tiếp với một MQTT broker. Máy khách MQTT có thể là bất kỳ thiết bị hoặc ứng dụng nào (từ bộ vi điều khiển như Arduino đến máy chủ ứng dụng đầy đủ được lưu trữ trên đám mây) chạy thư viện MQTT."

2.2.2.1 Message

Một cách khác để MQTT giảm thiểu quá trình truyền của nó là sử dụng cấu trúc message nhỏ, được xác định chặt chẽ. Mỗi message có một header cố định chỉ 2 byte. Có thể sử dụng header tùy chọn nhưng sẽ làm tăng kích thước của message. Tải trọng message được giới hạn chỉ 256 MB. Ba mức Chất lượng Dịch vụ (QoS) khác nhau cho phép các nhà thiết kế mạng lựa chọn giữa giảm thiểu truyền dữ liệu và tối đa hóa độ tin cậy.

- **QoS 0** – Cung cấp lượng truyền dữ liệu tối thiểu. Với cấp độ này, mỗi message được gửi đến một subscriber một lần mà không cần xác nhận lại. Không có cách nào để biết liệu subscriber có nhận được tin nhắn hay không. Phương pháp này đôi khi được gọi là "fire and forget" hoặc "at most once delivery". Bởi vì cấp độ này giả định rằng quá trình gửi đã hoàn tất, nên các message không được lưu trữ để gửi cho các máy khách đã ngắt kết nối khi nó kết nối lại sau này.
- **QoS 1** – Broker gửi message và sau đó đợi phản hồi xác nhận từ subscriber. Nếu không nhận được xác nhận trong khung thời gian đã chỉ định, message sẽ được gửi lại. Sử dụng phương pháp này, subscriber có thể nhận được tin nhắn nhiều lần nếu broker không nhận được xác nhận của subscriber kịp thời. Và nó đôi khi được gọi là "at least once delivery".
- **QoS 2** – Client và broker sử dụng four-step handshake để đảm bảo rằng message được nhận và message đó chỉ được nhận một lần. Điều này được gọi là "exactly once delivery".

Đối với các trường hợp mà giao tiếp là đáng tin cậy nhưng tài nguyên bị hạn chế, thì QoS 0 có thể là lựa chọn tốt nhất. Đối với các trường hợp mà giao tiếp là không đáng tin cậy, nhưng khi các kết nối không bị giới hạn về tài nguyên, thì QoS 2 sẽ là lựa chọn tốt nhất. Còn QoS 1 là một loại giải pháp tốt cho cả hai trường hợp nhưng nó yêu cầu ứng dụng dữ liệu phải biết cách xử lý các bản sao từ việc gửi message nhiều lần.

2.2.2.2 Topic

Message trong MQTT được publish dưới dạng topic. Các Topic là các cấu trúc trong hệ thống phân cấp sử dụng ký tự gạch chéo (/) làm dấu phân

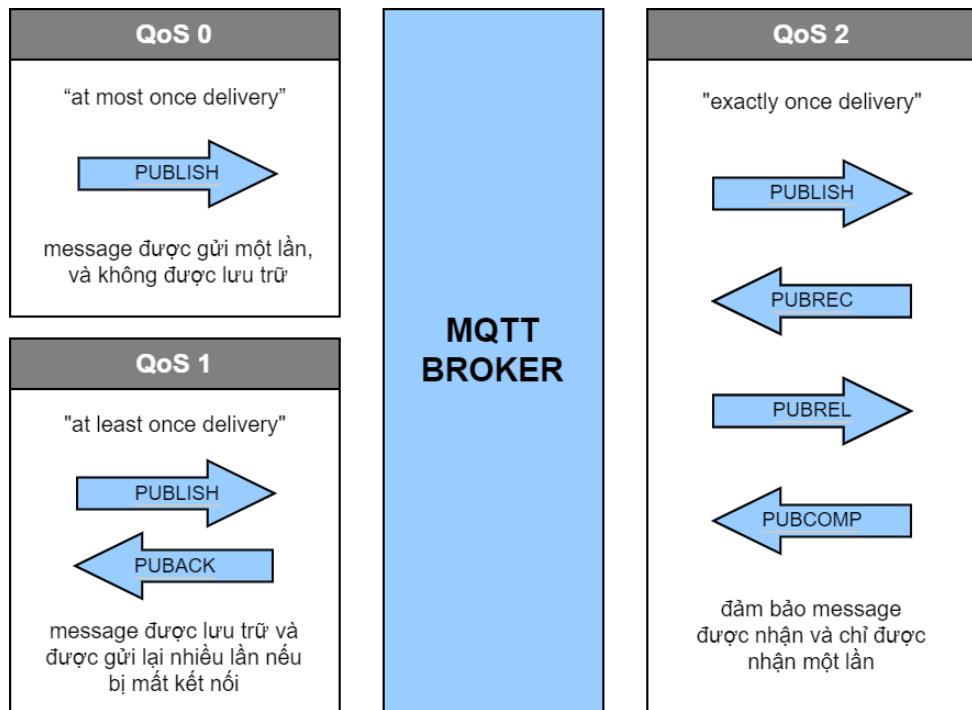


Figure 2.3: Ba mức Chất lượng Dịch vụ (QoS)

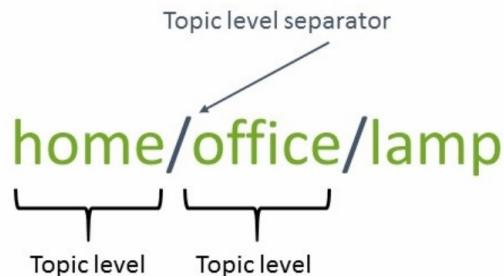


Figure 2.4: Cấu trúc của topic

cách. Cấu trúc này tương tự như cấu trúc của cây thư mục trên hệ thống tệp máy tính. Một cấu trúc như **sensors/OilandGas/Pressure/** cho phép subscriber chỉ định rằng chỉ nên gửi dữ liệu từ các máy khách publisher lên topic Pressure hoặc để có cái nhìn rộng hơn, có lẽ tất cả dữ liệu từ các máy khách publisher lên bất kỳ topic **sensors/OilandGas** nào. Topic không được tạo rõ ràng trong MQTT. Nếu một broker nhận được dữ liệu được publish cho một topic hiện không tồn tại, topic đó sẽ được tạo đơn giản và client có thể subscribe topic mới.

2.2.3 Ưu và Nhược điểm của MQTT

2.2.3.1 Ưu điểm

Các thuộc tính lightweight và chi phí tối thiểu của kiến trúc giao thức MQTT giúp đảm bảo truyền dữ liệu trơn tru với băng thông thấp và giảm tải cho CPU và RAM. Các ưu điểm của MQTT so với các giao thức cạnh tranh như sau:

- truyền dữ liệu hiệu quả và triển khai nhanh chóng do đây là một giao thức lightweight;
- mức sử dụng mạng thấp, do các gói dữ liệu được giảm thiểu;
- phân phối dữ liệu hiệu quả;
- thực hiện thành công remote sensing và điều khiển;
- truyền tải thông điệp nhanh chóng, hiệu quả;
- sử dụng lượng điện năng nhỏ, tốt cho các thiết bị được kết nối;
- tối ưu hóa băng thông mạng.

2.2.3.2 Nhược điểm

Nhược điểm tiềm ẩn đối với MQTT bao gồm:

- MQTT có chu kỳ truyền chậm hơn so với Constrained Application Protocol (CoAP).
- Khám phá tài nguyên của MQTT hoạt động trên đăng ký chủ đề linh hoạt, trong khi CoAP sử dụng hệ thống khám phá tài nguyên ổn định.
- MQTT không được mã hóa. Thay vào đó, nó sử dụng TLS/SSL (Transport Layer Security/Secure Sockets Layer) để mã hóa bảo mật.
- Rất khó để tạo một mạng MQTT có thể mở rộng toàn cầu.
- Các thách thức khác liên quan đến bảo mật, khả năng tương tác và xác thực.

Do giao thức MQTT không được thiết kế với mục đích bảo mật nên giao thức này thường được sử dụng trong các mạng back-end an toàn cho các mục đích dành riêng cho ứng dụng. Cấu trúc topic của MQTT có thể dễ dàng tạo thành một cây khổng lồ và không có cách rõ ràng nào để chia cây thành các miền logic nhỏ hơn. Điều này gây khó khăn cho việc tạo ra mạng MQTT có thể mở rộng toàn cầu bởi vì khi kích thước của cây chủ đề tăng lên, độ phức tạp sẽ tăng lên.

Một nhược điểm khác của MQTT là thiếu khả năng tương tác (interoperability). Bởi vì payload của message là binary, nếu không có thông tin về cách chúng được mã hóa, thì các vấn đề có thể phát sinh – đặc biệt là trong các kiến trúc mở, nơi các ứng dụng khác nhau từ các nhà sản xuất khác nhau hoạt động cùng nhau.

Như đã đề cập trước đây, MQTT có các tính năng xác thực tối thiểu được tích hợp trong giao thức. Tên người dùng và mật khẩu được gửi ở dạng văn bản rõ ràng và bất kỳ hình thức sử dụng MQTT an toàn nào cũng phải sử dụng SSL/TLS, và thật không may, đây không phải là một giao thức nhẹ.

Xác thực ứng dụng khách (client) bằng client-side certificate không phải là một quy trình đơn giản và MQTT không có cách nào để kiểm soát ai sở hữu topic và ai có thể publish thông tin trên đó, ngoại trừ sử dụng các phương tiện độc quyền (proprietary), ngoài băng tần (out-of-band). Điều này giúp dễ dàng đưa các message có hại vào mạng, do cố ý hoặc do nhầm lẫn.

Hơn nữa, không có cách nào để người nhận message biết ai đã gửi message gốc trừ khi thông tin đó có trong message. Các tính năng bảo mật phải được triển khai trên MQTT theo kiểu độc quyền sẽ làm tăng code footprint và khiến việc triển khai trở nên khó khăn hơn.

2.2.4 Ứng dụng của MQTT

Do các đặc tính nhẹ của nó, MQTT hoạt động tốt cho các ứng dụng liên quan đến giám sát từ xa, bao gồm:

- đồng bộ hóa các cảm biến, chẳng hạn như đầu báo cháy hoặc cảm biến chuyển động để phát hiện hành vi trộm cắp, để xác định xem mối nguy hiểm có hợp lệ hay không;
- theo dõi các thông số sức khỏe bằng cảm biến cho các bệnh nhân đã

xuất viện; và

- cảm biến cảnh báo con người về sự nguy hiểm.

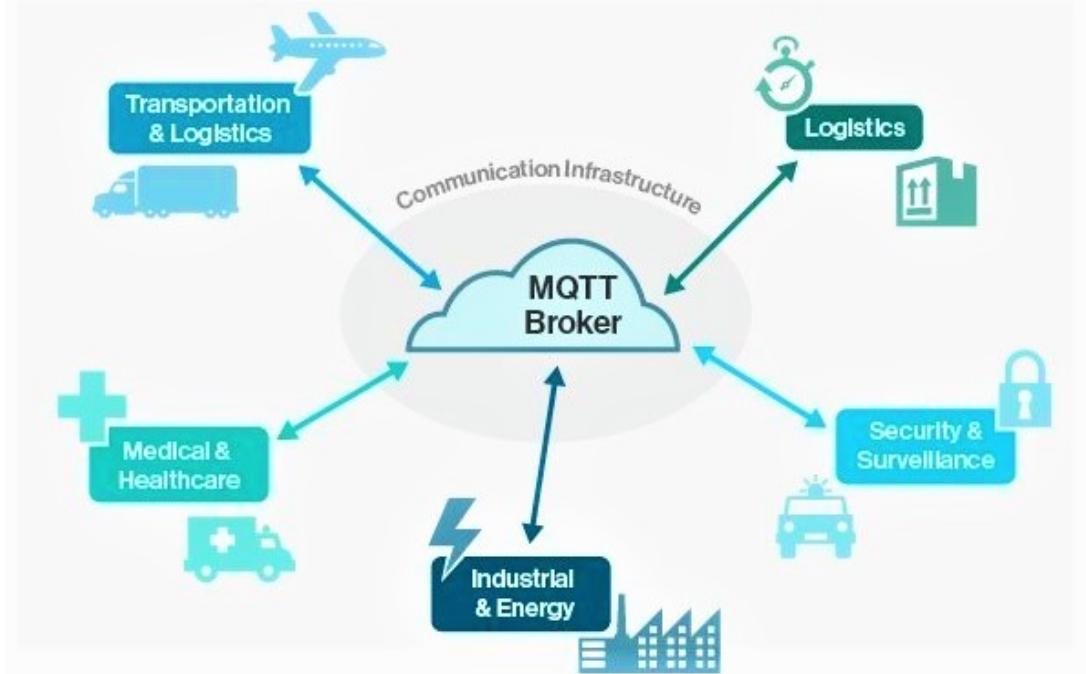


Figure 2.5: Ứng dụng của MQTT

Một ứng dụng khác là ứng dụng nhắn tin dựa trên văn bản để liên lạc theo thời gian thực, tận dụng mức sử dụng năng lượng và dữ liệu thấp của MQTT. Ví dụ: Facebook sử dụng MQTT cho ứng dụng Messenger của mình, không chỉ vì giao thức này tiết kiệm pin trong khi nhắn tin từ điện thoại đến điện thoại mà còn vì giao thức này cho phép gửi tin nhắn hiệu quả trong một phần nghìn giây, mặc dù kết nối internet không nhất quán trên toàn cầu.

Hầu hết các nhà cung cấp dịch vụ đám mây, bao gồm Amazon Web Services (AWS), Google Cloud, IBM Cloud và Microsoft Azure, đều hỗ trợ MQTT.

MQTT rất phù hợp với các ứng dụng sử dụng thiết bị M2M và IoT cho các mục đích như phân tích thời gian thực, bảo trì và giám sát phòng ngừa trong các môi trường, bao gồm nhà thông minh, chăm sóc sức khỏe, logistics, công nghiệp và sản xuất.

2.3 OPC UA

OPC Unified Architecture (OPC UA) là một giao thức giao tiếp machine-to-machine được sử dụng cho tự động hóa công nghiệp và được phát triển bởi OPC Foundation. OPC UA đơn giản hóa kết nối công nghiệp để bạn có thể tích hợp tất cả các thiết bị, hệ thống tự động hóa và ứng dụng phần mềm của mình bằng cách sử dụng tiêu chuẩn an toàn và độc lập với nền tảng. Phần tiếp theo sẽ trình bày rõ hơn về Khái niệm giao thức OPC-UA cũng như Sự cải tiến của OPC-UA so với phiên bản trước đó của nó - OPC Classic.Thêm vào đó, chúng ta sẽ tìm hiểu về Nguyên lý hoạt động của OPC-UA, phân tích một số Ưu - nhược điểm của giao thức này và điểm qua các ứng dụng của nó trong Công nghiệp 4.0.

2.3.1 OPC UA là gì?

OPC UA (Open Platform Communications Unified Architecture) là một giao thức truyền thông machine-to-machine theo hướng dịch vụ, chủ yếu được sử dụng trong tự động hóa công nghiệp và được định nghĩa trong IEC 62541. Các mục tiêu chính của nó là cung cấp một giao thức truyền thông đa nền tảng khi sử dụng mô hình thông tin để mô tả dữ liệu được truyền. Các tính năng và thành phần khác nhau của OPC UA được mô tả trong các phần đặc tả kỹ thuật khác nhau do OPC Foundation phát hành và công bố rộng rãi . OPC UA chủ yếu được thúc đẩy bởi ngành công nghiệp sản xuất ở Châu Âu, do đó nó ngày càng giành được vị trí quan trọng lĩnh vực này, đồng thời nó cũng đang trở thành một trong những giao thức trọng yếu trên toàn thế giới. Điểm mạnh chính của OPC UA là mô tả ngữ nghĩa của mô hình không gian địa chỉ cùng với những thông số kỹ thuật đồng hành khác nhau giúp mở rộng các mô tả ngữ nghĩa cơ bản cho các miền khác nhau như PLCopen, rô bốt hoặc thị giác máy tính.

OPC UA tương thích với Windows, macOS, Android và Linux. Nó cũng có thể được sử dụng trong các hệ thống nhúng và hệ thống bare-metal - hệ thống không sử dụng hệ điều hành. OPC UA hoạt động trên PC, cloud-based infrastructures, PLC, micro-controllers và cyber physical systems (CPS).

2.3.2 Điểm khác nhau giữa OPC UA và OPC Classic

OPC Classic, tiền thân của OPC UA, dựa trên các công nghệ của Microsoft trong khi OPC UA độc lập với nền tảng.

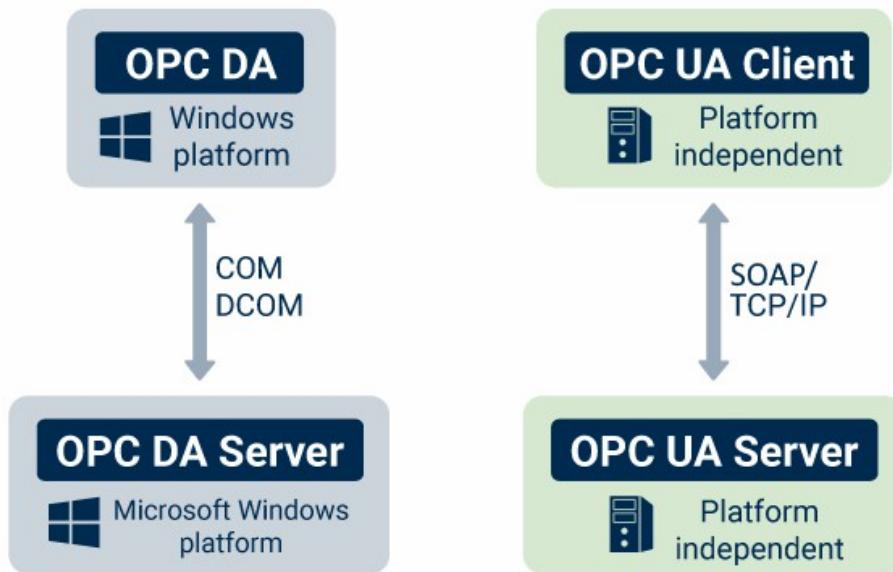


Figure 2.6: OPC Classic vs. OPC UA

OPC Classic không có bảo mật tích hợp để kiểm soát truy cập, xác thực và mã hóa. OPC UA cho phép mã hóa dữ liệu tại nguồn dữ liệu, đảm bảo truyền an toàn mà không cần dựa vào tường lửa mạng tại lõi của hệ thống. Điều này có nghĩa là tính bảo mật được đảm bảo ngay từ khi bắt đầu truyền dữ liệu, thay vì chỉ được xác nhận khi dữ liệu đến tường lửa của mạng. OPC UA triển khai bảo mật đa nền tảng dựa trên public key infrastructure (PKI) bằng chứng chỉ kỹ thuật số x.509 tiêu chuẩn ngành.

Trái ngược với OPC UA, OPC Classic không hỗ trợ mô hình thông tin động, thứ cho phép các nhà sản xuất xác định các mô hình dữ liệu tùy chỉnh tùy thuộc vào yêu cầu của ngành của họ.

Trong OPC UA, các thông số kỹ thuật của OPC Classic cung cấp các mô-đun chức năng có thể được truy vấn đặc biệt. Các thông số kỹ thuật này là OPC DA (Truy cập dữ liệu), OPC AE (Cảnh báo và sự kiện), OPC Security, OPC Batch, OPC Command, OPC XML, OPC Data Exchange (DX) và OPC HAD (Dữ liệu truy cập lịch sử). Chúng thường được gọi chung là OPC Classic hoặc đơn giản là OPC. Trong OPC UA, chúng được gọi là profile. Ví dụ: tiêu chuẩn OPC DA là một profile OPC UA DA.

Các profile này tạo thành một lớp nằm trên lớp cơ sở của OPC UA, lớp này chạy các dịch vụ phổ biến. Tất cả các profile OPC UA sử dụng cùng một cơ sở mã chung trong khi đó, trong thiết lập OPC Classic, mỗi tiêu chuẩn có cơ sở mã riêng, điều này tạo ra sự trùng lặp trong các hệ thống - có nhiều hơn một đặc tả OPC Classic được triển khai.

2.3.3 OPC UA hoạt động như thế nào?

Model

Sử dụng các mô hình, OPC UA chỉ định các quy tắc cơ bản để hiển thị dữ liệu cho bất kỳ ứng dụng hoặc thiết bị nào muốn sử dụng nó. Bản thân OPC UA là một mô hình dữ liệu tập trung vào thông tin (information-centric data). Nó bao gồm một mô hình đối tượng chung (generic object model) với một hệ thống kiểu có thể mở rộng với các mô hình tích hợp để truy cập dữ liệu. Các mô hình tích hợp này chỉ định các chức năng như thông tin báo động và sự kiện, thông tin về dữ liệu lịch sử, chi tiết truy cập dữ liệu, mô tả thiết bị và để thực thi các chương trình.

Dữ liệu cũng có thể được truy cập thông qua các mô hình tùy chỉnh, được gọi là mô hình đồng hành (companion model). Chúng được sử dụng trong các ngành công nghiệp khác nhau như sản xuất máy ép phun hoặc kỹ thuật chế tạo robot.

Luồng dữ liệu và kết nối

OPC UA hỗ trợ giao tiếp giữa các thành phần ở năm level trong các tổ chức công nghiệp: enterprise, management, operations, control, and field (thiết bị dành riêng cho nhà cung cấp).

Các thiết bị hiển thị dữ liệu của chúng thông qua OPC UA, cho phép vận chuyển thông tin này qua mạng tới một ứng dụng tiêu thụ bằng các dịch vụ web tiêu chuẩn. Dữ liệu được vận chuyển bằng các giao thức dựa trên IP và SOAP nhờ đó các máy chủ cấp thấp (low-end server) có thể sử dụng UA TCP. Việc sử dụng các dịch vụ web SOAP tiêu chuẩn qua HTTP cho phép các máy khách không thuộc OPC UA có thể yêu cầu dữ liệu do máy chủ OPC UA publish.

Phần mềm cầu nối và cổng (Bridging and gateway software) được gọi là OPC UA wrappers cho phép luồng dữ liệu trên phần cứng dành riêng cho

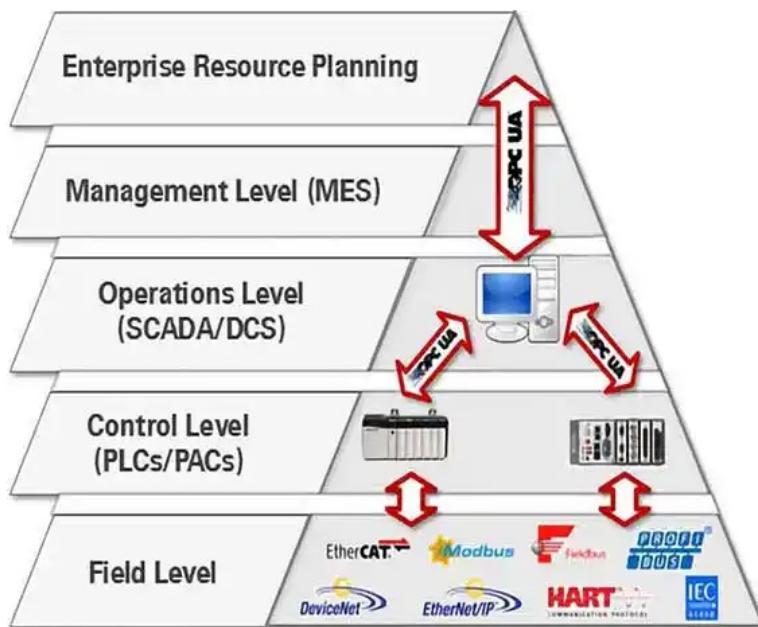


Figure 2.7: Năm level trong tổ chức công nghiệp

nhà cung cấp giữa các OPC UA level. OPC UA wrappers cũng có thể được sử dụng để chuyển đổi từ OPC Classic sang OPC UA hoặc khi máy chủ OPC hỗ trợ UA nhưng máy khách OPC thì không.

Kiến trúc hướng dịch vụ (Service-oriented architecture) - SOA

OPC UA giao tiếp dựa trên SOA client-server framework. Trong OPC UA, có máy chủ OPC UA (OPC UA servers) và máy khách OPC UA (OPC UA clients).

Máy chủ OPC UA (OPC UA server) là cơ sở của giao tiếp OPC UA. Nó là một phần mềm triển khai tiêu chuẩn OPC và từ đó cung cấp các giao diện OPC được tiêu chuẩn hóa cho thế giới bên ngoài. Máy chủ OPC UA cung cấp cho máy khách OPC UA các ứng dụng và hệ thống điều khiển, ví dụ như MES và SCADA, đồng thời có quyền truy cập an toàn vào dữ liệu tự động hóa công nghiệp bằng cách sử dụng các mô hình thông tin OPC UA chỉ định cách tổ chức, lưu trữ và thu thập dữ liệu. Thuật ngữ máy chủ OPC UA đề cập đến tiêu chuẩn phần mềm OPC UA trên máy chủ không phải bản thân phần cứng, có thể là một máy chủ ảo.

Máy khách OPC UA (OPC UA client) là máy khách có thể hỗ trợ mô hình thông tin OPC UA. Máy khách OPC UA yêu cầu dữ liệu và ghi dữ liệu vào các thành phần trong hệ thống thông qua máy chủ OPC UA. Do các Máy chủ OPC UA triển khai các giao diện được xác định trước của tiêu

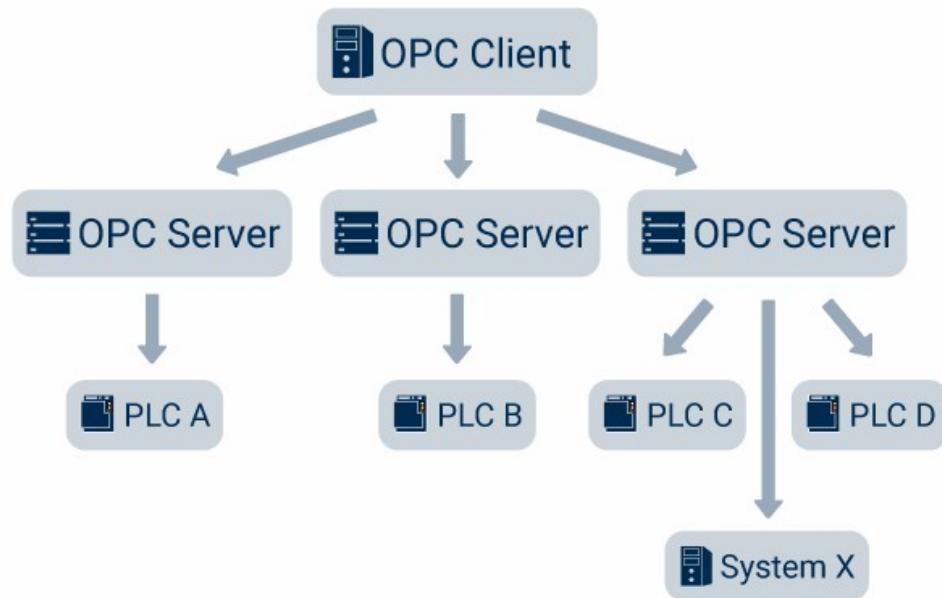


Figure 2.8: OPC UA server

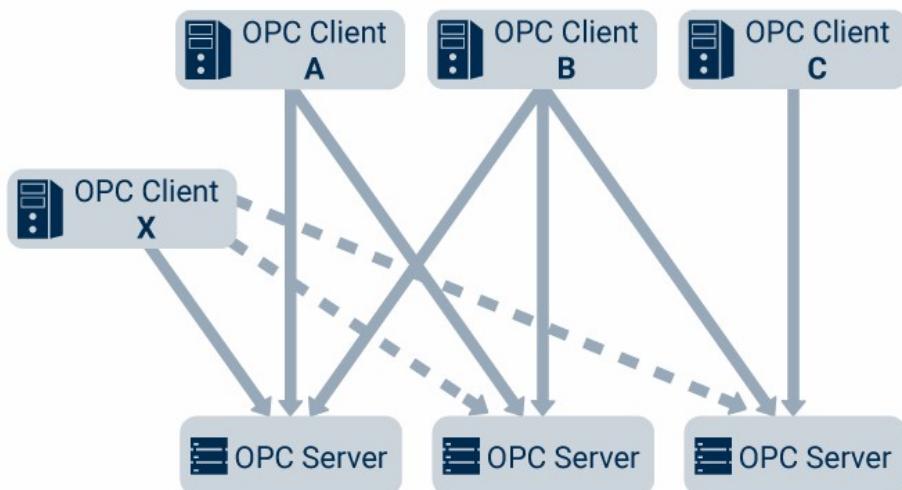


Figure 2.9: OPC UA client

chuẩn OPC UA nên mỗi máy khách có thể truy cập bất kỳ Máy chủ OPC UA nào và trao đổi dữ liệu với máy chủ theo cùng một cách.

Các hệ thống SOA như OPC UA tích hợp các ứng dụng khác nhau qua mạng và kết nối các thiết bị trên các nút mạng khác nhau.

Node

Node là đơn vị dữ liệu cơ bản trong không gian địa chỉ OPC UA, cung cấp một cách tiêu chuẩn cho các máy chủ OPC UA để biểu diễn các đối tượng cho các máy khách OPC UA. Các Node là các mẫu thông tin (ví dụ: nhiệt

độ duy nhất) và bao gồm các thuộc tính, giá trị dữ liệu thực và một hoặc nhiều tham chiếu đến các Node khác, mỗi Node nằm trong không gian địa chỉ riêng của nó. Do đó, một nhiệt độ duy nhất sẽ chiếm nhiều địa chỉ trong một không gian địa chỉ.

Các Node được tham chiếu bởi một Node ID duy nhất: một namespace URI (số nhận dạng tài nguyên duy nhất), loại dữ liệu (data type) và số nhận dạng (identifier). Mỗi Node thuộc về một namespace cụ thể. Namespace URI được đặt trong một bảng namespace riêng trên máy chủ OPC UA. Bảng namespace lưu trữ các URI riêng biệt cho các mô hình thông tin được sử dụng bởi các tổ chức riêng lẻ có yêu cầu riêng về việc dữ liệu sẽ trông như thế nào và hoạt động ra làm sao. Điều này cho phép OPC UA mở rộng các dịch vụ của mình mà không thay đổi thiết kế cơ bản của tiêu chuẩn.

Trong OPC UA, các Node có nhiều lớp cho phép tạo các biến thể trên Node cơ bản. Các lớp Node đối tượng trong OPC UA là chìa khóa để nó có thể tạo dữ liệu phức tạp và phân biệt giữa các thực thể tương tự nhưng khác nhau, ví dụ như cảm biến nhiệt độ cho máy điều hòa không khí và cảm biến nhiệt độ cho nồi hơi.

2.3.4 Ưu và nhược điểm của OPC UA

2.3.4.1 Ưu điểm

- **Phân quyền**

Về mặt lịch sử, kim tự tháp tự động hóa trong các hệ thống công nghiệp là một cấu trúc phân cấp mô tả luồng thông tin từ các thiết bị cấp thấp như bộ điều khiển, cảm biến hoặc đồng hồ đo đến các ứng dụng ERP cấp cao. Ở hướng ngược lại là một luồng kiểm soát, từ các ứng dụng ERP cấp cao đến các thiết bị cấp thấp. Các thành phần cấp thấp được kết nối qua mạng MES thông qua PLC và HMI.

OPC UA loại bỏ cấu trúc kim tự tháp này bằng cách phi tập trung hóa các thành phần hệ thống và tạo điều kiện thuận lợi cho việc sử dụng các cấu trúc mô hình hóa dữ liệu linh hoạt hơn trong mạng lưới. OPC UA đạt được điều này bằng cách xác định cấu trúc dữ liệu nhất quán mà tất cả các thành phần sử dụng, ví dụ: ứng dụng ERP và cảm biến trường đều có thể sử dụng cùng một mô hình thông tin.

- **Nền tảng độc lập**

Trước đây, các hệ thống công nghiệp chạy trên phần mềm dựa trên Windows. OPC UA là nền tảng bất khả tri; các hệ thống công nghiệp có thể tích hợp phần mềm từ bất kỳ nhà cung cấp nào, sử dụng bất kỳ hệ điều hành nào. OPC UA có thể được triển khai trên các hệ thống nhúng và trên đám mây.

- **Khả năng mở rộng**

OPC UA cho phép các tổ chức phát triển các hệ thống SCADA có thể mở rộng để thiết bị nhà máy hiện có có thể tích hợp với các mô-đun phần mềm mới mà không cần cấu hình bổ sung. Một ví dụ về điều này là trong ngành công nghiệp khí đốt và dầu mỏ, nơi có thể thu thập dữ liệu từ các cảm biến hiệu chuẩn, đo lường và đo lưu lượng từ xa, giúp thanh tra tại chỗ mà không phải kiểm tra thực tế việc lắp đặt.

- **Khả năng Khám phá**

OPC UA có khả năng plug-and-play. Khi các nhà máy từ xa mới được thêm vào một tổ chức hoặc các nhà cung cấp mới được đưa vào hoạt động, OPC UA có thể tự động khám phá mạng của họ, định cấu hình và tích hợp chúng vào mạng công ty.

- **Khả năng tương tác**

Khả năng tương tác của OPC UA cho phép người dùng cuối xây dựng các hệ thống công nghiệp tùy chỉnh bằng cách sử dụng các thiết bị và phần mềm từ các nhà cung cấp khác nhau.

2.3.4.2 Nhược điểm

- **Giới hạn tính năng cho thiết bị**

Một số nhà sản xuất phần mềm độc quyền đã báo cáo các hạn chế tính năng cho thiết bị, ví dụ như giữa máy chủ OPC UA và iFIX của General Electric và các thành phần HMI/SCADA được sử dụng trong các sản phẩm tự động hóa phần mềm của công ty. Những hạn chế này bao gồm việc thiếu hỗ trợ cho các tính năng cụ thể như Chữ ký điện tử, Chuyển đổi dự phòng nâng cao và các nguồn dữ liệu lịch sử.

- **Cấu hình phức tạp**

Trong thế giới thực, OPC UA thường quản lý việc trao đổi dữ liệu giữa

các hệ thống thông tin MES và SCADA và giữa các thiết bị cấp thấp. Nó lý tưởng cho việc giám sát và báo cáo hệ thống. Mặc dù được thiết kế để quản lý khả năng tương tác giữa các thiết bị không đồng nhất, nhưng nó đã bị chỉ trích là không linh hoạt khi xử lý các cấu trúc dữ liệu đa dạng từ các nhà cung cấp khác nhau và việc triển khai phức tạp

2.3.5 Ứng dụng của OPC UA

OPC UA được sử dụng trong các hệ thống công nghiệp, ví dụ như dầu khí, nông nghiệp, y tế và dược phẩm, các dịch vụ quan trọng như lưới điện và nhà máy xử lý nước thải cũng như các hệ thống IoT như ứng dụng thành phố thông minh.

Các ứng dụng OPC UA phổ biến bao gồm chẩn đoán thiết bị, quản lý tài sản, quản lý sản xuất, kiểm soát chất lượng, thu thập dữ liệu, báo cáo doanh nghiệp, bảo mật dữ liệu, tích hợp dữ liệu cho giao diện GUI, hỗ trợ nhân viên từ xa và giám sát sự kiện.

Các ví dụ trong thế giới thực bao gồm giám sát thời gian hoạt động của camera an ninh, gửi cảnh báo cho các cảm biến bị trực trặc, kiểm soát nhiệt độ văn phòng, quản lý máy tự động từ xa, ước tính khối lượng công việc, liên kết các thiết bị nhúng và hỗ trợ nhân viên từ xa.

OPC UA cũng hỗ trợ internet vạn vật (IIoT) công nghiệp. Ví dụ: OPC UA có thể được sử dụng để đẩy dữ liệu từ các thiết bị nhúng như cảm biến nhiệt độ lên đám mây, chẳng hạn như để phân tích hiệu quả sử dụng và thiết bị.

2.3.6 OPC UA trong Công nghiệp 4.0

Trong lĩnh vực sản xuất công nghiệp, các thuật ngữ *Công nghiệp 4.0* (*Industry 4.0*) và *Cách mạng công nghiệp lần thứ tư* (*Fourth Industrial Revolution*) được sử dụng thay thế cho nhau để chỉ xu hướng tự động hóa gia tăng, tập trung vào khả năng kết nối của thiết bị, học máy và Internet vạn vật cùng những thứ khác.

Các tính năng chính của OPC UA, bao gồm bảo mật tích hợp, khả năng lập mô hình thông tin, phát hiện thiết bị tự động, khả năng mở rộng, sử dụng dữ liệu ngữ nghĩa và tiêu chuẩn hóa giao thức, đáp ứng các yêu cầu về

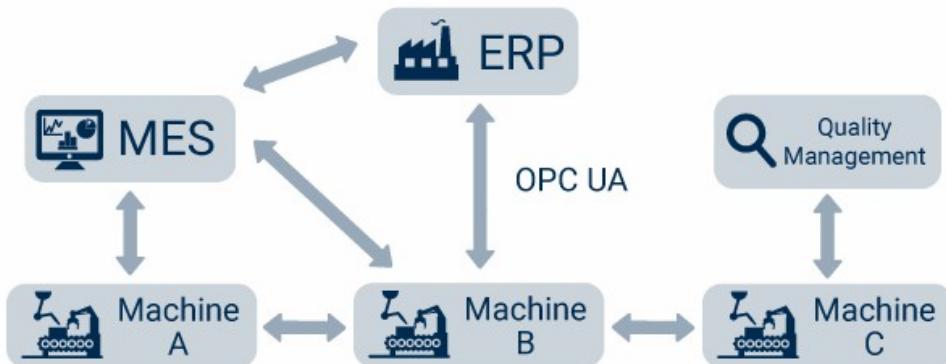


Figure 2.10: OPC UA and Industry 4.0

tuân thủ Công nghiệp 4.0.

Một trong những thách thức chính đối với Công nghiệp 4.0 trong sản xuất là thu thập dữ liệu thời gian thực từ các thiết bị cấp thấp. OPC UA cho phép một tổ chức công nghiệp nhúng máy chủ OPC UA vào tất cả các thiết bị của mình. Điều này có nghĩa là một lượng lớn dữ liệu thời gian thực có thể được định tuyến đến các hệ thống điều khiển và ứng dụng cấp doanh nghiệp để phân tích, sắp xếp và trao đổi với các ứng dụng tiêu thụ khác nhau.

Các nhóm làm việc quốc tế khác nhau góp phần biến OPC UA trở thành tiêu chuẩn truyền thông cho các sản phẩm và dịch vụ của Công nghiệp 4.0 trong các ngành cụ thể.

Để được Tổ chức OPC phân loại là tương thích với Công nghiệp 4.0, một sản phẩm phải tuân thủ tiêu chuẩn OPC UA, bằng cách sử dụng OPC UA tích hợp hoặc bằng cách sử dụng phần mềm cổng.

Tóm lại, Kiến trúc publish-subscribe của MQTT khác biệt đáng kể so với kiến trúc dựa trên client-server của OPC UA. Mỗi loại giao thức mang lại những lợi thế riêng và mỗi loại phù hợp hơn cho các trường hợp sử dụng khác nhau. Do đó, ở chương tiếp theo chúng em sẽ tập trung vào việc thực hiện thí nghiệm để so sánh và phân tích một vài thông số kỹ thuật thu được khi áp dụng hai loại giao thức này vào một số tình huống và kiến trúc nhất định.

CHAPTER 3

THỰC NGHIỆM SO SÁNH OPC-UA VÀ MQTT

Như đã trình bày ở phần trên, OPC-UA và MQTT là 2 giao thức phổ biến trong các ứng dụng IIOT - trong thời đại 4.0, nơi vạn vật kết nối với nhau và chia sẻ thông tin, dữ liệu trực tuyến. Ở chương 3 này, nhóm em sẽ tiến tới so sánh hiệu năng của hai giao thức dựa trên việc so sánh Round Trip Time (RTT) khi sử dụng giao thức trong kiến trúc tương đương nhau, qua đó đánh giá được tốc độ gửi nhận của gói tin trong một số tình huống khi sử dụng giao thức OPC-UA hoặc MQTT

3.1 Phương pháp thực nghiệm

Để tạo được tính khách quan trong quá trình thực nghiệm, trong chương này, nhóm đề xuất hai mô hình thực nghiệm tương đối giống nhau cho giao thức MQTT và OPC-UA, sau đó thực hiện một số thực nghiệm để đo đạc RTT nhằm nhận xét được tốc độ gửi nhận gói tin của hai giao thức. Với mỗi giao thức, nhóm sẽ tiến hành các thao tác gửi nhận với độ dài gói tin khác nhau, lặp lại 1000 lần cho mỗi độ dài gói tin. Ngoài ra, thực nghiệm sẽ được tiến hành trong hai tình huống: mạng bình thường và khi mạng có tải cao.

3.1.1 MQTT

Để thực hiện thực nghiệm đo đạc RTT sử dụng giao thức MQTT, nhóm đề xuất mô hình hệ thống như sau:

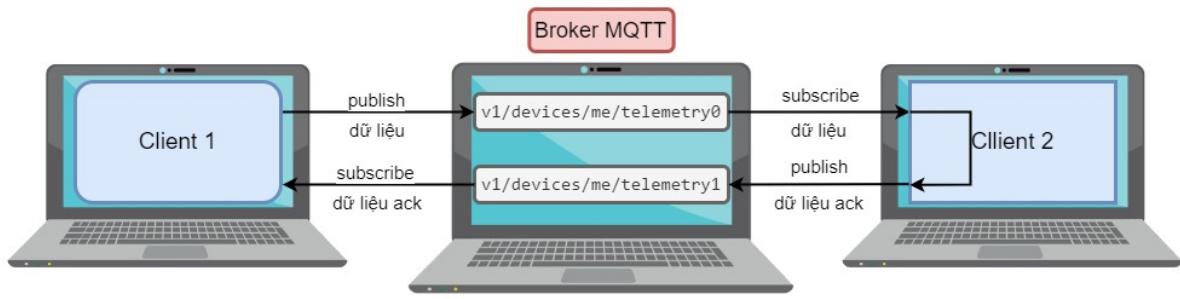


Figure 3.1: Mô hình hệ thống đo đạc RTT sử dụng MQTT

Từ Client_1, nhóm sẽ publish dữ liệu với độ dài cho trước tới topic **v1/devices/me/telemetry0**. Topic này sẽ được lắng nghe (subscribe) bởi Client_2. Khi Client_2 nhận được dữ liệu, nó sẽ phản hồi một ack data về **v1/devices/me/telemetry1**, sau đó được broker gửi ngược về Client_1. Nhóm sẽ tiến hành lưu thời gian trước khi Client_1 gửi data (T1) cho tới khi Client_1 nhận được ack data (T2). RTT sẽ được tính theo công thức sau:

$$RTT_{MQTT} = T2 - T1 \quad (3.1)$$

Để quan sát sự ảnh hưởng của độ dài gói tin lên RTT khi sử dụng giao thức, nhóm lặp lượt thử nghiệm gửi từ Client_1 data có các kích thước là 8b, 32b, 128b, 512b, 2Kb, 8Kb, 32Kb, 128Kb. Với mỗi kích thước data được gửi từ Client_1, thực nghiệm sẽ được lặp lại 1000 lần để có đánh giá tốt hơn về RTT và các thuộc tính liên quan ứng với mỗi kích thước data.

Mặt khác, vì MQTT hoạt động theo cơ chế publish - subscribe, khi Client_1 gửi data tới broker ở dạng publish, Client_1 sẽ không chờ phản hồi từ Client_2 gửi xuống mà sẽ gửi ngay data tiếp theo. Chính vì vậy, nhóm có hiện thực cơ chế để đảm bảo chỉ khi Client đã nhận được data ack từ Client_2 thì mới tiếp tục gửi data ở lần lặp sau để có thể dễ dàng tính toán RTT. Điều này có nghĩa, Client_1 sẽ thực hiện gửi data và chờ data ack từ Client_2 suốt 1000 lần với mỗi kích thước data một cách tuần tự, lần lặp tiếp theo chỉ bắt đầu khi Client_1 đã nhận được data ack từ Client_2.

Ngoài ra, thực nghiệm sẽ được thực hiện ở 2 trường hợp: điều kiện mạng bình thường và khi mạng có tải cao. Kết nối MQTT giữa broker và client sẽ không có bảo mật, và sử dụng QoS 0.

3.1.2 OPC-UA

Để tạo ra sự tương đồng, công bằng về kiến trúc giữa MQTT và OPC-UA trong đo đạc, nhóm đề xuất ra mô hình kiến trúc của OPC-UA như sau:

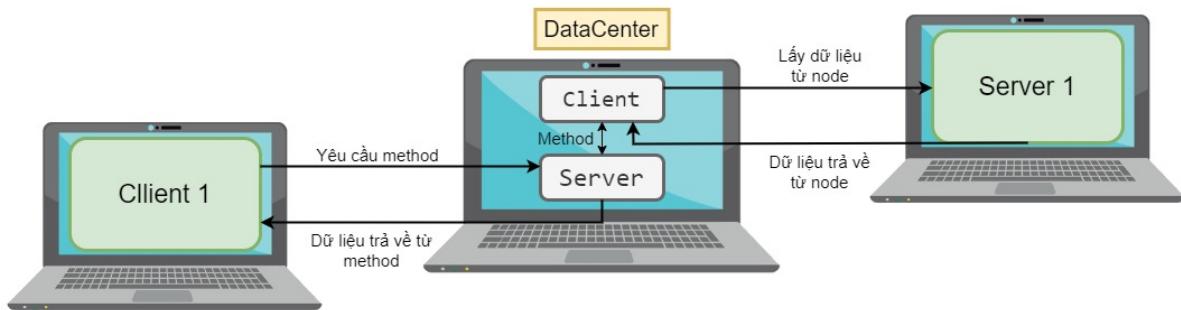


Figure 3.2: Mô hình hệ thống đo đạc RTT sử dụng OPC-UA

Mô hình kiến trúc có 3 thiết bị chính, bao gồm:

- **Client** : Kết nối tới **DataCenter** và gọi method để truy xuất về giá trị của node tương ứng
- **DataCenter**: **DataCenter** là một thiết bị chứa 1 sub client và 1 sub server, với nhiệm vụ tương ứng là kết nối tới **Server** để lấy ra các node dữ liệu và trả về dữ liệu tương ứng khi method được gọi ở **Client**
- **Server**: Đây là nơi lưu trữ các node dữ liệu để trả về cho **DataCenter** khi được truy xuất tới

Từ **Client**, chúng ta kết nối tới sub server của **DataCenter** và gọi method để lấy về giá trị của node tương ứng ở **Server** (Thời điểm này sẽ đánh dấu là T1), ở **DataCenter** sau khi sub server nhận được request call method từ **Client** thì thực thi một hàm tương ứng với method đã được gọi, hàm này sẽ trả về giá trị mà sub client lấy ra từ node tương ứng ở **Server** , cuối cùng giá trị node đã được lấy ra sẽ trả về cho **Client** (thời điểm sau khi nhận giá trị trả về được đánh dấu là T2). RTT của cả quá trình gửi nhận sẽ được tính theo công thức tương tự như ở MQTT:

$$RTT_{OPCUA} = T2 - T1 \quad (3.2)$$

Tương tự như ở MQTT, để quan sát sự ảnh hưởng của độ dài gói tin lên RTT khi sử dụng giao thức, nhóm lần lượt thử nghiệm với các node từ **Server** có các kích thước là 8b, 32b, 128b, 512b, 2Kb, 8Kb, 32Kb, 128Kb. Với mỗi kích thước data, thực nghiệm sẽ được lặp lại 1000 lần để có đánh giá tốt hơn về RTT và các thuộc tính liên quan ứng với mỗi kích thước data. Thực nghiệm được thực hiện cũng trong 2 trường hợp: điều kiện mạng ổn định và khi mạng có lượng tải cao. Tốc độ đường truyền sử dụng trong thực nghiệm với OPCUA là tương tự so với khi dùng trong thí nghiệm MQTT

Tương tự như MQTT, kết nối OPC-UA trong thực nghiệm này sẽ không sử dụng bảo mật, tài khoản hay mật khẩu, là một kết nối vô danh (anonymous).

3.1.3 Lựa chọn công nghệ hiện thực thực nghiệm

Trong thực nghiệm với MQTT, nhóm sử dụng 3 máy tính đóng các vai trò khác nhau: 1 máy đóng vai trò Client_1, 1 máy đóng vai trò Client_2 và một máy đóng vai trò Broker. Broker được nhóm sử dụng là broker từ **mosquitto**. Thứ tự khởi chạy sẽ là: server broker -> Client_2 -> Client_1.

Tương tự như MQTT, trong thực nghiệm với OPC-UA, nhóm sử dụng 3 máy tính đóng các vai trò khác nhau: 1 máy dùng để chạy **Client**, 1 máy dùng để chạy **DataCenter** và còn lại dùng để chạy **Server**

Cả 3 máy sẽ đều cùng kết nối vào một mạng wifi và mở các port trong tường lửa để cho phép các máy có thể kết nối với nhau.

Cả 2 thực nghiệm được hiện thực trên ngôn ngữ lập trình Python và dùng các thư viện có sẵn cho các giao thức đã được hiện thực trên Python, với giao thức MQTT nhóm sử dụng thư viện paho, với OPC-UA nhóm sử dụng thư viện opcua

3.1.3.1 Giới thiệu và cài đặt Mosquitto

Mosquitto là một Message broker mã nguồn mở hiện thực giao thức MQTT phiên bản 5.0, 3.1.1 và 3.1. Mosquitto nhẹ, đơn giản và phù hợp để sử dụng trên mọi thiết bị từ những chiếc máy tính nhúng năng lượng thấp tới các server hiệu năng cao.

Mosquitto có một test server tại địa chỉ test.mosquitto.org, hoặc người dùng có thể tải bản cài đặt của server mosquitto và cài tại chính máy tính cá

nhân / server của mình. Trong phần so sánh này, nhóm quyết định tải bản cài đặt mosquitto về máy để chạy server broker.

Trong hiện thực đo đạc RTT của giao thức MQTT, Moquitto được nhóm chọn để làm server broker.

3.1.3.1.1 Cài đặt và tùy chỉnh broker Mosquitto

Để cài đặt Mosquitto, ta có thể truy cập vào địa chỉ sau Mosquitto Download

The screenshot shows the Mosquitto download page. At the top, there are links for 'mosquitto' (with a logo), 'ECLIPSE FOUNDATION' (with a logo), and 'cedalo'. To the right are links for 'Home', 'Blog', 'Download', and 'Documentation'. A 'Download' button is prominently displayed. Below it, a 'Source' section lists 'mosquitto-2.0.15.tar.gz (GPG signature)' and 'Git source code repository (github.com)'. A note says 'Older downloads are available at <https://mosquitto.org/files/>'. Under 'Binary Installation', it says 'The binary packages listed below are supported by the Mosquitto project. In many cases Mosquitto is also available directly from official Linux/BSD distributions.' It then lists 'mosquitto-2.0.15-install-windows-x64.exe' (64-bit build, Windows Vista and up, built with Visual Studio Community 2019) and 'mosquitto-2.0.15-install-windows-x32.exe' (32-bit build, Windows Vista and up, built with Visual Studio Community 2019).

Figure 3.3: Tải về bản cài đặt của Mosquitto trên trang chủ

Tải về bản cài đặt 64bit dành cho Windows và tiến hành cài đặt như mặc định. Nhóm đã cài broker Mosquitto vào địa chỉ **C: \Program Files \mosquitto**



Figure 3.4: Tiến hành cài đặt moquitto

Sau khi cài đặt xong, ta cần chỉnh thêm một số tùy chỉnh để server có thể chạy không cần bảo mật (certificate) và có thể được phát hiện bởi các máy trong cùng subnet. Việc tùy chỉnh này sẽ được thực hiện trong file **mosquitto.conf** ở ngay trong thư mục cài đặt của mosquitto server.

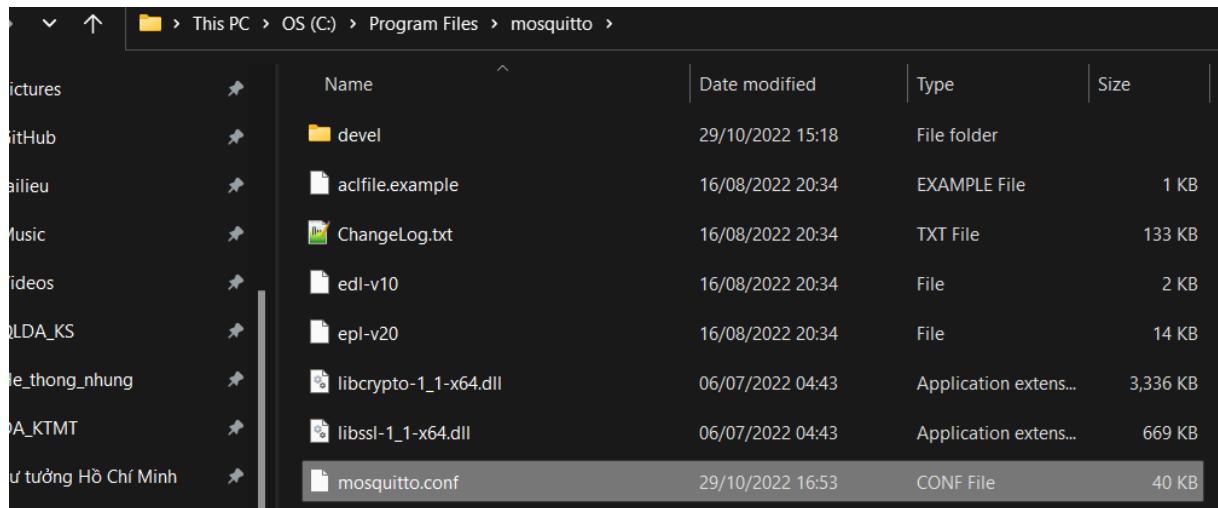


Figure 3.5: Vị trí của file mosquitto.conf trong thư mục cài đặt

Kích hoạt "allow anonymous" lên true cho phép người dùng có thể truy cập vào broker mà không cần tên người dùng và mật khẩu

```

524  # Boolean value that determines whether clients that connect
525  # without providing a username are allowed to connect. If set to
526  # false then a password file should be created (see the
527  # password_file option) to control authenticated client access.
528 #
529 # Defaults to false, unless there are no listeners defined in the configuration
530 # file, in which case it is set to true, but connections are only allowed from
531 # the local machine.
532 allow anonymous true

```

Figure 3.6: Chính sửa thông số "allow anonymous"

Mở listener gán với số port là 1883, host name của máy chạy server là NaruKiri để có thể chạy server broker và được nhận thấy bởi các máy chung subnet.

```

211 # =====
212 # Listeners
213 # =====
214
215 # Listen on a port/ip address combination. By using this variable
216 # multiple times, mosquitto can listen on more than one port. If
217 # this variable is used and neither bind_address nor port given,
218 # then the default listener will not be started.
219 # The port number to listen on must be given. Optionally, an ip
220 # address or host name may be supplied as a second argument. In
221 # this case, mosquitto will attempt to bind the listener to that
222 # address and so restrict access to the associated network and
223 # interface. By default, mosquitto will listen on all interfaces.
224 # Note that for a websockets listener it is not possible to bind to a host
225 # name.
226 #
227 # On systems that support Unix Domain Sockets, it is also possible
228 # to create a Unix socket rather than opening a TCP socket. In
229 # this case, the port number should be set to 0 and a unix socket
230 # path must be provided, e.g.
231 # listener 0 /tmp/mosquitto.sock
232 #
233 # listener port-number [ip address/host name/unix socket path]
234 listener 1883 NaruKiri

```

Figure 3.7: Chính sửa thông số listener

Sau khi chỉnh setting xong, để khởi chạy broker, ta vào command line của windows, di chuyển tới vị trí của folder cài đặt broker mqtt và nhập dòng lệnh:

```
C:\Program Files\mosquitto>mosquitto.exe -c mosquitto.conf -p 1883 -v
```

Dòng lệnh trên giúp khởi chạy broker mosquitto.exe với file config (-c) là mosquitto.conf, gắn với số port (-p) là 1883 và ghi log lại (-v). Khi chạy thành công, terminal sẽ hiện ra như sau:

```

1 C:\Program Files\mosquitto>mosquitto.exe -c mosquitto.conf -p 1883 -v
2 1669174826: mosquitto version 2.0.15 starting
3 1669174826: Config loaded from mosquitto.conf.
4 1669174826: Opening ipv6 listen socket on port 1883.
5 1669174826: Opening ipv4 listen socket on port 1883.
6 1669174826: Opening ipv6 listen socket on port 1883.
7 1669174826: Opening ipv6 listen socket on port 1883.
8 1669174826: Opening ipv6 listen socket on port 1883.
9 1669174826: Opening ipv6 listen socket on port 1883.
10 1669174826: Opening ipv4 listen socket on port 1883.
11 1669174826: Opening ipv4 listen socket on port 1883.
12 1669174826: mosquitto version 2.0.15 running

```

Server mosquitto MQTT đã khởi chạy và sẽ chấp nhận các kết nối tới server. Vì đã bỏ qua phần bảo mật (tài khoản, mật khẩu), Client chỉ cần biết được địa chỉ IP của server broker trong cùng subnet và số port (1883) là đủ để tạo kết nối tới server broker và hiện thực các thao tác publish,

subscribe vào các topic cần thiết.

3.1.3.2 Giới thiệu thư viện Paho-MQTT

Paho-MQTT là một thư viện hiện thực MQTT Client trên ngôn ngữ python, hiện thực phiên bản 5.0, 3.1.1 và 3.1 của protocol MQTT.

Thư viện này cung cấp một lớp client (client class) cho phép ứng dụng sử dụng có thể kết nối với MQTT broker để xuất bản (publish) gói tin, hoặc đăng ký (subscribe) vào các chủ đề (topics) và nhận các gói tin được xuất bản vào chủ đề đó. Thư viện cũng cung cấp một số hàm hỗ trợ giúp xuất bản một gói tin lên MQTT server broker rất nhanh chóng, dễ hiểu.

Thư viện Paho-MQTT hỗ trợ cho Python phiên bản 2.7.9 trở lên, hoặc phiên bản 3.6 trở lên. Paho là một phần của dự án Eclipse Foundation

Để cài đặt thư viện paho-mqtt trong Python, ta nhập dòng lệnh sau trong terminal của windows:

```
1 | pip install paho-mqtt
```

3.1.3.3 Giới thiệu thư viện opcua

Opcua là một thư viện mã nguồn mở, được phát triển trên ngôn ngữ lập trình Python, **opcua** cung cấp API cho các giao tiếp thông qua giao thức OPCUA, với **opcua** chúng ta có thể dễ dàng khởi tạo 1 server opc hoặc 1 client chỉ gói gọn trong vài dòng lệnh một cách dễ dàng.

Thư viện **opcua** hỗ trợ cho Python từ phiên bản 2.7.0 trở lên. Chi tiết về hướng dẫn sử dụng có thể tham khảo tại đây

Để cài đặt thư viện **opcua** trong Python, ta nhập dòng lệnh sau trong terminal của windows:

```
1 | pip install opcua
```

3.1.3.4 Giới thiệu và cài đặt Packet Sender

Packet Sender là một ứng dụng mã nguồn mở cho phép gửi và nhận các gói tin TCP, UDP và SSL (gói tin TCP được mã hóa) cũng như những yêu cầu (request) HTTP/HTTPS,... . Phần mềm chính thức hỗ trợ chạy trên nền tảng hệ điều hành Windows, Mac, và Linux (với Qt).

Trong thực nghiệm này, Packet sender được nhóm sử dụng để tạo một lưu lượng mạng ảo nhằm tăng tải của mạng, phục vụ cho thực nghiệm khi gửi các gói tin bằng giao thức MQTT hoặc OPC-UA trong điều kiện mạng tải cao.

3.1.3.4.1 Cài đặt và tùy chỉnh

Để cài đặt Packet sender, nhóm truy cập vào đường link sau và tải về bản cài đặt dành cho windows Packet sender download

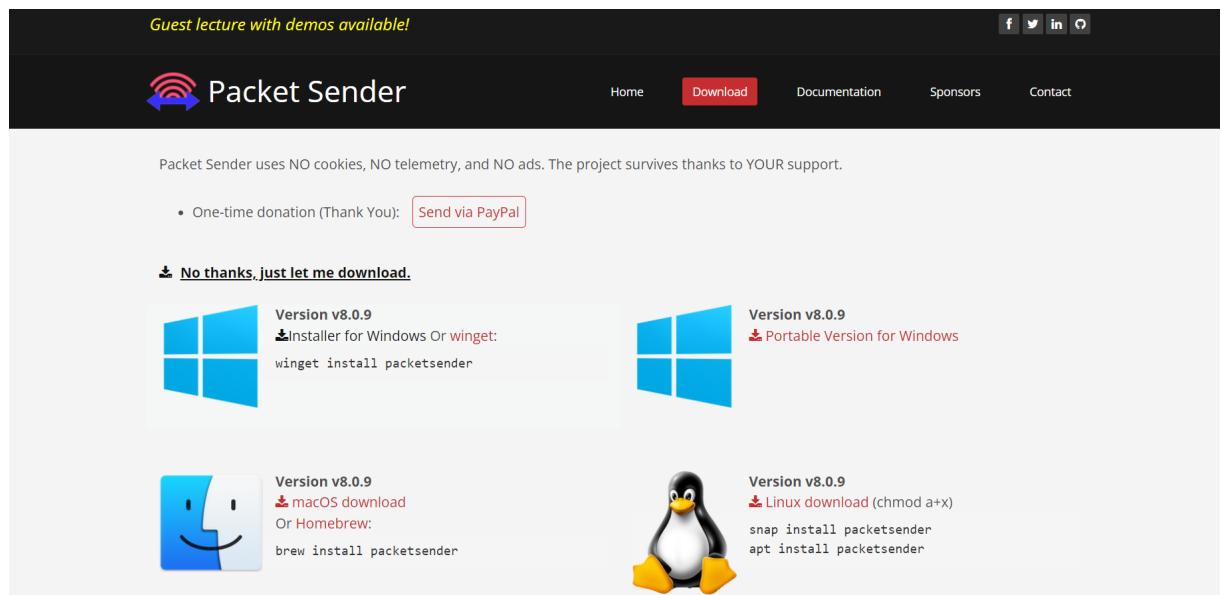


Figure 3.8: Tải về packet sender từ trang chủ

Sau khi tải về xong, tiến hành cài đặt phần mềm như mặc định

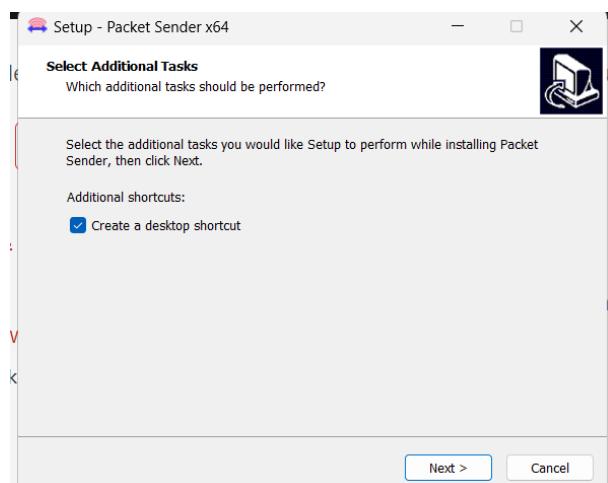


Figure 3.9: Tiến hành cài đặt

3.1. Phương pháp thực nghiệm

Khi cài đặt xong, ta sẽ khởi chạy phần mềm. Phần mềm ban đầu có giao diện như sau:

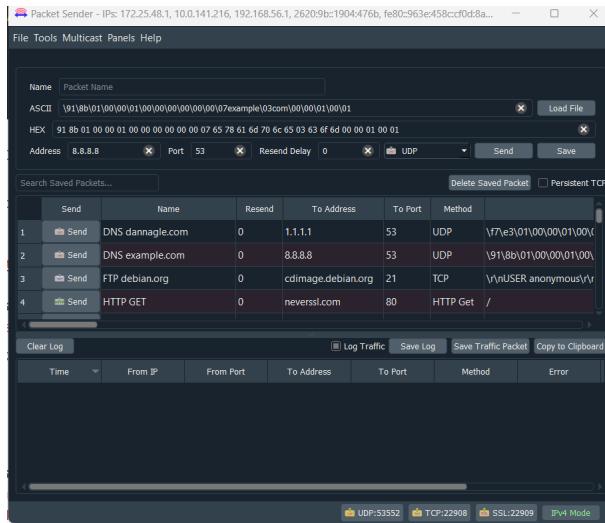


Figure 3.10: Giao diện packet sender

Để tạo được tải ảo cho mạng, nhóm làm theo các bước sau:

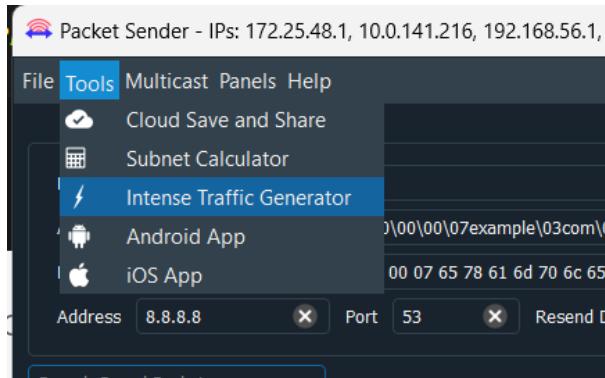


Figure 3.11: Chọn Intense traffic (tạo tải nặng). Chức năng này sẽ gửi một loạt nhiều gói tin giống nhau tới một địa chỉ và số port cho trước

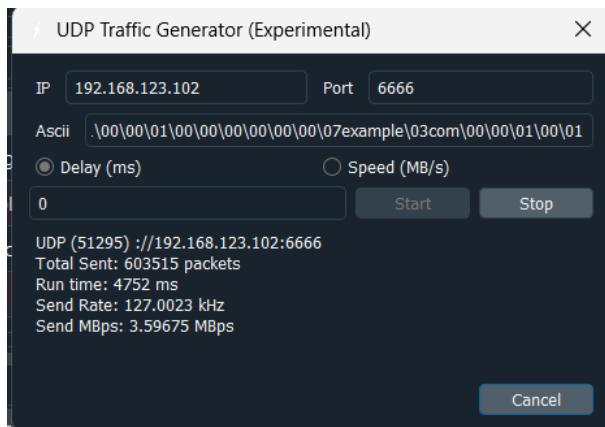


Figure 3.12: Điền IP và số port, nhấn start để bắt đầu tạo tải mạng bằng cách gửi hàng loạt các gói tin. Thông số thông kê được thể hiện trong hình.

Số port được chọn ở bước trên nên là số port ít được sử dụng, còn IP là địa chỉ của 2 máy tính còn lại trong 3 máy tính kết nối với nhau ở thực nghiệm này. Ở đây nhóm chọn số port là 6666, sau đó nhấn start để bắt đầu tạo các gói tin UDP. Vì gửi rất nhanh và nhiều, phương pháp này sẽ tạo 1 tải rất lớn cho mạng. Đây là phương pháp nhóm chọn để giả lập tình huống mạng bị quá tải.

3.2 Kết quả thực nghiệm và đánh giá

Nhóm đã tiến hành thực nghiệm cho hai giao thức trên trên 3 máy tính khác nhau trong cùng một mạng, lưu lại kết quả và sử dụng phần mềm Excel để trích xuất ra các đặc trưng của giá trị RTT cho từng độ dài gói tin và từng trạng thái của mạng. Ở phần kế tiếp, nhóm sẽ trình bày chi tiết kết quả có được sau thực nghiệm và những nhận xét, đánh giá về RTT cho từng giao thức, sau cùng là so sánh chúng với nhau sử dụng biểu đồ trực quan.

3.2.1 Thực nghiệm MQTT

Khi thực hiện thực nghiệm với giao thức MQTT trong điều kiện mạng bình thường và điều kiện mạng quá tải, cho kết quả trong bảng sau:

3.2. Kết quả thực nghiệm và đánh giá

Table 3.1: Kết quả thực nghiệm đo RTT của MQTT trong điều kiện mạng bình thường và tải cao

Kích thước	Điều kiện mạng	Trung bình	Trung vị	Độ lệch chuẩn	Giá trị nhỏ nhất	Giá trị lớn nhất
2b	Thường	0.1267	0.1058	0.0652	0.0471	1.1237
	Tải cao	0.3889	0.3490	0.0602	0.2245	7.3684
8b	Thường	0.1324	0.1128	0.0657	0.0468	0.7879
	Tải cao	0.3557	0.3331	0.0073	0.2144	1.0092
32b	Thường	0.1346	0.1238	0.0569	0.0309	0.6981
	Tải cao	0.3488	0.3252	0.0074	0.2305	1.0942
128b	Thường	0.1374	0.1219	0.0579	0.0419	0.4157
	Tải cao	0.3887	0.3500	0.0144	0.2145	1.0979
512b	Thường	0.1222	0.1057	0.0529	0.0469	0.4857
	Tải cao	0.4484	0.3985	0.0385	0.2174	3.3640
2Kb	Thường	0.1198	0.1037	0.0473	0.0449	0.4079
	Tải cao	0.4532	0.3531	0.2768	0.2245	7.1150
8Kb	Thường	0.1360	0.1183	0.0620	0.0476	0.5177
	Tải cao	0.4160	0.3570	0.0439	0.2383	2.8284
32Kb	Thường	0.1654	0.1456	0.0752	0.0529	0.8238
	Tải cao	0.7358	0.5465	0.2836	0.2813	4.7323
128Kb	Thường	0.2393	0.2145	0.1254	0.1084	1.6087
	Tải cao	2.0553	1.8895	0.8789	0.6513	9.9854

Theo như thực nghiệm, ta thấy có được một số nhận xét sau:

- Trong điều kiện bình thường, nhìn chung, khi tăng kích thước gói tin thì RTT trung bình cũng tăng lên. Độ lệch chuẩn với từng kích thước gói tin từ 2 byte tới 32 Kilobytes không chênh lệch quá nhiều, khoảng xấp xỉ 0.0604 giây (60.4ms). Tuy nhiên, khi kích thước gói tin đạt cao hơn - 128Kb, độ lệch chuẩn tăng lên gấp đôi (0.1254s)
- Khi tải cao, gói tin có độ dài payload từ 2b tới 128b, RTT trung bình cao gấp xấp xỉ 3 lần so với RTT trung bình ở điều kiện mạng bình

3.2. Kết quả thực nghiệm và đánh giá

thường. Từ 512b tới 8Kb, RTT cao gấp 4 lần. với 32Kb, RTT cao hơn 6 lần và với 128Kb, RTT cao hơn đến 9 lần. Có thể thấy, trong điều kiện tải nặng, khoảng cách về RTT trung bình so với điều kiện bình thường tăng nhanh khi kích thước gói tin tăng.

- RTT lớn nhất là 9.9854 giây khi gửi gói tin 128Kb trong điều kiện mạng tải cao. RTT nhỏ nhất là 0.0419 giây khi gửi gói tin 128b trong điều kiện mạng bình thường.
- Ngoài ra, trong quá trình thực nghiệm, dù gửi các gói tin lớn với tốc độ cao, MQTT không xảy ra hiện tượng mất gói.

Biểu đồ sau đây biểu diễn RTT trung bình khi sử dụng giao thức MQTT gửi các gói tin trong điều kiện mạng bình thường và mạng có tải cao:

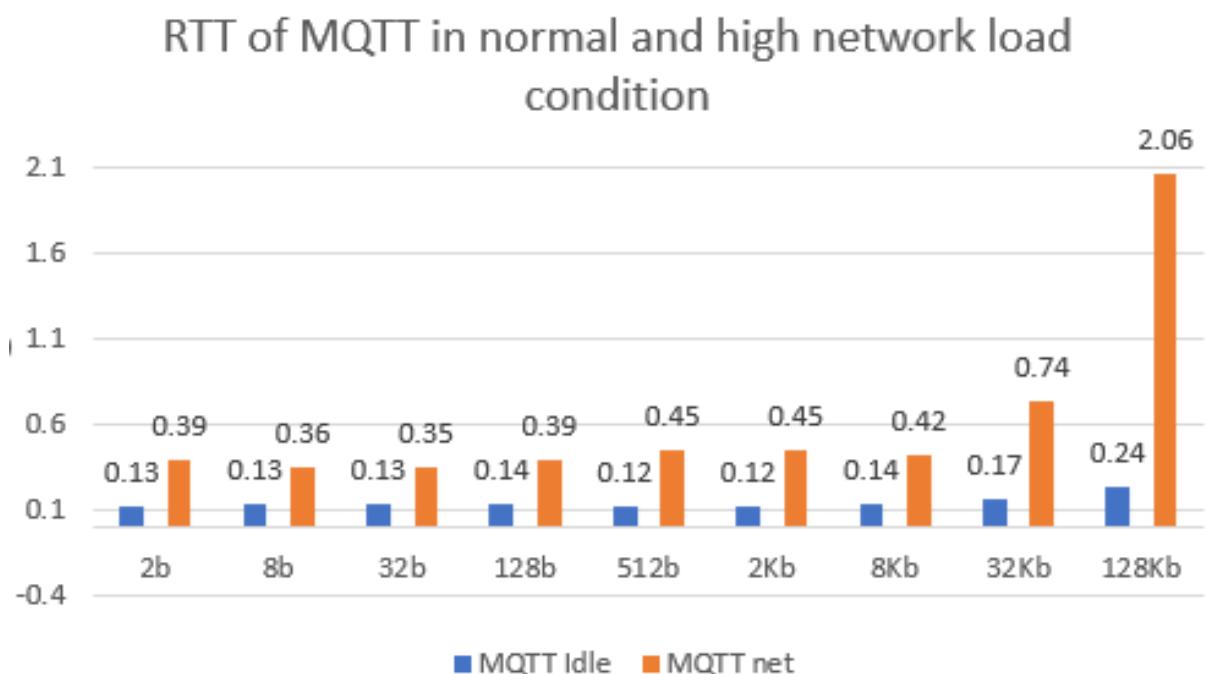


Figure 3.13: Biểu đồ RTT của MQTT trong điều kiện mạng bình thường và quá tải

3.2.2 Thực nghiệm OPC-UA

Khi thực hiện thực nghiệm với giao thức OPCUA trong các điều kiện mạng tương tự như với MQTT, cho kết quả trong bảng sau:

Table 3.2: Kết quả thực nghiệm đo RTT của OPCUA trong điều kiện mạng bình thường và tải cao

Kích thước	Điều kiện mạng	Trung bình	Trung vị	Độ lệch chuẩn	Giá trị nhỏ nhất	Giá trị lớn nhất
2b	Thường	0.085	0.079	0.0282	0.0381	0.3556
	Tải cao	0.3148	0.2739	0.1654	0.1446	3.2371
8b	Thường	0.0848	0.0798	0.0258	0.0402	0.3536
	Tải cao	0.3167	0.2833	0.1945	0.0582	3.211
32b	Thường	0.0832	0.0792	0.0217	0.0378	0.2048
	Tải cao	0.3363	0.3036	0.1263	0.1421	1.6016
128b	Thường	0.0829	0.0790	0.0221	0.0461	0.2206
	Tải cao	0.4461	0.4061	0.1909	0.1843	2.2632
512b	Thường	0.0853	0.0796	0.0274	0.0434	0.3194
	Tải cao	0.4555	0.4224	0.1604	0.2081	2.2834
2Kb	Thường	0.0875	0.0831	0.0271	0.0431	0.4167
	Tải cao	0.3341	0.2926	0.1501	0.1528	2.7507
8Kb	Thường	0.0884	0.0841	0.0291	0.0453	0.5967
	Tải cao	0.3185	0.2851	0.1777	0.1617	4.5577
32Kb	Thường	0.0959	0.0892	0.0370	0.0389	0.6254
	Tải cao	0.3370	0.3077	0.1157	0.1630	1.4815
128Kb	Thường	0.1555	0.1411	0.0586	0.1007	0.6887
	Tải cao	0.8955	0.6886	0.6410	0.2284	4.0007

Nhìn vào kết quả từ thực nghiệm, ta có một số đánh giá như sau:

- Trường hợp mạng bình thường, cũng tương tự như MQTT, khi tăng kích thước gói tin thì RTT trung bình cũng tăng lên. Độ lệch chuẩn với từng kích thước gói tin từ 2 byte tới 128 Kilobytes không chênh lệch quá

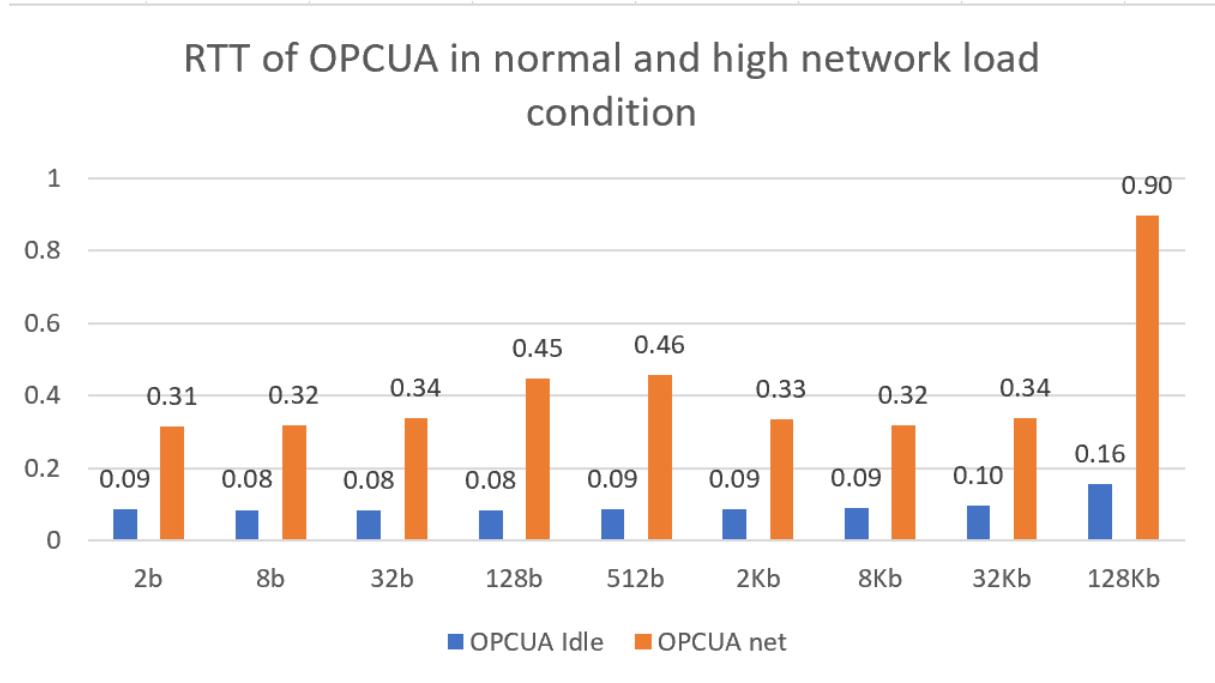


Figure 3.14: Biểu đồ RTT của OPCUA trong điều kiện mạng bình thường và quá tải

nhiều, khoảng xấp xỉ 0.031 giây (30.1ms). Điều này cho ta thấy được sự ổn định khá cao của OPCUA trong quá trình truyền tải dữ liệu trong điều kiện mạng ổn định.

- Từ thực nghiệm với MQTT, ta có thể thấy RTT của OPCUA trong các điều kiện thực nghiệm đều thấp hơn so với RTT của MQTT
- Trong quá trình thực nghiệm, khi gửi gói tin với tốc độ cao, OPCUA thỉnh thoảng xảy ra hiện tượng mất gói tin, đây là điểm trừ khi so sánh tính toàn vẹn gói tin so với MQTT

3.2.3 So sánh kết quả thực nghiệm MQTT và OPC-UA

Sử dụng kết quả từ 2 thực nghiệm với 2 giao thức, nhóm có một số kết quả so sánh sau đây:

Có thể thấy, trong thực nghiệm này, giao thức OPC-UA có RTT tốt hơn MQTT trong hầu hết mọi kích thước gói tin, cả trong điều kiện mạng bình thường và khi mạng có tải cao. Khi kích thước gói tin tăng thì OPC-UA càng thể hiện khả năng gửi nhận nhanh hơn so với MQTT, tuy nhiên, nếu nhanh về tốc độ thì OPC-UA lại bị nhược điểm là đôi lúc xảy ra mất gói khi kích

3.2. Kết quả thực nghiệm và đánh giá

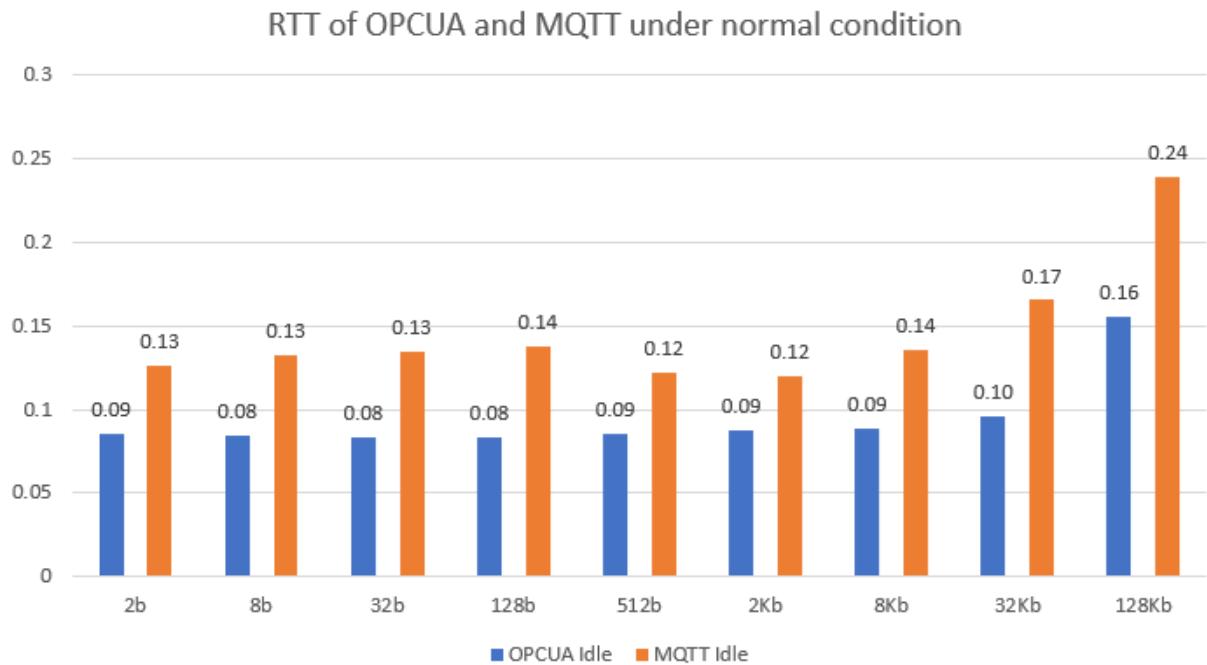


Figure 3.15: Biểu đồ RTT trung bình của OPC-UA và MQTT trong điều kiện mạng bình thường

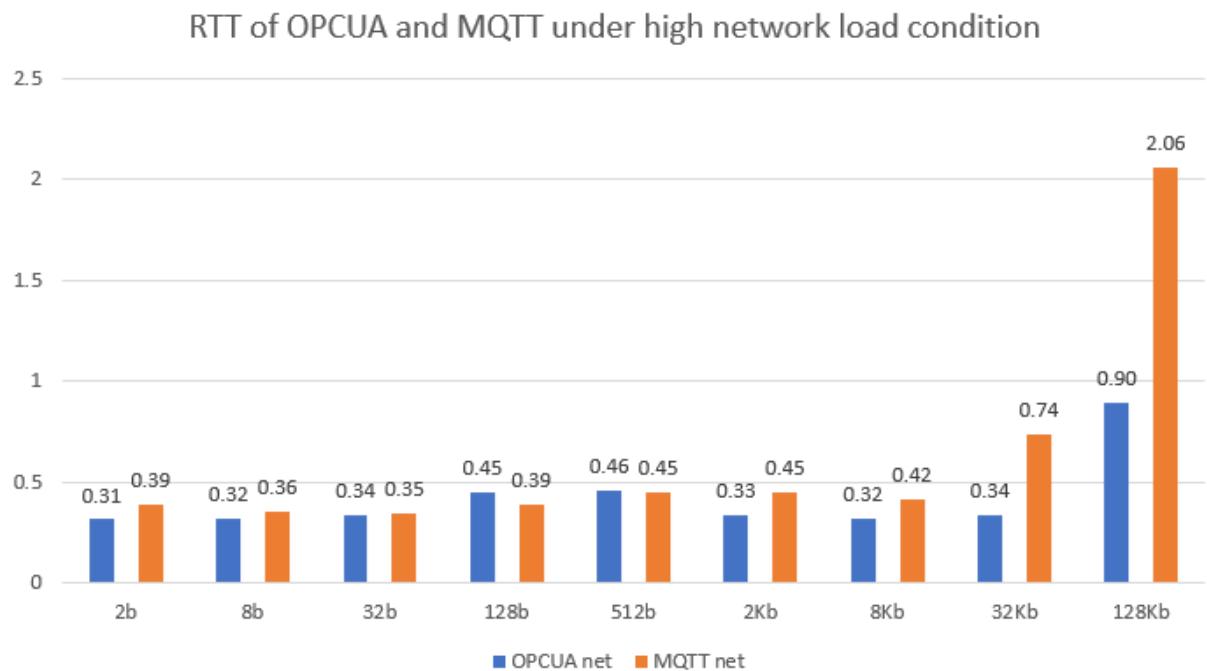


Figure 3.16: Biểu đồ RTT trung bình của OPC-UA và MQTT trong điều kiện mạng tải cao

thuộc dữ liệu lớn trong điều kiện mạng tải cao.

3.2. Kết quả thực nghiệm và đánh giá

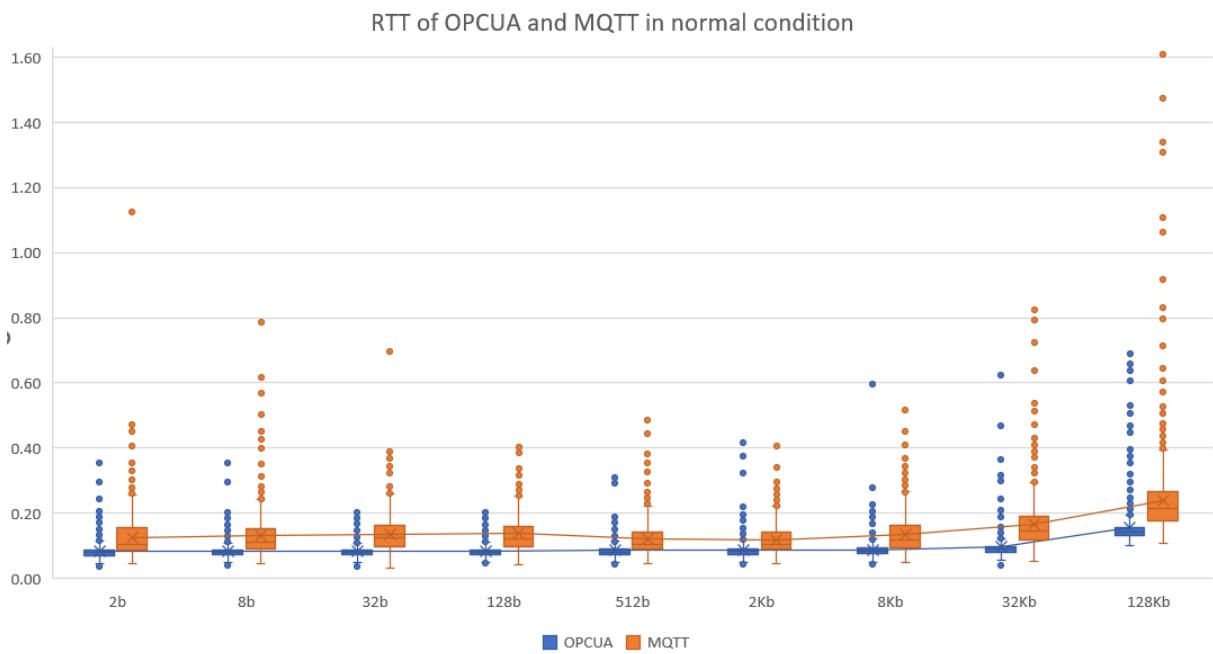


Figure 3.17: Biểu đồ Boxplot của OPC-UA và MQTT trong điều kiện mạng bình thường

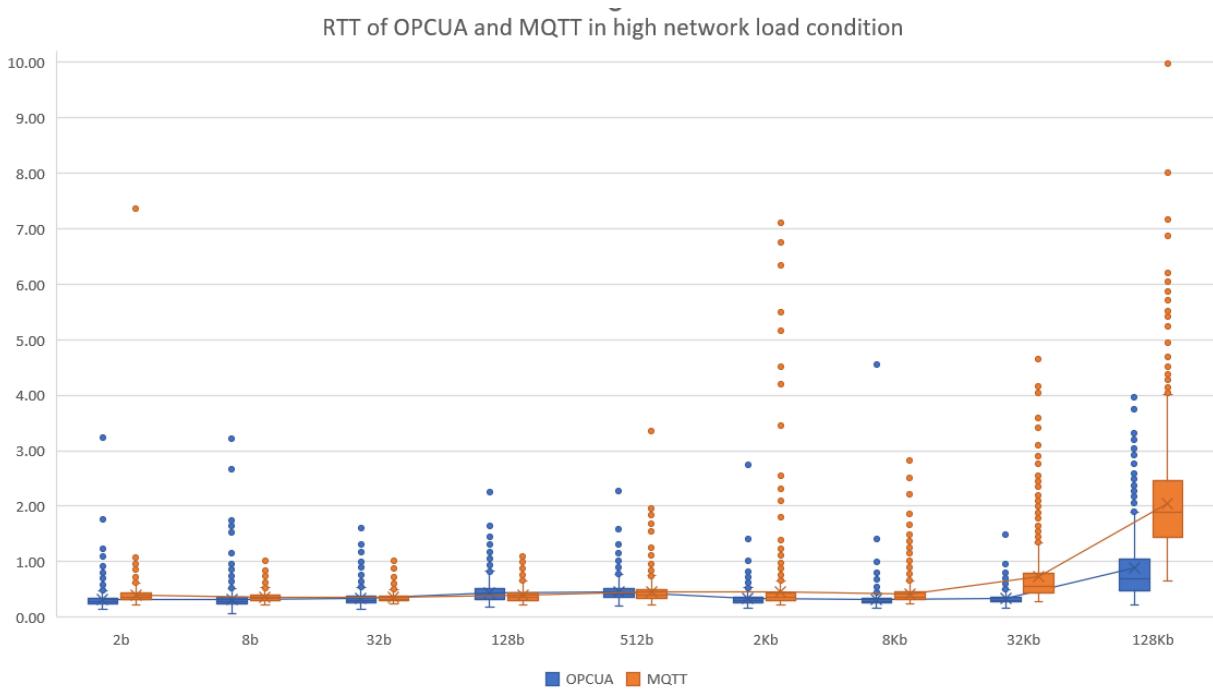


Figure 3.18: Biểu đồ Boxplot của OPC-UA và MQTT trong điều kiện mạng tải cao

Tiếp đến, biểu đồ boxplot của OPCUA và MQTT được vẽ nêu. Ta có một số nhận xét sau:

- Trong điều kiện mạng bình thường, như nhận xét trước, OPC-UA có RTT trung bình tốt hơn so với MQTT. Điều này dẫn theo các thông số liên quan như Mode, hay trung vị của MQTT cũng lớn hơn so với

OPC-UA. Thêm đó, các điểm dữ liệu của OPC-UA tập trung gần trung bình hơn (độ lệch chuẩn nhỏ hơn). Còn với MQTT, các điểm dữ liệu phân tán xa hơn so với giá trị trung bình và cũng có nhiều outlier hơn, cho thấy sự bất ổn định của giao thức khi sử dụng cho các ứng dụng ưu tiên về tốc độ.

- Trong điều kiện có tải cao, ở các gói tin có kích thước nhỏ (từ 2b tới 512b), OPC-UA tỏ ra thua kém hơn so với MQTT khi các điểm dữ liệu trở nên phân tán nhiều hơn và nhiều outlier hơn, tuy nhiên giá trị trung bình của RTT vẫn nhỏ hơn so với MQTT. Khi gói tin có kích thước lớn hơn (2Kb trở lên), OPC-UA chiếm lợi thế và các điểm dữ liệu ổn định hơn. Các outlier của OPC-UA không vượt quá 4 giây, do trong hiện thực OPC-UA, timeout cho một request chỉ là 4 giây, quá 4 giây OPC-UA sẽ bỏ gói tin đi. MQTT thoải mái hơn về vấn đề timeout vì chỉ dùng QoS 0, do đó các điểm outlier cũng xa hơn và phân tán hơn, đỉnh điểm ở gói tin 128Kb, RTT lớn nhất của MQTT lên tới 10 giây. MQTT thể hiện khả năng đảm bảo truyền dữ liệu tốt cả khi gói tin lớn và mạng tải cao, nhưng đánh đổi về tốc độ khi thời gian delay khá lớn. OPC-UA có yêu cầu khắt khe về tốc độ và thời gian phản hồi, do đó gửi nhanh hơn MQTT trong các trường hợp khác nhau về độ lớn gói tin, tuy nhiên, do giới hạn của hiện thực OPC-UA trong python, OPC-UA sẽ bỏ gói tin khi request phản hồi lâu hơn 4 giây. **tại sao lại bỏ gói tin?**

CHAPTER 4

ĐIỀU KHIỂN CÁNH TAY ROBOT DỰA TRÊN OPC-UA

Trong bối cảnh của nhà máy, đặc biệt là nhà máy thông minh, cánh tay robot công nghiệp (Industrial robot arm) đóng một vai trò không thể thiếu trong việc tự động hóa quy trình sản xuất sản phẩm. Cánh tay robot công nghiệp những năm gần đây đã trở thành ‘trợ thủ đắc lực’ trong các nhà máy sản xuất. Cánh tay robot có vai trò quan trọng, thay thế con người trong các hoạt động sản xuất mang tính lặp lại và trong các môi trường làm việc rủi ro cao. Ở chương này, đề tài sẽ tiến hành tìm hiểu về cánh tay robot công nghiệp, cấu tạo và ứng dụng của nó, sau đó hiện thực việc điều khiển hoạt động của cánh tay sử dụng giao thức OPC-UA với các thiết bị, kiến trúc và công nghệ phù hợp.

4.1 Cánh tay robot công nghiệp

Như đã trình bày ở phần trên, cánh tay robot là một trong những thiết bị quan trọng, hỗ trợ các thao tác tự động hóa trong bối cảnh nhà máy, đặc biệt là nhà máy thông minh. Trong phần này, ta sẽ tìm hiểu chi tiết hơn về cánh tay robot trong công nghiệp, thành phần cấu tạo thường thấy của một cánh tay robot trong công nghiệp và ứng dụng của nó.

4.1.1 Khái niệm

Cánh tay robot công nghiệp (Cánh tay cơ khí) là một thiết bị được lập trình để hoạt động tương tự như cánh tay con người, với các khớp chuyển

động theo một trục dọc và có thể xoay theo các hướng nhất định.

Hầu hết các robot công nghiệp được chế tạo và lập trình để thực hiện các nhiệm vụ cụ thể. Kích cỡ của chúng có thể nhỏ để cầm nắm để thực hiện các thao tác phức tạp, hoặc đủ lớn để nâng nhắc khối lượng nặng.

Cánh tay robot lần đầu tiên được ứng dụng trong ngành công nghiệp ô tô với công việc hàn lặp đi lặp lại. Đến nay, nhờ sự phát triển của công nghệ nó đã góp mặt trong nhiều lĩnh vực: cơ khí, y tế, thực phẩm, hàng không vũ trụ, ...

4.1.2 Cấu tạo của cánh tay robot

Cánh tay robot được cấu tạo bởi 3 phần:

- **Tay máy (Phần cứng):** Phần tay máy thường được chế tạo từ gang và thép – những vật liệu có độ bền cao. Tay máy được mô phỏng như cánh tay người, có phần cổ tay, cẳng tay, khuỷu tay, vai và chân đế. Các cánh tay robot trong công nghiệp có từ 4 – 6 khớp nối với 6 bậc tự do tương đương với 6 cách di chuyển khác nhau.
- **Hệ thống điều khiển:** Chịu trách nhiệm điều khiển sau khi nhận và xử lý các tín hiệu từ bên ngoài. Sau khi nhận thông tin, bộ phận điều khiển sẽ phát thông tin để động cơ dịch chuyển theo yêu cầu, hình thành một chuỗi động học. Chức năng của hệ thống điều khiển cũng được phân cấp từ đơn giản như đến chức năng phức tạp.
- **Hệ thống quản lý và vận hành:** Đây là phần mềm được cài đặt trên máy tính, nơi các kỹ thuật viên lập trình các thao tác của robot. Hệ thống quản lý và điều khiển sẽ được kết nối với nhau để truyền và nhận tin.

4.1.3 Ứng dụng của cánh tay robot trong công nghiệp

Trong các nhà máy thông minh, cánh tay robot được ứng dụng cho nhiều mục đích khác nhau. Với tốc độ cao, độ chính xác lớn, độ rung thấp, cánh tay robot không chỉ có thể cải thiện đáng kể hoạt động sản xuất, giúp lắp

ráp chính xác cao mà còn hoạt động ổn định khi nó di chuyển nhanh chóng đến một địa điểm cụ thể.

Bên cạnh đó, cánh tay robot cũng có thể được tùy chỉnh để làm các tác vụ hỗ trợ định lượng trong các dây chuyền sản xuất và góp phần nâng cao năng suất nhờ các báo cáo dữ liệu được trích xuất trong thời gian thực, thông qua tích hợp cảm biến, thuật toán tự học, đo lường và tích hợp với hệ thống ERP, MES trong sản xuất,... Một vài ứng dụng phổ biến của cánh tay robot phổ như sau:

- Xếp chồng hàng hóa lên các pallet
- Hàn cơ khí
- Kiểm tra chất lượng
- Gắp, thả sản phẩm

Cánh ra robot ra đời đã đóng góp nhiều ý nghĩa, đem lại hiệu quả cao trong sản xuất thông minh. Giúp các nhà sản xuất gia tăng tốc độ sản xuất, cải thiện chất lượng sản phẩm, giảm chi phí vận hành và ưu điểm lớn của cánh tay robot là có thể thay thế con người trong các môi trường có rủi ro cao như hàn, cơ khí, hóa chất, ... và làm việc trong môi trường nhiệt độ dao động từ -40 độ đến 120 độ. Từ đó, phòng tránh các tai nạn lao động và giúp tiết kiệm chi phí nhân công.

4.2 Hiện thực điều khiển cánh tay robot với giao thức OPCUA

Ở phần trên, ta thấy được giao thức OPC-UA có lợi thế hơn hẳn giao thức MQTT về tốc độ gửi tin thông qua RTT, đáp ứng được nhu cầu đồng bộ dữ liệu khắc khe trong môi trường công nghiệp. Ta cũng tìm hiểu thêm về một trong những thiết bị quan trọng hỗ trợ công việc tự động hóa trong nhà máy là các cánh tay robot (hay cánh tay cơ khí), với thiết kế đa dạng và các chức năng đặc trưng, có thể được lập trình để thực thi những chuỗi hành động phức tạp. Ở phần này, nhóm sẽ tiến hành kết hợp giao thức chuẩn công nghiệp OPC-UA để thực hiện điều khiển cánh tay robot thông qua ứng dụng

trên điện thoại. Ứng dụng sẽ được xây dựng bằng phần mềm Unity và chạy trên hệ điều hành android. Đối với cánh tay robot sẽ được điều khiển bằng mạch lập trình nhúng Mini là Yolobit kết hợp với một server OPC-UA. Kiến trúc điều khiển của hiện thực sẽ được trình bày chi tiết trong phần sau.

4.2.1 Giới thiệu thiết bị

Để điều khiển được hành động của cánh tay robot, nhóm sử dụng mạch lập trình nhúng Yolo:bit, một chiếc mạch "đa năng" phù hợp cho các ứng dụng STEM, kết nối cánh tay robot với 3 bậc tự do được điều khiển bằng servo. Phần tiếp theo trình bày chi tiết về mạch lập trình nhúng Yolo:bit và giới thiệu về các thành phần của cánh tay robot được sử dụng trong đề tài.

4.2.1.1 Yolo:bit

Yolo:bit là một mạch lập trình mini phù hợp với dạy và học lập trình STEM. Mạch có thể lập trình được bằng ngôn ngữ Python hoặc các block lập trình kéo thả được cung cấp bởi nhà sản xuất.



Figure 4.1: Hình ảnh mạch Yolo:bit

Yolo:bit hỗ trợ kết nối qua USB / Bluetooth, với Yolo:bit chúng ta có thể thỏa sức sáng tạo và xây dựng nhiều dự án vui nhộn, từ điều khiển đèn LED chớp tắt cơ bản cho đến các dự án IoT phức tạp như làm robot thông

minh, xây dựng hệ thống tự động tưới cây, nhà thông minh,... hoặc nhiều ứng dụng khác để giải quyết vấn đề trong cuộc sống.

Trong quá trình hiện thực điều khiển cánh tay robot, Yolo:bit cùng với bộ mở rộng kèm theo sẽ được dùng như một vi điều khiển, trực tiếp gửi lệnh điều khiển đến cánh tay.

4.2.1.2 Cánh tay robot

Thiết bị được điều khiển, là mô phỏng của một cánh tay robot công nghiệp với 4 bậc tự do có cấu tạo như sau:

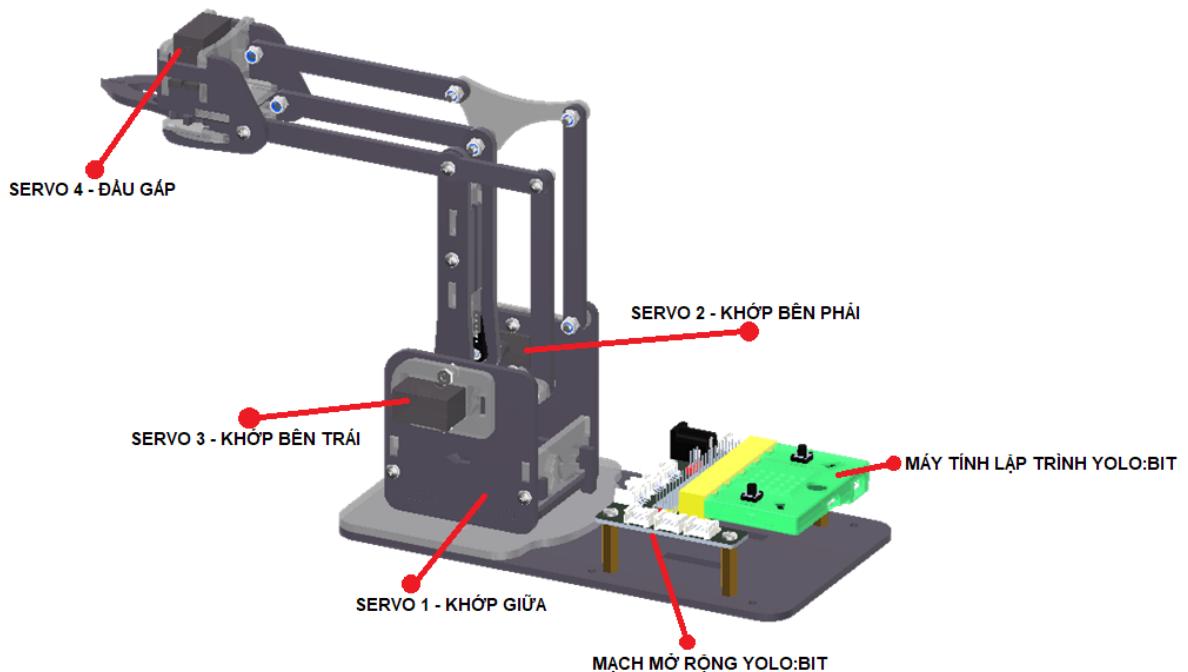


Figure 4.2: Hình ảnh mô phỏng kiến trúc của cánh tay cùng Yolo:bit

Chức năng hoạt động chính của từng servo như sau:

- Servo 1 - Khớp giữa: có chức năng xoay trái-phải. Có thể điều khiển bằng cách cho servo có thể quay từ 0 độ (bên trái) tới 180 độ (bên phải)
- Servo 2 - Khớp bên phải: có chức năng nâng lên hạ xuống. Có thể điều khiển bằng cách cho servo có thể quay từ 100 độ (lên trên) tới 180 độ (xuống dưới)
- Servo 3 - Khớp bên trái: có chức năng chuyển động tới lui. Có thể điều khiển bằng cách cho servo có thể quay từ 0 độ (lui về) tới 100 độ (tiến lên trước)

- Servo 4 - Đầu gấp: Có chức năng đóng mở đầu gấp. Có thể điều khiển bằng cách cho servo có thể quay từ 80 độ (đóng đầu gấp) tối 180 độ (Mở đầu gấp)

4.2.2 Kiến trúc hệ thống

Để nâng cao khả năng mở rộng của đồ án, nhóm đã đề xuất một Kiến trúc của cho hệ thống, kiến trúc được chia ra thành 3 thành phần chính như sau:

- Mobile App: App điều khiển cánh tay được phát triển trên thiết bị di động, điều khiển cánh tay hoạt động với các chức năng xoay cơ bản thông qua giao diện thân thiện với người dùng.
- DataCenter: Một module trung gian được tích hợp vào hệ thống sử dụng trong việc chuyển nhận, xử lý lưu trữ dữ liệu về gốc xoay của từng Servo, việc thêm vào 1 module trung gian sẽ cho phép ta mở rộng việc điều khiển và kết nối tới nhiều thiết bị khác nhau hơn là chỉ gói gọn trong mô hình 1 client kết nối 1 server. Các dữ liệu về gốc xoay của cánh tay sẽ được lưu lại tại DataCenter và sẽ được dùng để phát triển các chức năng khác mở rộng hơn trong giai đoạn luận văn. DataCenter sẽ vừa đóng vai trò là một server OPC-UA để ứng dụng di động truy cập vào, vừa là một Client OPC-UA để điều khiển hoặc nhận dữ liệu từ server OPC-UA tại module Control
- Control: module Control bao gồm hiện thực server OPC-UA và Yolo:bit nối với cánh tay, là nơi tiếp nhận dữ liệu cuối cùng. Khi dữ liệu về góc xoay cánh tay trong server OPC-UA thay đổi do yêu cầu từ DataCenter, server sẽ xử lý và gửi lệnh điều khiển tương ứng xuống mạch Yolo:Bit thông qua giao tiếp UART, khi nhận được dữ liệu Yolo:bit sẽ thực hiện xoay các servo tương ứng với góc xoay mà nó nhận được từ module Control thông qua UART. Dưới đây là sơ đồ kiến trúc của hệ thống:

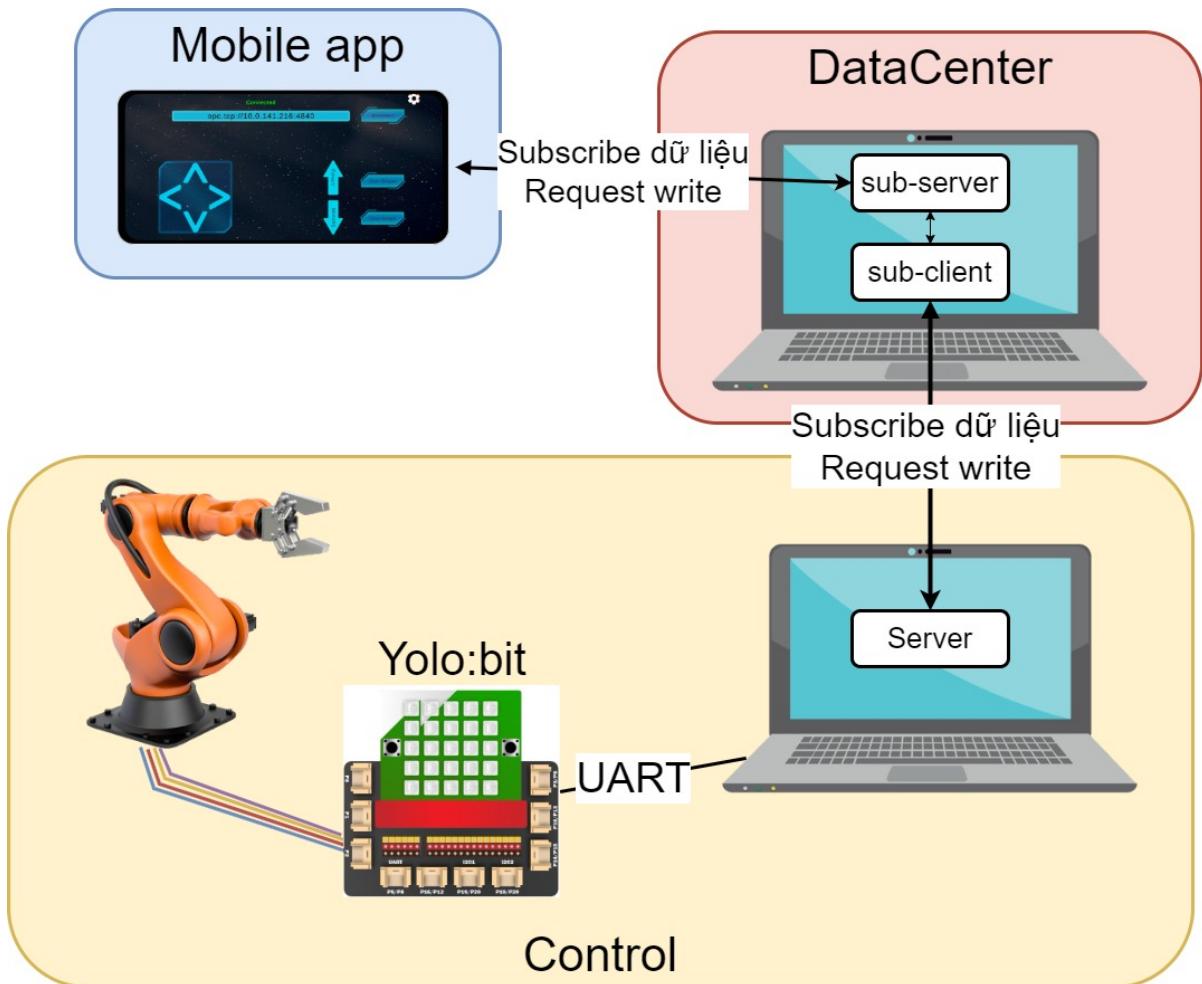


Figure 4.3: Kiến trúc các module giao tiếp qua OPCUA

4.2.3 Hiện thực module Control

Module Control như đã trình bày ở trên, sẽ gồm 2 phần: Server OPC-UA để tiếp nhận các yêu cầu điều khiển từ DataCenter, và mạch nhúng Yolo:bit để xuất các xung điều khiển servo, qua đó điều khiển chuyển động của cánh tay. Giữa Server OPC-UA và mạch Yolo:bit kết nối thông qua giao tiếp UART. Để dễ dàng trong việc truyền nhận dữ liệu UART, nhóm sử dụng thêm chip USB to UART để hỗ trợ truyền dữ liệu uart từ server OPC-UA trên máy tính và mạch Yolo:bit. Phần tiếp theo mô tả chi tiết cách kết nối của các thiết bị và phần mã nguồn hiện thực server OPC-UA cho module Control.

4.2.3.1 Thiết lập cánh tay và Yolo:bit

Cánh tay robot và mạch điều khiển Yolo:bit được thiết lập với các bước như sau:

1. Cắm dây servo theo các thứ tự sau: P8 - Servo khớp giữa, P9 - Servo khớp phải, P10 - Servo khớp trái, P11 - Servo đầu gấp

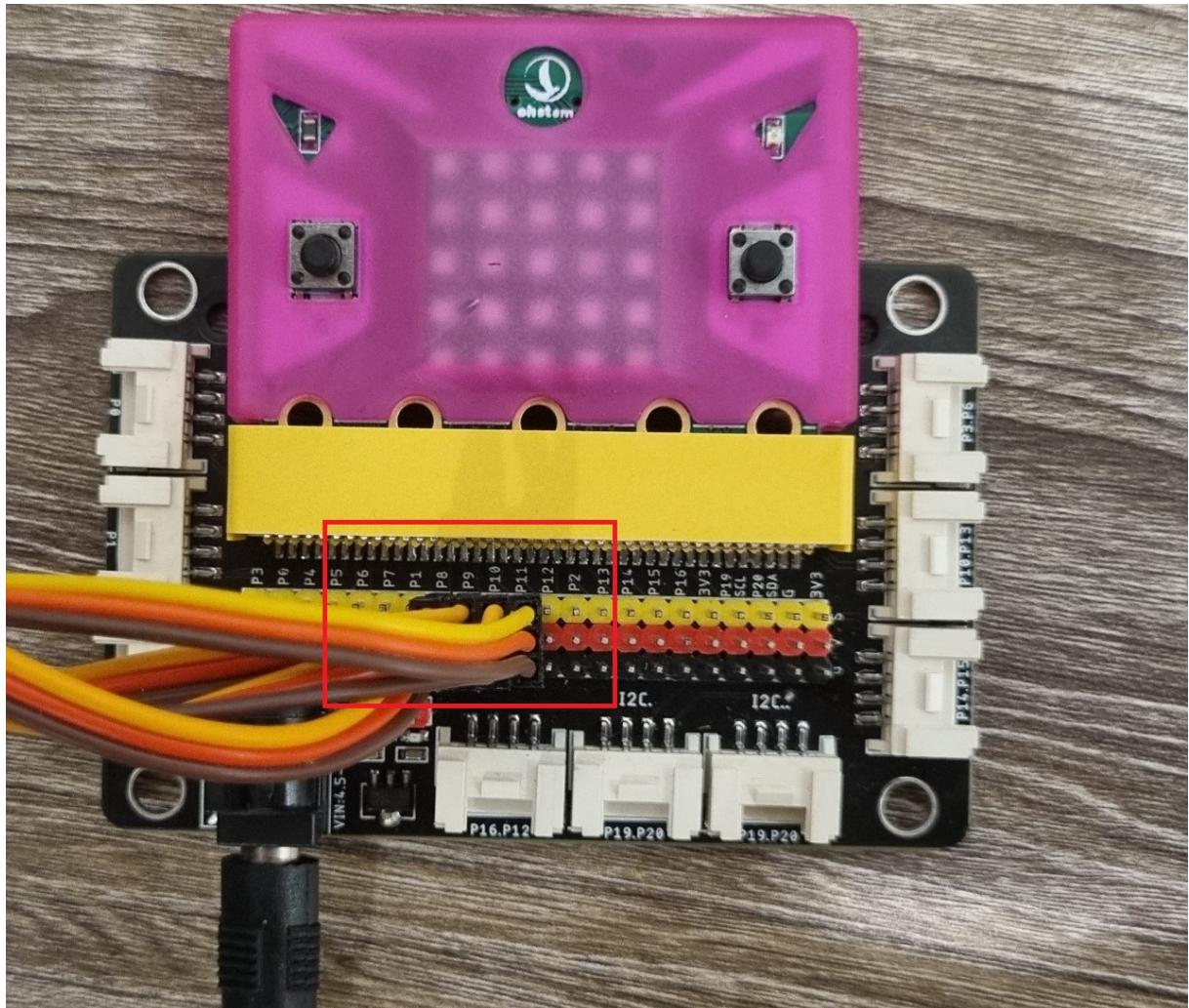


Figure 4.4: Cắm chấn Servo

4.2. Hiện thực điều khiển cánh tay robot với giao thức OPCUA

2. Cắm cổng DC cấp nguồn vào mạch mở rộng. Nguồn cấp tối thiểu là 500mA và tối đa là 2A để Robot hoạt động ổn định

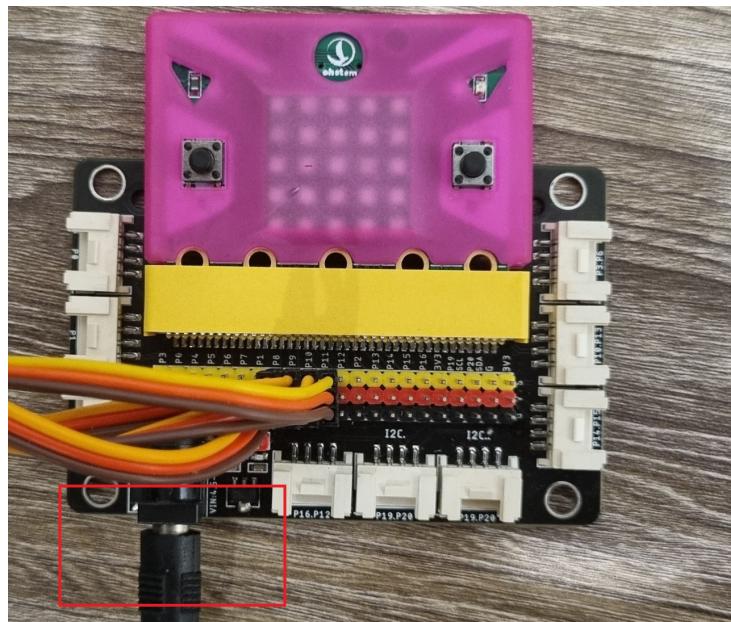


Figure 4.5: Cáp nguồn mạch mở rộng

3. Cắm dây thiết lập UART vào chân mở rộng P14-P15

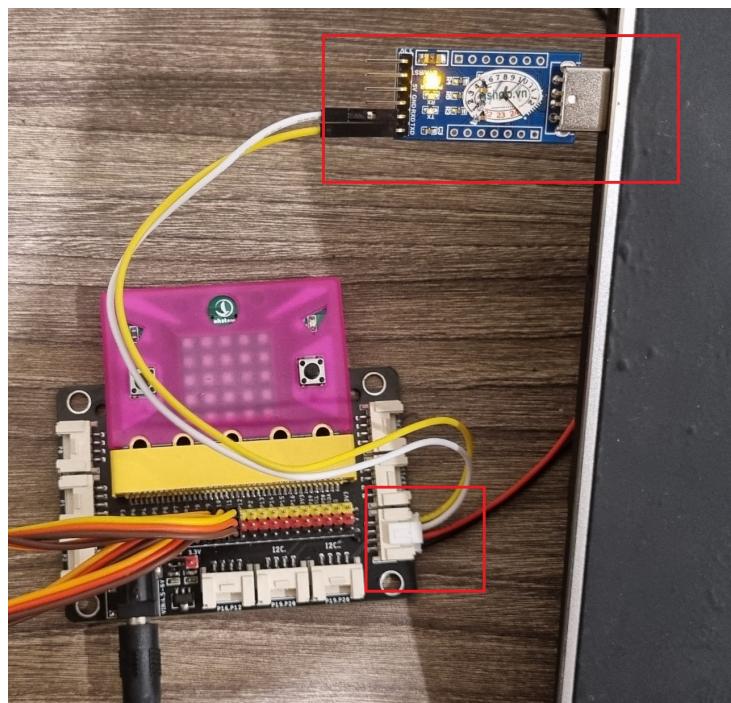


Figure 4.6: Kết nối UART vào chân P14-P15

4. Kết nối với laptop và nạp code điều khiển cho Yolo:Bit

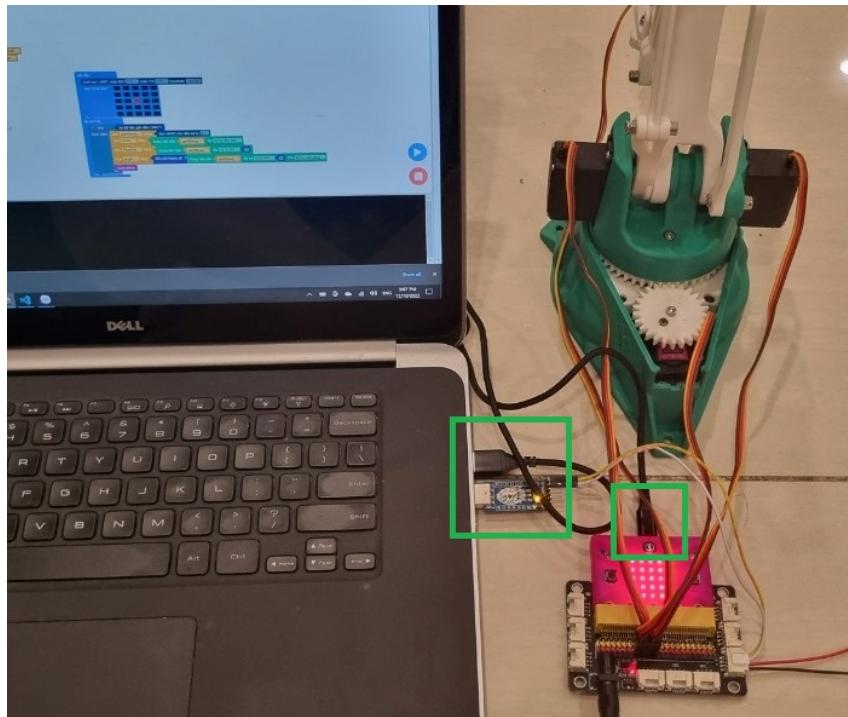


Figure 4.7: Cắm chắn Servo

4.2.3.2 Cấu hình giao tiếp OPCUA

Sau khi hoàn thành thiết lập cánh tay cùng Yolo:bit và kết nối chúng vào thiết bị để giao tiếp thông qua OPCUA (laptop hoặc raspberry pi), ta sẽ tiến hành cấu hình 1 server OPCUA cho thiết bị giao tiếp để kết nối và nhận dữ liệu điều khiển từ module DataCenter

1. Import các thư viện cần thiết và tạo ra server OPCUA cùng các node dữ liệu cần thiết cho việc điều khiển các khớp quay của cánh tay. Server được tạo ra cùng với 4 Node quan trọng:

- Node M (ns=2;i=2) - Là giá trị góc quay của servo giữa, điều khiển việc xoay trái - xoay phải
- Node R (ns=2;i=3) - Là giá trị góc quay của servo bên phải, điều khiển việc cánh tay hướng về trước - lùi về sau
- Node L (ns=2;i=4) - Là giá trị góc quay của servo bên trái, điều khiển việc cánh tay di chuyển lên - xuống
- Node F (ns=2;i=3) - Là giá trị góc quay của servo điều khiển cần gấp, dùng để mở - đóng cần gấp.

4.2. Hiện thực điều khiển cánh tay robot với giao thức OPCUA

```
1  from opcua import Server
2  import serial
3  import socket
4
5  # Init url by get current IP in network
6  hostname=socket.gethostname()
7  IPAddr=socket.gethostbyname(hostname)
8  url = "opc.tcp://" +IPAddr+":4840"
9
10 # Init server, add namespace, create node object
11 server= Server()
12 server.set_endpoint(url)
13 name = "OPCUA_SERVER"
14 addspace = server.register_namespace(name)
15 node = server.get_objects_node()
16 station = node.add_object(addspace,"Station")
17
18 # Init node for servo and set them writeable
19 MidServo = station.add_variable(addspace,"M",90)
20 RightServo = station.add_variable(addspace,"R",90)
21 LeftServo = station.add_variable(addspace,"L",90)
22 FoldServo = station.add_variable(addspace,"F",90)
23 MidServo.set_writable()
24 RightServo.set_writable()
25 LeftServo.set_writable()
26 FoldServo.set_writable()
27
28 # Run server
29 server.start()
30 print("Center started at {}".format(url))
31
```

- Đặt ra các thông số giới hạn về gốc quay cho các servo và khởi tạo serial để gửi lệnh điều khiển xuống Yolo:bit thông qua giao tiếp UART

```
1  # Range of each Servo
2  MIN_M = 0
3  MAX_M = 180
4
5  MIN_R = 100
6  MAX_R = 180
```

4.2. Hiện thực điều khiển cánh tay robot với giao thức OPCUA

```
7
8     MIN_L = 0
9     MAX_L = 100
10
11    MIN_F = 80
12    MAX_F = 180
13
14    # Init serial
15    # Yolo bit link: https://app.ohstem.vn/#!/share/yolobit/2
16    IFgJjpmNrN8mFni18MfmenU2vG
17
18    try:
19        ser = serial.Serial(
20            port='COM3',
21            baudrate=115200,
22            parity='N',
23            stopbits=1,
24            bytesize=8
25        )
26    except:
27        ser = None
```

3. Ý tưởng của việc gửi lệnh điều khiển xuống Yolo:bit khi nhận được dữ liệu mới từ DataCenter là Server sẽ tự tạo ra các subscription vào chính các node chứa dữ liệu của các khớp quay mà ta đã tạo ra ở phía trên, sau đó khi dữ liệu có sự thay đổi ở các node thì tạo ra một event và event này sẽ được handle thông qua một object có tên là **SubHandler**, với sự thay đổi của mỗi node servo, **SubHandler** sẽ lưu lại các request đó vào một list cmdList thông qua một method có tên là **datachange_notification**

```
1    # Create class SubHandler
2    class SubHandler(object):
3        def __init__(self, ser, preM, preR, preL, preF):
4            self.ser = ser
5            self.cmdList = list()
6
7
8        def datachange_notification(self, node, val, data):
9            print(node, val)
10           nameServo = node.get_browse_name().to_string()[2]
11           value = int(val)
```

4.2. Hiện thực điều khiển cánh tay robot với giao thức OPCUA

```
12         self.cmdList.append([nameServo,value])
13
14
15     def event_notification(self, event):
16         print("Python: New event", event)
17
18
19 # Create subscribe channel
20 handler = SubHandler(ser,preM, preR, preL, preF)
21 subscribe = server.create_subscription(1, handler)
22 subscribe.subscribe_data_change(MidServo)
23 subscribe.subscribe_data_change(RightServo)
24 subscribe.subscribe_data_change(LeftServo)
25 subscribe.subscribe_data_change(FoldServo)
```

4. Tiếp đến, tận dụng những request được lưu trong object Handler, ta sẽ lấy từng request ra và xử lý chúng thông qua hàm `sendUartCommand`. Hàm này sẽ xem xét từng request và gửi lệnh uart phù hợp xuống cho mạch Yolo:bit xử lý. Hàm này sẽ được gọi bởi 1 hàm while True ở cuối chương trình, với chu kỳ 10ms một lần.

```
1     def sendUartCommand(ser: serial.Serial, SubHandler: SubHandler):
2         :
3
4         if len(SubHandler.cmdList) > 0:
5             tempItem = SubHandler.cmdList.pop()
6             nameServo = tempItem[0]
7             val = tempItem[1]
8             ser.write((f"{nameServo}:"+str(val) + "#").encode())
9             print((f"{nameServo}:"+str(val) + "#"))
10
11
12     while True:
13         sendUartCommand(ser, handler)
14         time.sleep(0.01)
```

Full code hiện thực Server có thể tìm thấy tại đây

4.2.4 Hiện thực module DataCenter

DataCenter là module trung chuyển dữ liệu và là nơi lưu trữ lại dữ liệu về gốc xoay của các servo. Thông qua DataCenter ta có thể dùng các dữ liệu

4.2. Hiện thực điều khiển cánh tay robot với giao thức OPCUA

đó để phát triển mở rộng hệ thống cho ra nhiều chức năng có tính ứng dụng cao hơn. Tại DataCenter chúng ta sẽ hiện thực 2 thành phần chính là 1 sub client và 1 sub server, sub client sẽ là thành phần để kết nối tới Server ở module Control và sub server là thành phần kết nối mà Mobile App sẽ kết nối tới khi điều khiển cánh tay.

Sau đây là các bước hiện thực DataCenter:

1. Import các thư viện cần thiết và khởi tạo 1 sub client kết nối tới Server trong module Control

```
1  from opcua import Server
2  from opcua import Client
3
4  # Setup client
5  ServerUrl = URL_SERVER_CONTROL
6  client = Client(ServerUrl)
7  client.connect()
8  print('Sub_client connected to sub_server')
9  print('=====')  
10
11 M = client.get_node("ns=2;i=2")
12 R = client.get_node("ns=2;i=3")
13 L = client.get_node("ns=2;i=4")
14 F = client.get_node("ns=2;i=5")
15
```

2. Khởi tạo sub server (hay nhóm còn gọi là Datacenter), nơi mà Mobile App sẽ kết nối tới và gửi dữ liệu điều khiển, sub server sẽ nhận event thay đổi dữ liệu cũng thông qua cách tự tạo ra các subscription vào chính các node dữ liệu của nó, khi dữ liệu thay đổi SubHandler sẽ gọi tới các node ở sub client và set value cho node tương ứng. Ở sub server này cũng khởi tạo 4 node quan trọng là **M**, **L**, **R**, **F** như server.

```
1
2  # Init server , add namespace , create node object
3  server= Server()
4  server.set_endpoint(url)
5  name = "OPCUA_SUB_SERVER"
6  addspace = server.register_namespace(name)
7
8  # Init server , add namespace , create node object
```

4.2. Hiện thực điều khiển cánh tay robot với giao thức OPCUA

```
9     node = server.get_objects_node()
10    station = node.add_object(addspace, "Station")
11
12    # Init node for servo and set them writeable
13    MidServo = station.add_variable(addspace, "M", 90)
14    RightServo = station.add_variable(addspace, "R", 90)
15    LeftServo = station.add_variable(addspace, "L", 90)
16    FoldServo = station.add_variable(addspace, "F", 90)
17
18    MidServo.set_writable()
19    RightServo.set_writable()
20    LeftServo.set_writable()
21    FoldServo.set_writable()
22
23    #Run sub server
24    server.start()
25    print("Sub_server started at {}".format(url))
```

3. Tạo object Subhandle và xử lí khi có sự thay đổi dữ liệu ở sub server

```
1  class SubHandler(object):
2      def __init__(self, M, R, L, F):
3          self.M = M
4          self.R = R
5          self.L = L
6          self.F = F
7
8      def datachange_notification(self, node, val, data):
9          nameServo = node.get_browse_name().to_string()[2]
10         if nameServo == 'M':
11             self.M.set_value(val)
12         if nameServo == 'R':
13             self.R.set_value(val)
14         if nameServo == 'L':
15             self.L.set_value(val)
16         if nameServo == 'F':
17             self.F.set_value(val)
18
19     handler = SubHandler(M, R, L, F)
20     subscribe = server.create_subscription(1, handler)
21     subscribe.subscribe_data_change(MidServo)
```

```
22     subscribe.subscribe_data_change(RightServo)  
23     subscribe.subscribe_data_change(LeftServo)  
24     subscribe.subscribe_data_change(FoldServo)
```

Full code của DataCenter có thể tìm thấy tại đây

4.2.5 Hiện thực Mobile App điều khiển thiết bị

Trong phần này, nhóm hiện thực ứng dụng điện thoại để điều khiển cánh tay sử dụng giao thức OPC-UA bằng Unity, đồng thời hướng dẫn cách thức để cài đặt thư viện Opc.UaFx.Client trong môi trường Unity để phục vụ cho việc tạo Client OPC-UA.

4.2.5.1 Giới thiệu về Unity



Figure 4.8: Unity

Unity là một game engine (một software framework được thiết kế chủ yếu cho việc lập trình game, gồm các thư viện liên quan và phần mềm hỗ trợ) đa nền tảng, được xây dựng bởi Unity Technologies, được phát hành lần đầu vào năm 2005 tại Hội nghị lập trình viên quốc tế của Apple như một game engine dành cho Mac OS X. Unity dần mở rộng để hỗ trợ một lượng lớn các nền tảng máy tính bàn, di động, máy chơi game và thực tế ảo. Nó nổi tiếng

được sử dụng để xây dựng các ứng dụng chơi game trên di động cho iOS và Android, được xem là dễ sử dụng cho các lập trình viên mới, đồng thời nổi tiếng trong việc xây dựng các indie game (independent video game - những trò chơi được tạo bởi cá nhân hoặc các nhóm phát triển nhỏ không có nhiều kinh phí hoặc được hỗ trợ bởi các nhà phát hành trò chơi lớn).

Game engine Unity có thể được sử dụng để tạo các game ba chiều (3D) hoặc game hai chiều (2D), cũng như những mô phỏng tương tác. Unity đã được sử dụng cho các mảng khác ngoài làm game như làm phim, làm xe hơi, kiến trúc, kỹ thuật, xây dựng...

Unity có một cộng đồng phát triển rộng lớn với vô số các hướng dẫn, với mục tiêu khiến cho việc làm game và các ứng dụng liên quan trở nên dễ dàng hơn với tất cả mọi người, cộng thêm Asset Store - gồm rất nhiều các tài nguyên về đồ họa, môi trường, âm thanh, hiệu ứng... do chính người dùng đóng góp, khiến nó là một công cụ phát triển game và ứng dụng rất tốt cho người mới bắt đầu.

Ngoài ra, Unity dù được đội ngũ phát triển tạo nên bằng ngôn ngữ C++, nhưng nó lại cho phép người sử dụng dùng sức mạnh của ngôn ngữ C# để lập trình, tận dụng thế mạnh về OOP của C# và sự lớn mạnh của nó.

4.2.5.2 Cài đặt Unity Hub và Unity Editor

Vì Unity có nhiều phiên bản phát triển khác nhau theo các năm, nên cần thiết có một phần mềm để quản lý các phiên bản - Unity Hub. Khi cài đặt Unity, trước nhất người dùng cần tải về Unity Hub và lựa chọn phiên bản phù hợp - Unity Editor.

Để tải Unity Hub, ta vào trang web sau: Unity Hub download

Tiến hành cài đặt như bình thường. Sau đó, mở Unity Hub, ta cần cài đặt Unity Editor với phiên bản phù hợp để bắt đầu lập trình. Ta vào tab Install → Install Editor.

Ta nên ưu tiên các phiên bản Unity Editor có chữ LTS (Long time support) vì đó là các Unity Editor chuẩn, ổn định và được hỗ trợ lâu dài bởi Unity. Nhóm chọn Unity Editor phiên bản 2021.3.14f1. Ở đây, do nhóm đã install từ trước nên nút không cần Install lại. Trong trường hợp Install mới, khi nhấn nút, Unity sẽ hiện ra bảng các module sẽ cài đặt thêm cho Unity Editor. Vì để xây dựng ứng dụng android và mong đợi khả năng mở rộng sau này, ta sẽ

4.2. Hiện thực điều khiển cánh tay robot với giao thức OPCUA

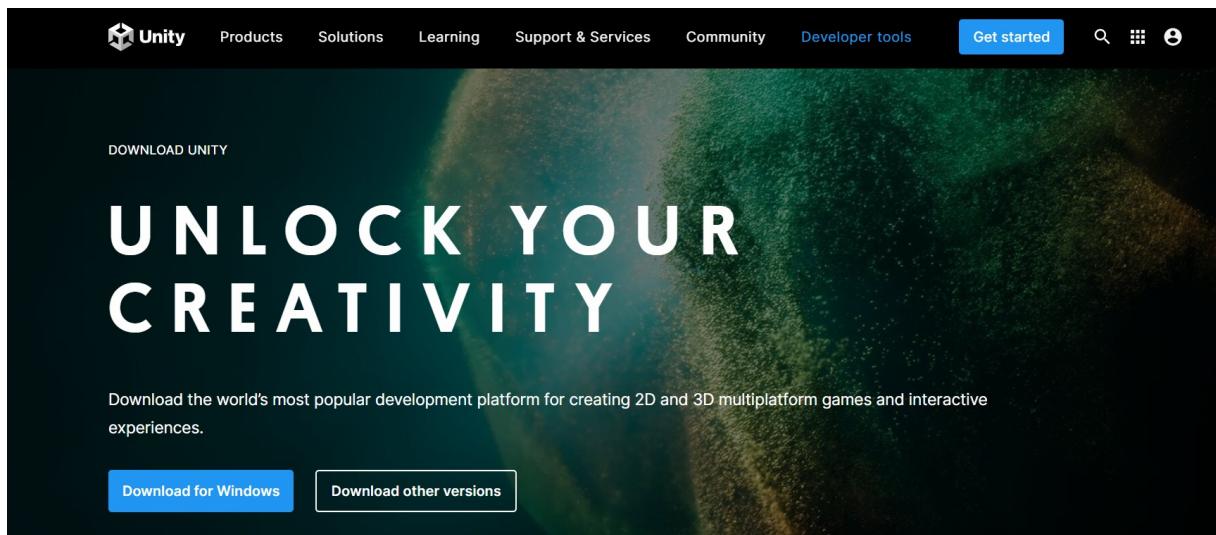


Figure 4.9: Trang web cài đặt Unity

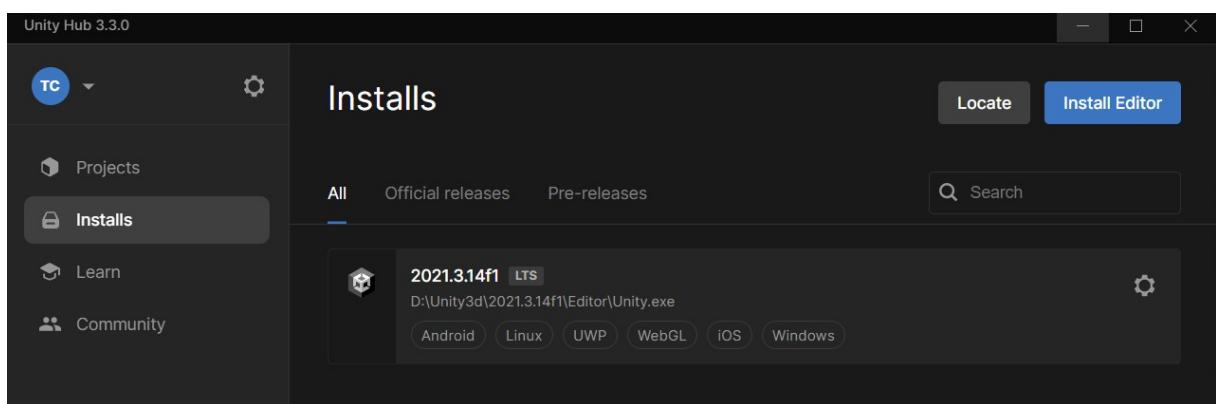


Figure 4.10: Nhấn Nút Install Editor

chọn các module sau:

- Dev tool: Microsoft Visual Studio Community 2019 (cần cài đặt nếu chưa có, nếu đã có sẵn Visual Studio phiên bản mới hoặc tốt hơn, sẽ không cần phải cài module này)
- Android Build Support (gồm các module để hỗ trợ lập trình trên Android)
- Linux Build Support (IL2CPP)
- Linux Build Support (Mono)
- Windows Build Support (IL2CPP)

Sau khi chọn xong Module, ta nhấn nút Install, Unity sẽ tiến hành tải các module và Unity Editor về. Tùy theo số lượng module mà ta thêm vào, dung

4.2. Hiện thực điều khiển cánh tay robot với giao thức OPCUA

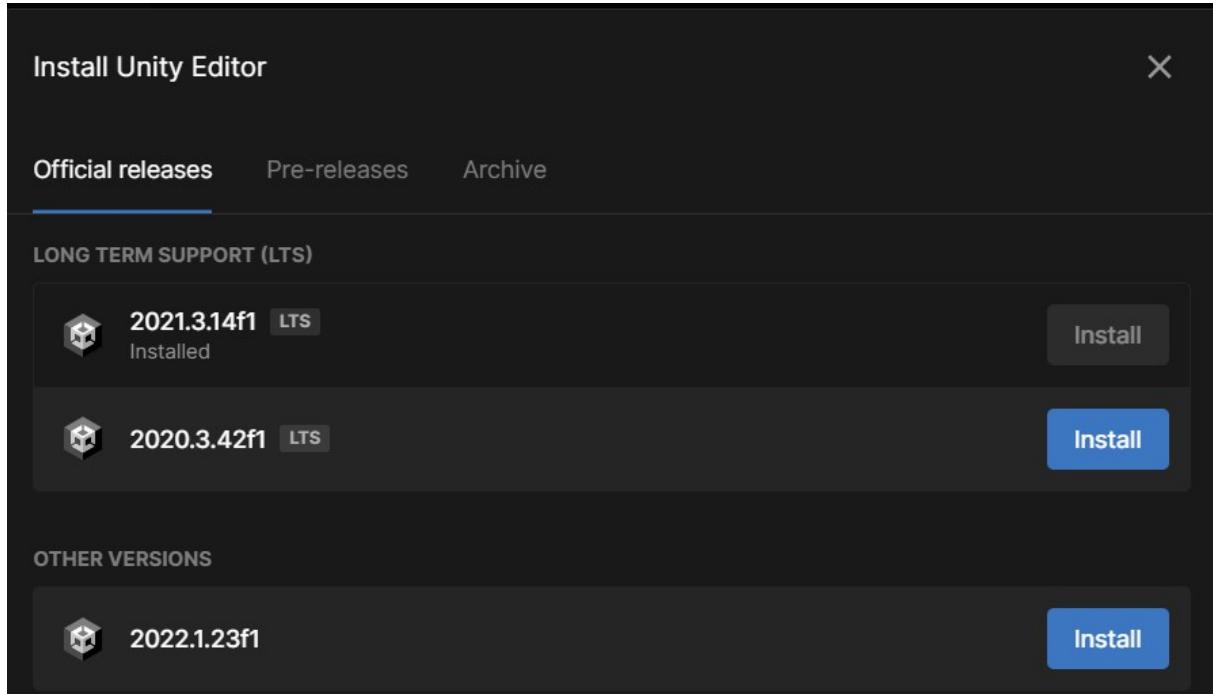


Figure 4.11: Chọn phiên bản Unity Editor

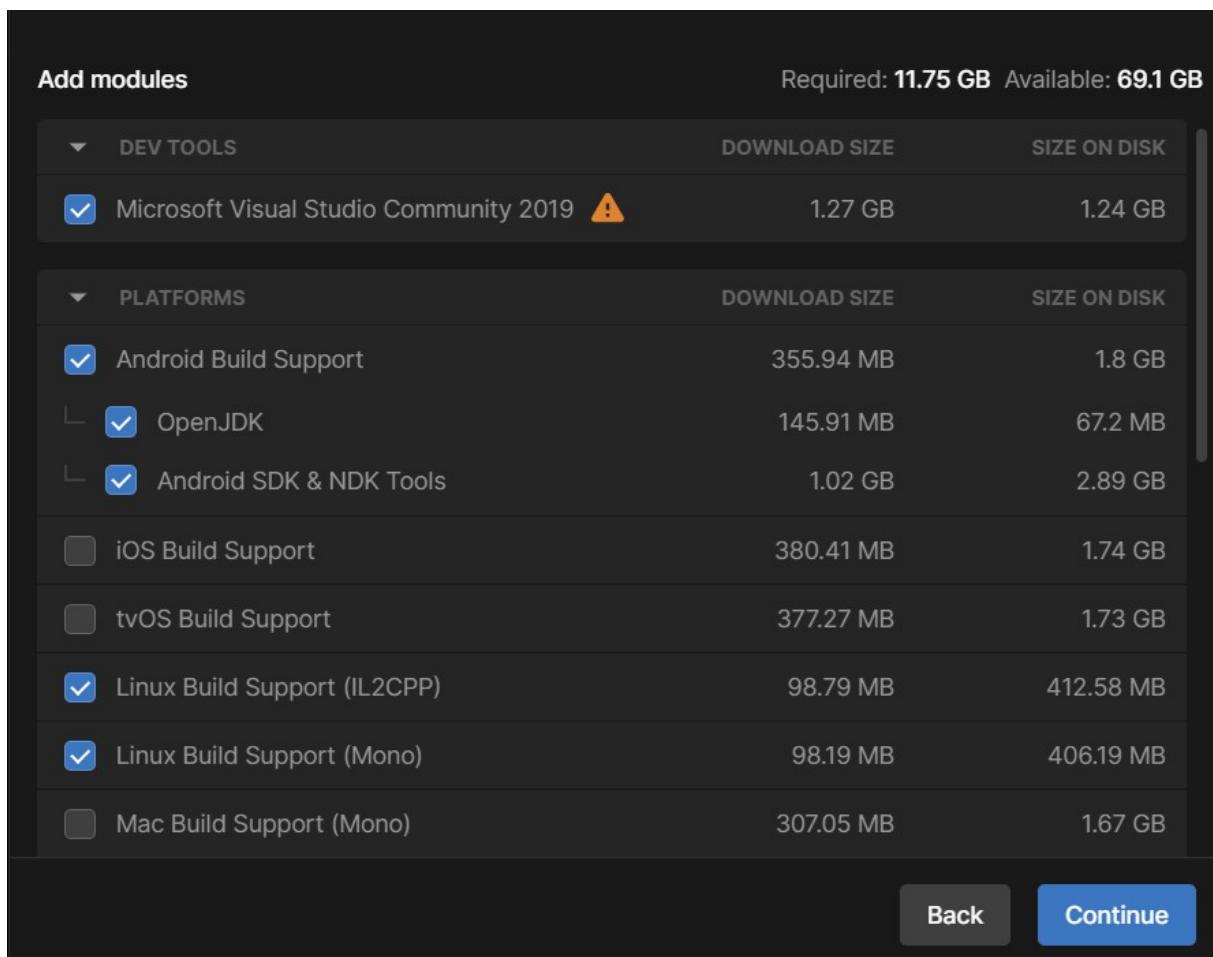


Figure 4.12: Chọn module cho Unity

4.2. Hiện thực điều khiển cánh tay robot với giao thức OPCUA

lượng tải xuống sẽ khá lớn. Để tránh làm đầy dung lượng ổ đĩa chính của hệ điều hành (ổ C), ta có thể thay đổi vị trí tải về của Unity Editor. Từ giao diện chính, nhấn vào hình bánh răng để vào Preferences, từ đó thay đổi vị trí Install location nếu cần thiết. Các Editor sẽ được tự động tải về và cài đặt tại vị trí Installs Location

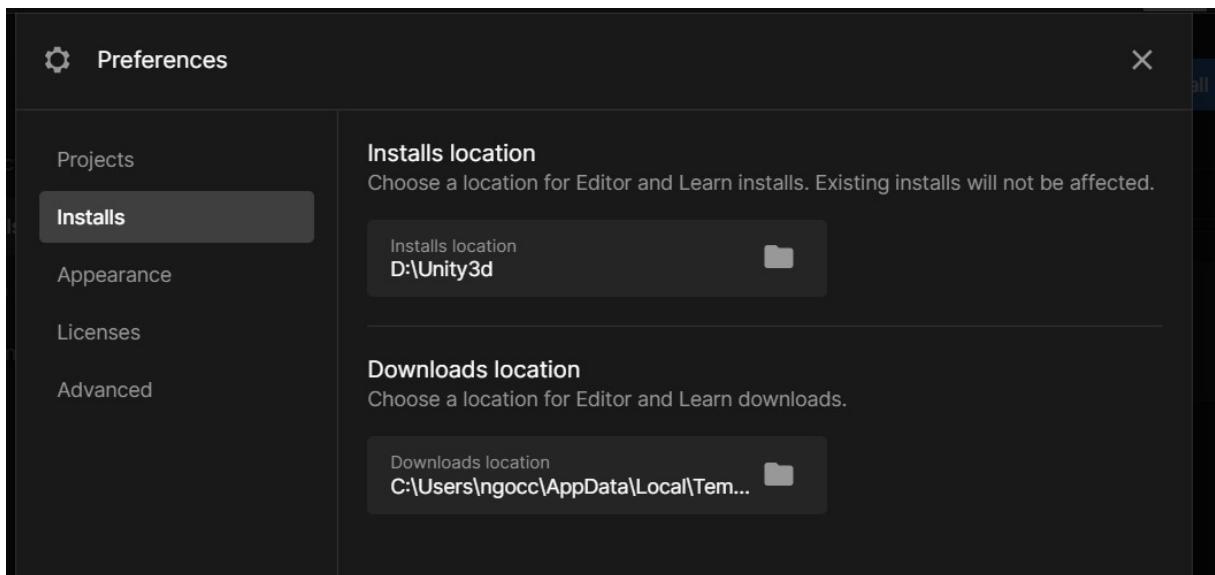


Figure 4.13: Chọn vị trí cài đặt Unity Editor

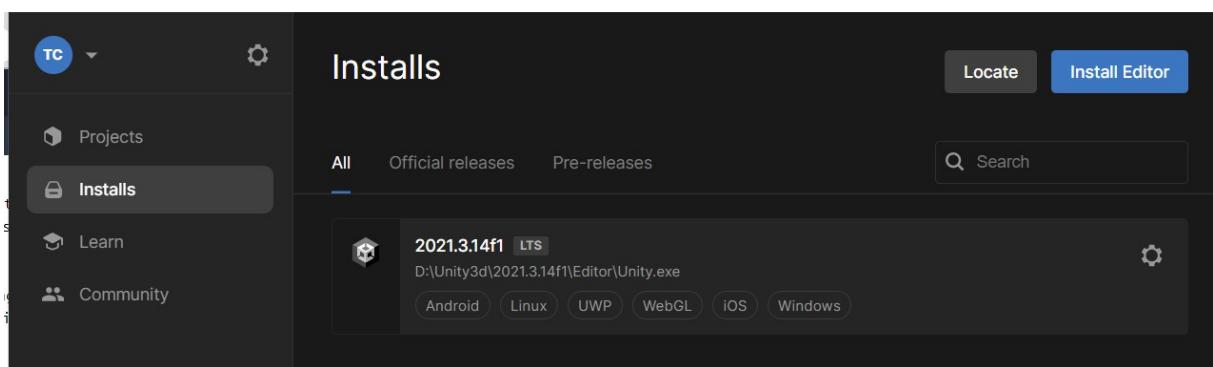


Figure 4.14: Khi Install thành công, Unity editor sẽ hiện ra như hình

4.2.5.3 Tạo Project trong Unity 3D

Ta vào Project và nhấn nút New Project để tạo project mới

Lúc này, sẽ hiện ra rất nhiều Template để ta lựa chọn. Để phục vụ cho việc mở rộng sau này, nhóm chọn Template là một project có **3D Mobile** là core. Việc chọn template thực chất chỉ là một phương pháp mà Unity làm để hỗ trợ ta cài đặt các module và package cần thiết cho mục đích tạo Project,

4.2. Hiện thực điều khiển cánh tay robot với giao thức OPCUA

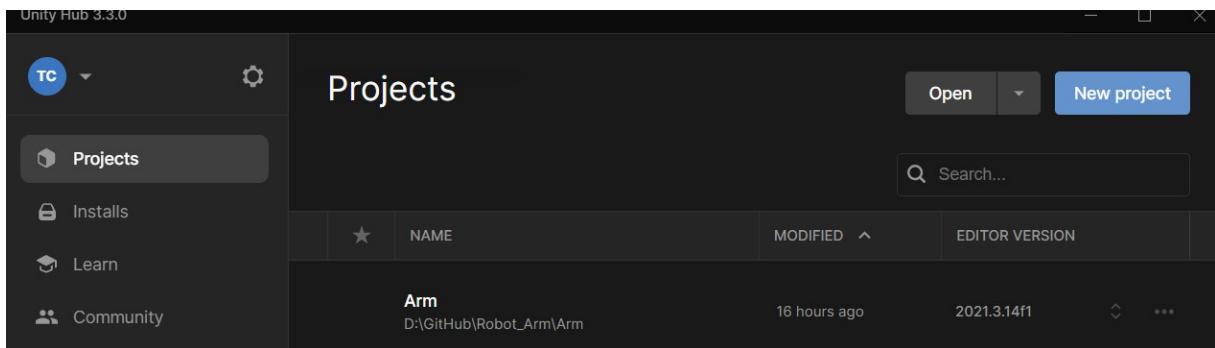


Figure 4.15: Tao project Unity mới

ta hoàn toàn có thể tự cài thêm các module khi cần, nên không cần phải lo lắng nếu như ta chọn nhầm ở bước Template.

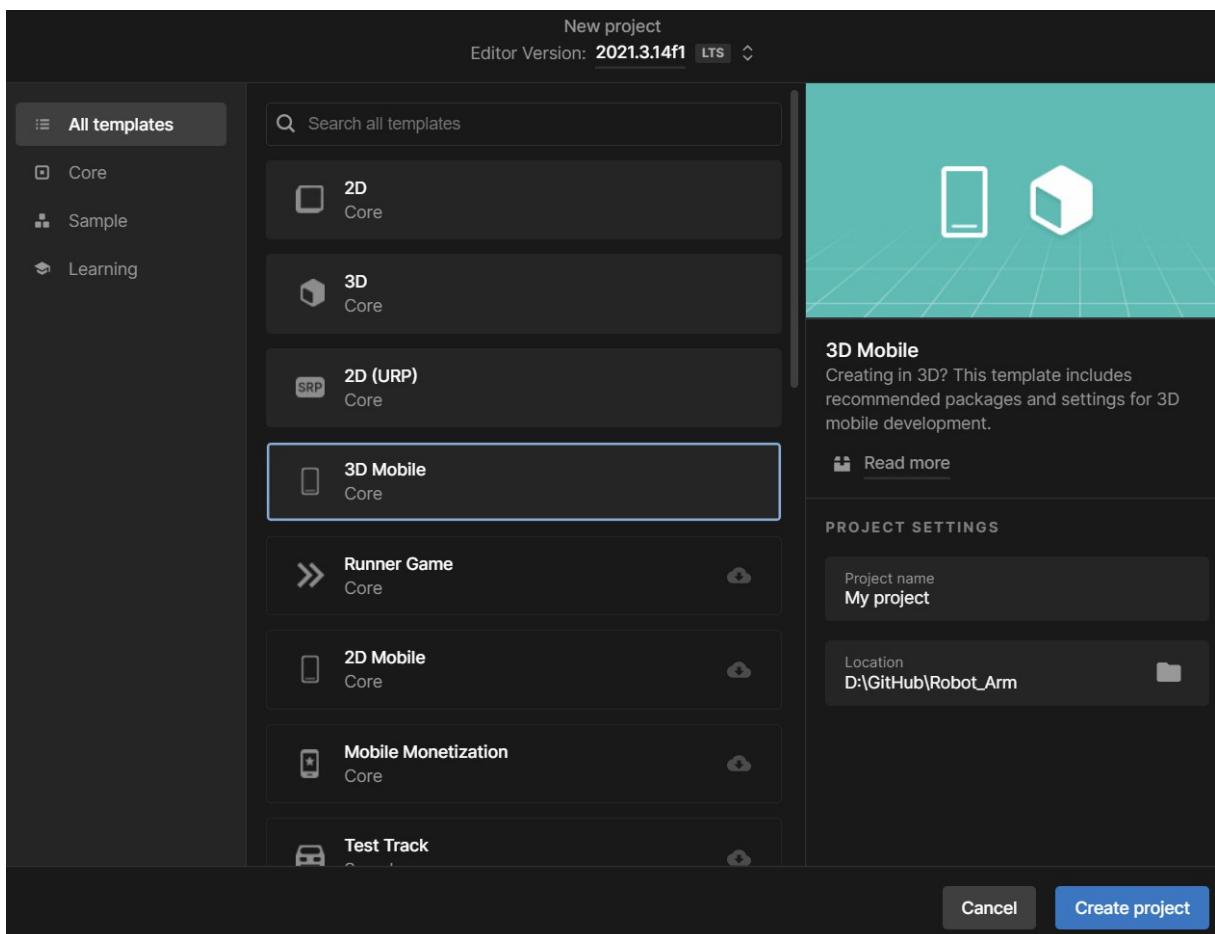


Figure 4.16: Chọn template phù hợp với ứng dụng

Khi mở project xong, ta sẽ có giao diện như sau:

4.2. Hiện thực điều khiển cánh tay robot với giao thức OPCUA

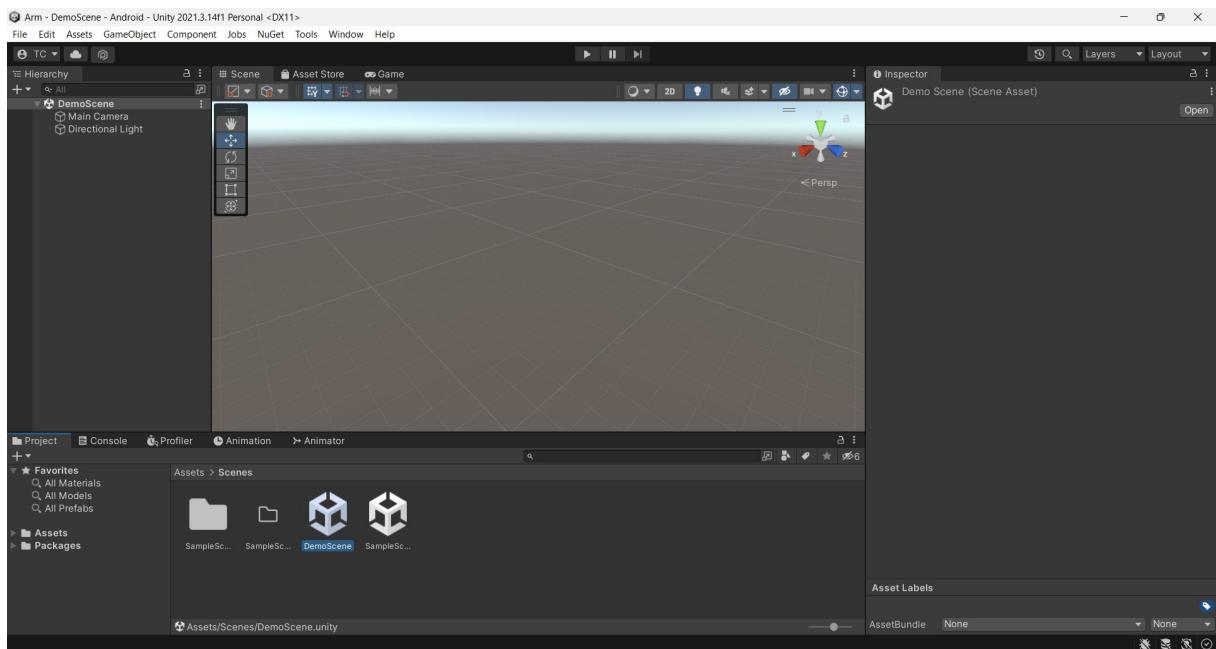


Figure 4.17: Giao diện của Unity

4.2.5.4 Hiện thực ứng dụng

Hiện tại, nhóm chỉ dự định làm một giao diện 2D để điều khiển cánh tay, do đó, ta sẽ bật sang góc nhìn 2D bằng cách chọn Icon 2D trong màn hình Scene

Tiếp đến, ta tạo một Canvas. Canvas là nơi tập hợp các thành phần của UI. Ta có thể thêm Canvas bằng cách nhấn chuột phải ở khung Hierarchy → UI → Canvas. Canvas được tạo ra một hình chữ nhật như sau:

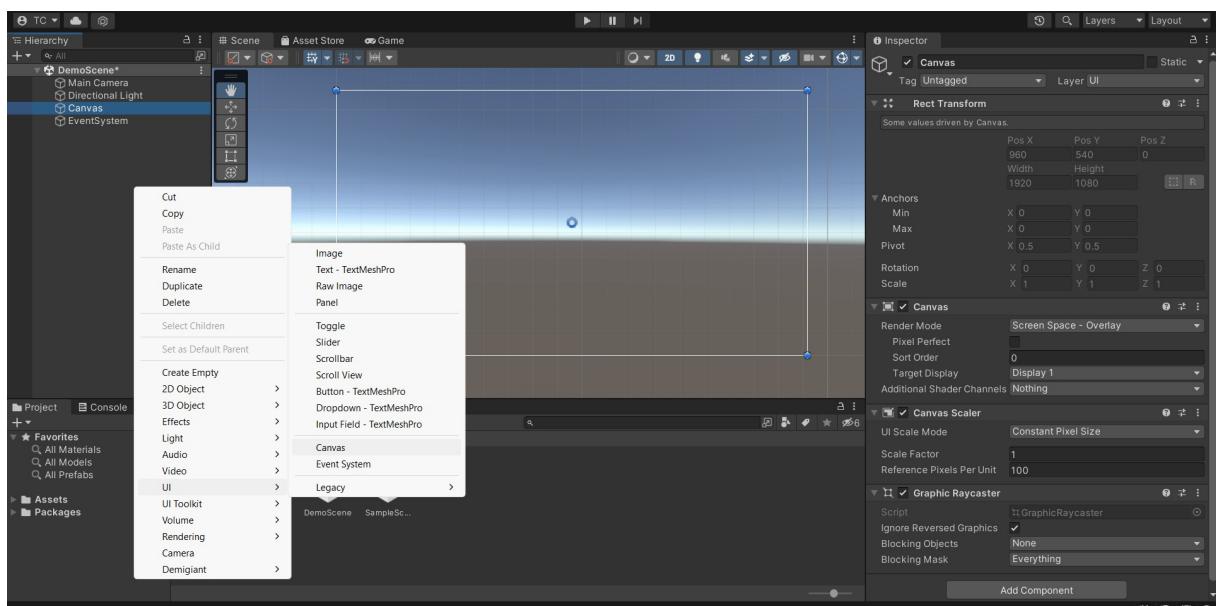


Figure 4.18: Tạo Canvas để làm giao diện trong Unity

Ta chỉnh lại Canvas và đặt Camera để camera tập trung vào canvas. Ở tab inspector bên phải, trong Canvas, ta chỉnh Render Mode thành Screen Space - Camera, và kéo thả Main Camera từ Hierarchy vào Render Camera.

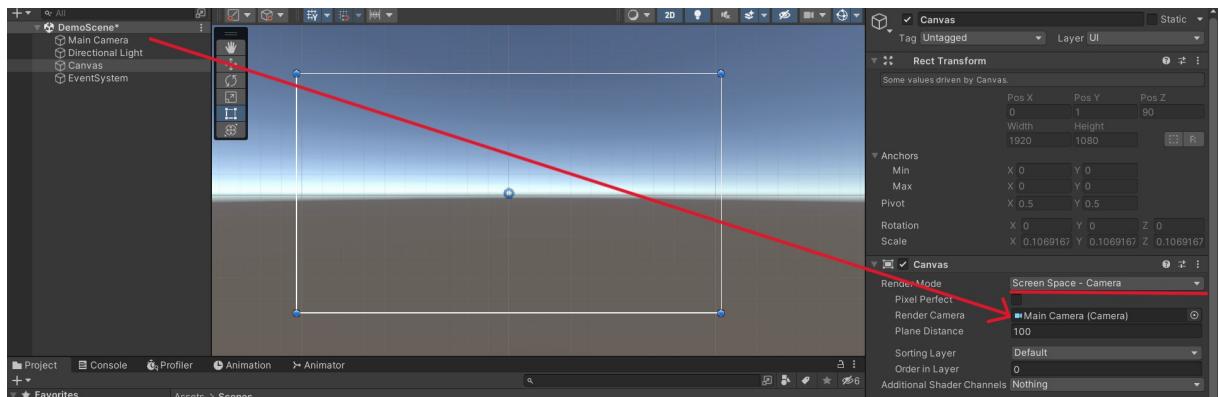


Figure 4.19: Thay đổi góc camera

Cài đặt xong, ta sẽ tiến hành thêm các hình ảnh (Images), Chữ (Texts) hoặc nút nhấn (Buttons) vào trong Canvas và sắp xếp chúng thành giao diện. Hình ảnh, icon cho các nút nhấn và chữ được lấy từ Asset Store của Unity. Các Asset được nhóm sử dụng là:

- Sci-fi GUI skin có các hình ảnh được sử dụng cho nút nhấn và Input field
- Minimal UI Sounds âm thanh đơn giản được sử dụng khi nhấn nút
- DOTween (HOTween v2) hỗ trợ các hàm để tạo ra sự hoạt họa mượt mà cho các đối tượng trong Unity.
- Clean Setting UI - chứa các đối tượng giúp dễ dàng tạo ra một giao diện Setting (cài đặt) đẹp và tinh giản.
- Sweet Land GUI Chứa các hình ảnh thân thiện dùng để tạo các đối tượng cho giao diện UI

4.2.5.4.1 Giao diện ứng dụng

Nhóm đề xuất giao diện của ứng dụng như sau:

Trong Canvas, sẽ có một đối tượng Image dùng để làm Background cho ứng dụng

4.2. Hiện thực điều khiển cánh tay robot với giao thức OPCUA

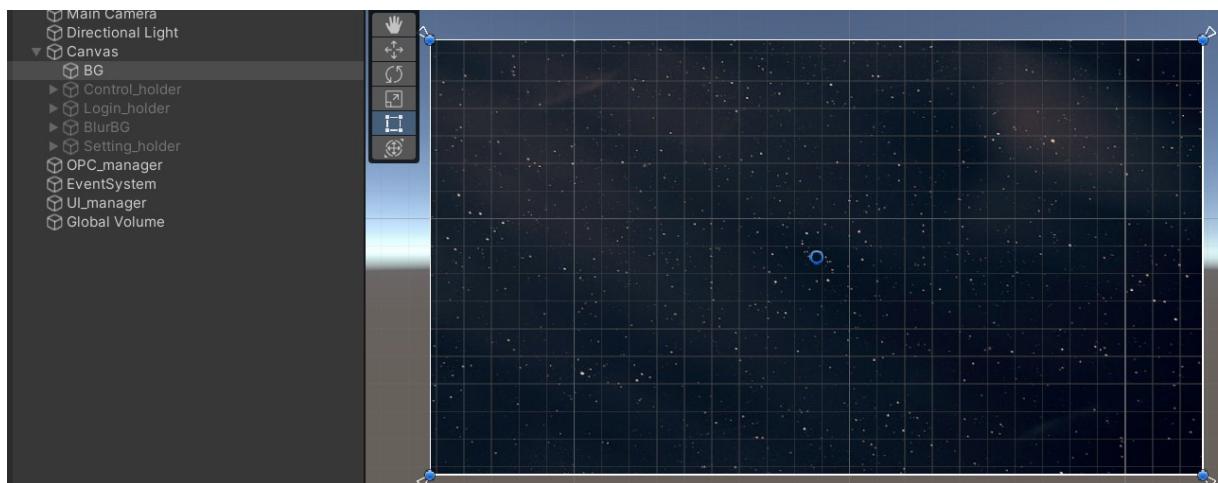


Figure 4.20: Background của ứng dụng

Tiếp đến, có đối tượng Login Holder là một Game Object, nó sẽ chứa các phần tử bên trong để hỗ trợ việc ghi địa chỉ của 1 server OPC-UA và kết nối với server đó.

- URL (Input field): một trường input để người dùng nhập vào địa chỉ của OPC-UA server cần kết nối kèm theo số port
- connectBtn (Button): một nút nhấn để tiến hành kết nối hoặc ngắt kết nối
- Text (TMP) (text): một ô chữ ghi trạng thái kết nối của ứng dụng đối với server OPC-UA. Ô text này cũng được sử dụng để thể hiện các lỗi kết nối nếu có

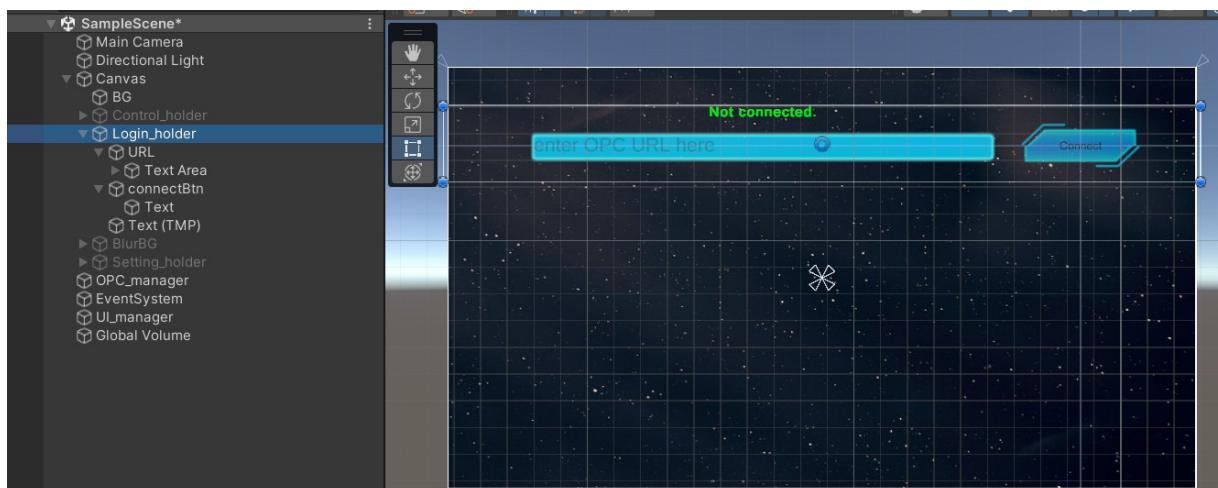


Figure 4.21: Game Object Login Holder

Đối tượng Control holder là một Game Object, chứa các phần tử để điều khiển hoạt động của cánh tay.

- Button_group_1: Game object, gồm các nút mũi tên lên, xuống, trái, phải để điều khiển cánh tay đi lên, đi xuống, xoay sang trái hoặc sang phải
- Button_group_2: Game object, gồm 2 nút nhấn, một nút để điều khiển cánh tay tiến tới trước, nút kia để điều khiển cánh tay lùi về sau
- Button_group_3: Game object, gồm 2 nút nhấn, một nút để điều khiển đầu gấp đóng lại, và một nút để điều khiển đầu gấp mở ra.

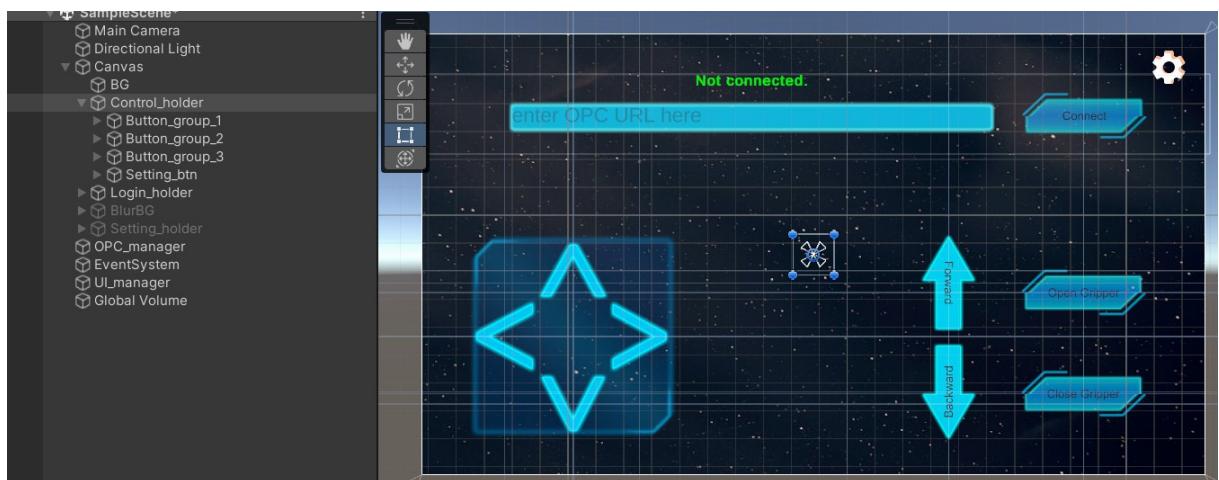


Figure 4.22: Game Object Control holder

4.2. Hiện thực điều khiển cánh tay robot với giao thức OPCUA

Đối tượng BlurBG là một Image, phía trong có chứa một hình tròn tượng trưng cho việc loading (đang tải). Image này sẽ hiện ra khi ứng dụng đang làm hành động xử lý dài.

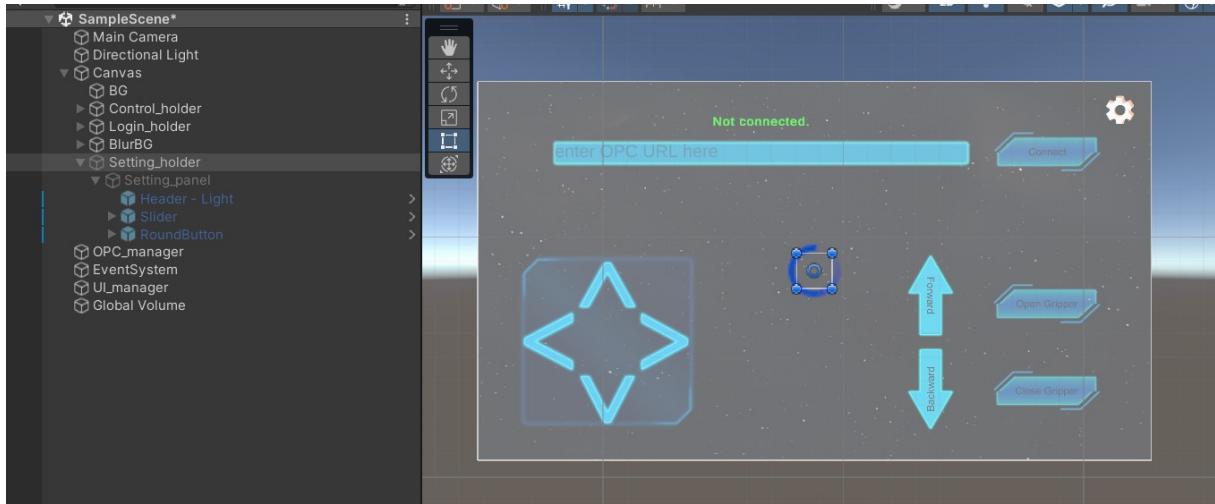


Figure 4.23: Game Object BlurBG

Đối tượng Setting Holder là một Game Object, chứa một bảng setting gồm các đối tượng được tạo sẵn (prefabs) lấy từ Asset Clean UI, nhằm tạo 1 bảng setting đơn giản nhưng dễ sử dụng và đẹp. Setting holder gồm những thành phần sau:

- Setting_panel (Image): Khung nền trắng cho setting
 - Header - Light (prefabs): Một khung chữ làm tiêu đề của hộp thoại setting
 - Slider (prefabs): Một thanh kéo trượt (slider) dùng để chỉnh tốc độ quay của cánh tay
 - RoundButton (prefabs): Một nút nhấn dùng để đóng giao diện setting.

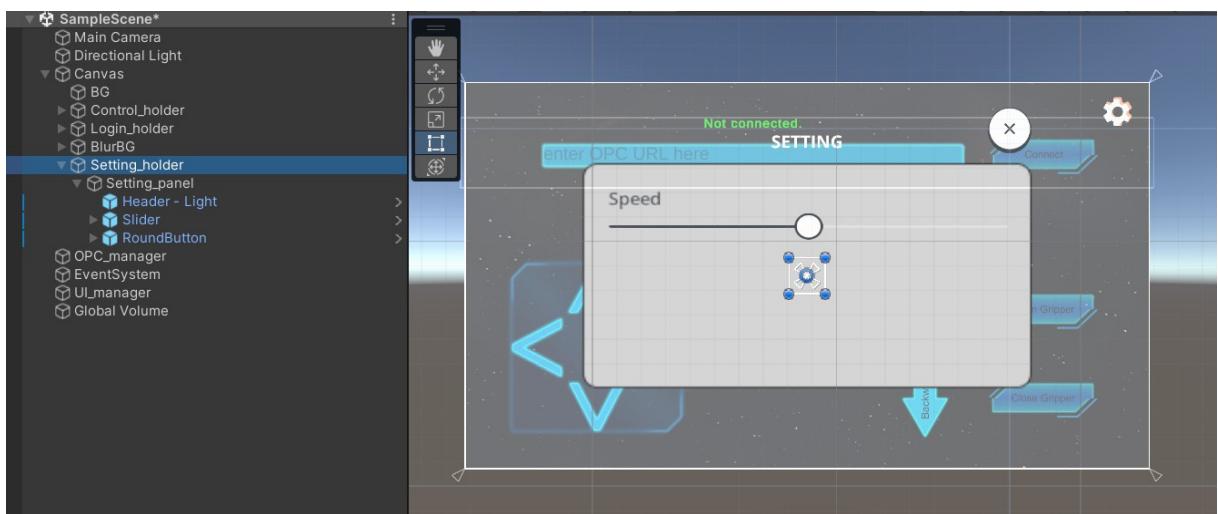


Figure 4.24: Game Object Setting Holder

4.2.5.4.2 Cài đặt thư viện OPC-UA cho ứng dụng

Giới thiệu sơ lược về .NET framework và .NET platform

.NET Framework là một software framework độc quyền phát triển bởi Microsoft để xây dựng và chạy các ứng dụng trên Windows, nó là một phần của .NET platform, gồm tập hợp nhiều công nghệ khác nhau để có thể xây dựng các ứng dụng để chạy cho Linux, macOS, Windows, iOS, Android và hơn nữa. Hiện nay, có nhiều hiện thực của .NET platform, cho phép .NET code có thể thực thi trên nhiều nền tảng khác nhau, có thể kể đến như:

- .NET framework là hiện thực ban đầu của .NET, nó hỗ trợ chạy trang web, dịch vụ, ứng dụng máy tính và nhiều thứ khác trên Windows
- .NET là hiện thực đa nền tảng để chạy trang web, dịch vụ, ứng dụng console trên Windows, Linux và macOS.
- Xamarin/Mono là hiện thực của .NET để chạy các ứng dụng trên các hệ điều hành nổi tiếng dành cho di động, kể cả Android và iOS.

Kiến trúc của .NET Framework

Có hai thành phần chính của .NET Framework là Common Language Runtime và .NET Framework Class Library

- **Common Language Runtime (CLR)** là một "execution engine" xử lý việc chạy ứng dụng. Nó cung cấp các dịch vụ như quản lý luồng (thread management), thu thập rác (garbage collection), gõ an toàn (type-safety), xử lý ngoại lệ (exception handling) và nhiều hơn nữa
- **Class Library** cung cấp một tập API và kiểu dữ liệu cho các chức năng thông dụng. Nó cung cấp kiểu dữ liệu cho chuỗi, ngày tháng, số, v.v.. Class Library cũng chứa các API để đọc và ghi tệp, kết nối cơ sở dữ liệu, vẽ, v.v..

Các ứng dụng .Net được viết bằng ngôn ngữ C#, F# hoặc Visual Basic. Code được biên dịch thành "language-agnostic Common Intermediate Language (CIL)". Code đã được biên dịch sẽ được lưu ở dạng assembles (hợp ngữ) ở dạng file với đuôi là .dll (dynamic link library) hoặc .exe (executable file).

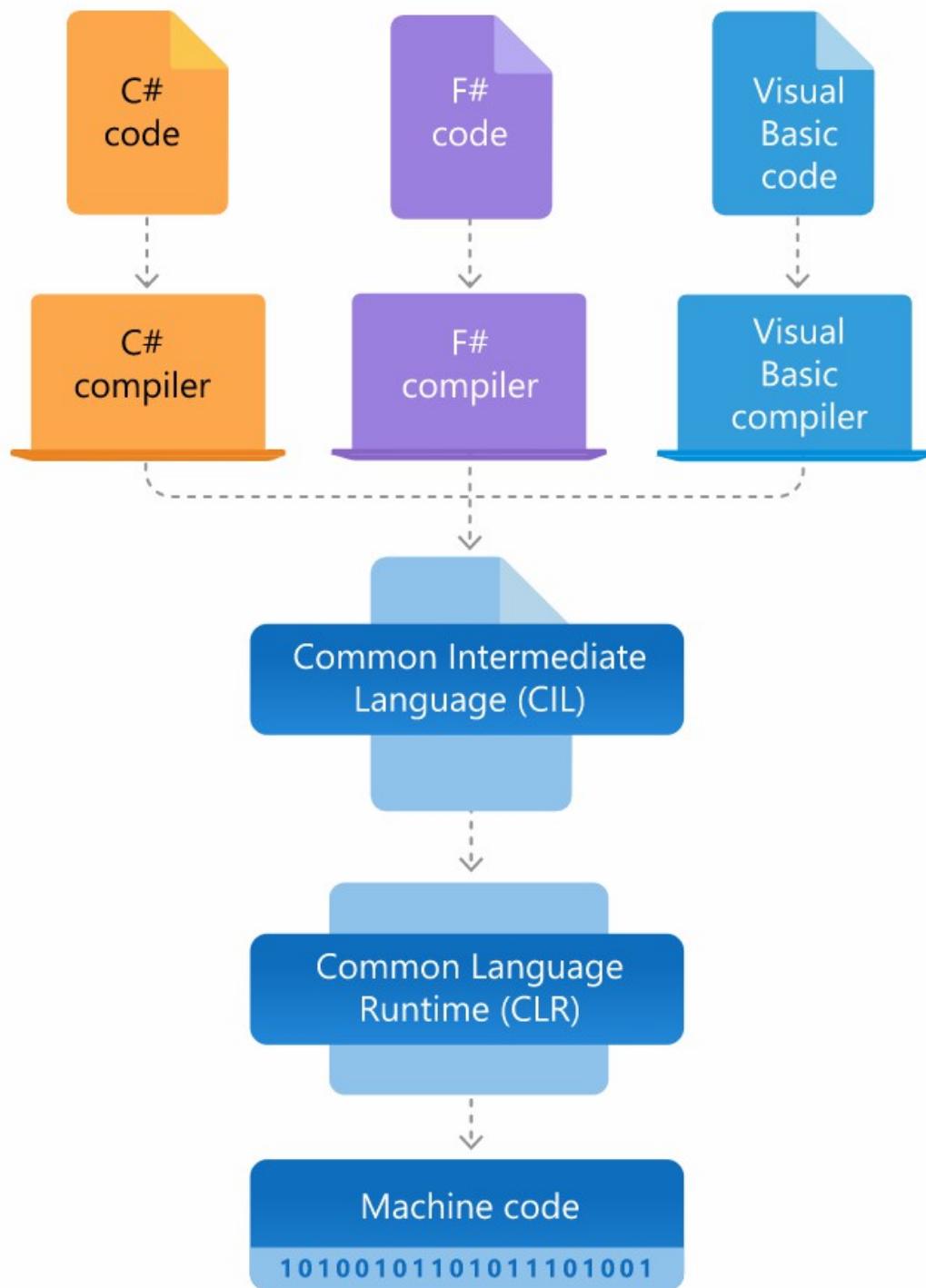


Figure 4.25: Thành phần trong .NET Framework

Khi một ứng dụng chạy, CLR (Common Language Runtime) sẽ lấy assembly và sử dụng Just-In-Time compiler (JIT) để chuyển chúng thành mã máy để có thể chạy trên một kiến trúc máy tính cụ thể mà nó đang được chạy.

Để có thể giao tiếp với server bằng OPC-UA, nhóm chọn thư viện Opc.UaFx.Client để cài đặt cho ứng dụng. Thư viện này hỗ trợ trực tiếp cho nền tảng .NET

4.2. Hiện thực điều khiển cánh tay robot với giao thức OPCUA

Framework. Như ta thấy, thư viện này hỗ trợ khá nhiều các nền tảng .NET khác nhau.

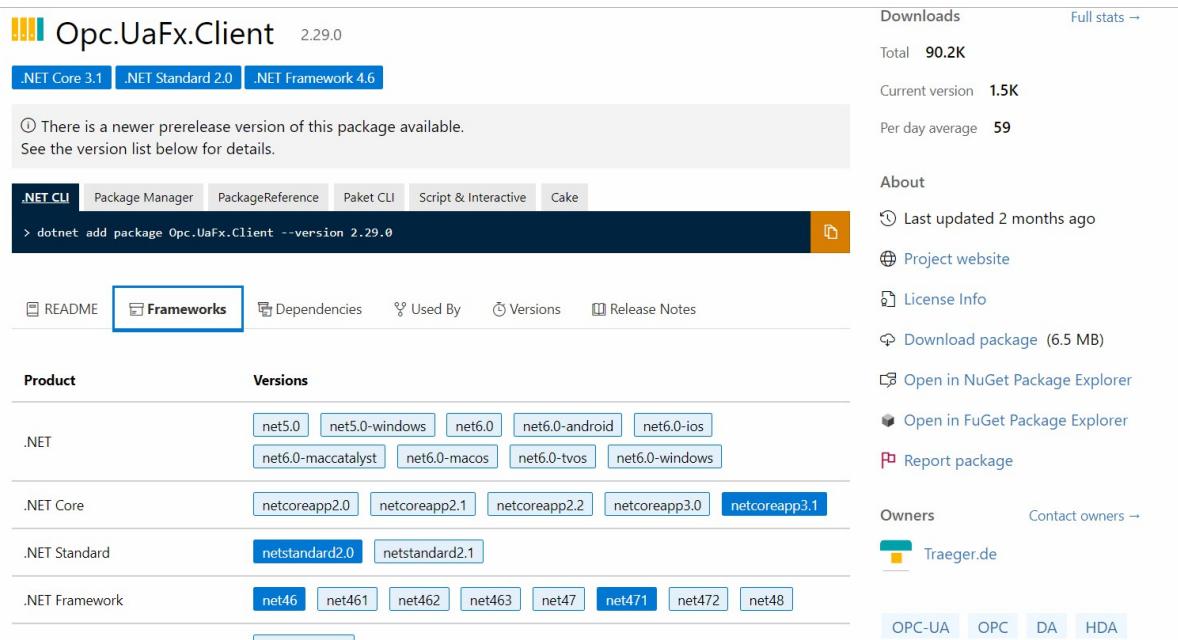


Figure 4.26: Packet Opc.UaFx.Client

Ở đây, ta chọn .NET Framework, bản 4.71. Nhấn chữ "Download Packet" để download packet này về. Kèm theo đó, ta cũng cần phải download cả những packet khác mà thư viện này phụ thuộc vào. Nhấn vào tab Dependency để nhìn thấy các phiên bản packet cần thiết, và ta sẽ tải đầy đủ các packet đó. Danh sách các packet sẽ như sau:

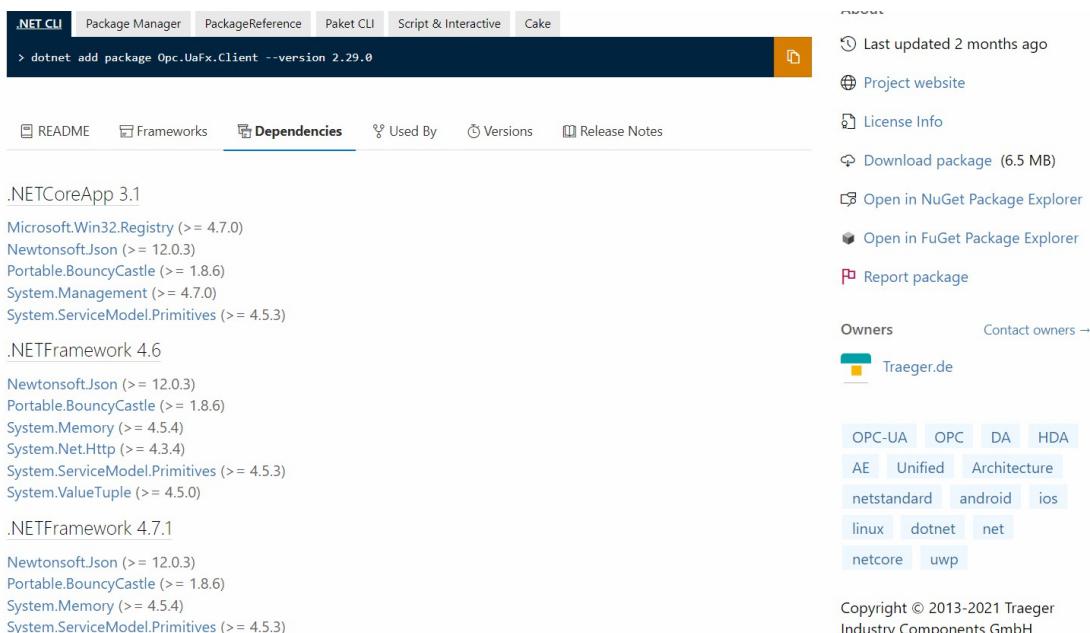


Figure 4.27: Tải về packet và xem danh sách cách packet liên quan

- Opc.UaFx.Client
 - Newtonsoft.Json
 - Portable.BouncyCastle (.NETFramework 4.0)
 - System.Memory (.NETFramework 4.6.1)
 - * System.Buffers (.NETFramework 4.6.1)
 - * System.Numerics.Vectors (.NETFramework 4.6)
 - * System.Runtime.CompilerServices.Unsafe (.NETFramework 4.6.1)
 - System.ServiceModel.Primitives (.NETFramework 4.6.1)

Sau khi tải về các packet, ta sẽ lấy các file .dll từ trong packet đó bằng cách thủ công. Packet Newtonsoft.Json đã được tích hợp sẵn trong Unity nên sẽ không cần thiết để tải module đó về. Để lấy các file .dll bằng tay, ta làm theo các bước sau:

1. Đổi đuôi của các packet từ .nupkg thành .zip
2. Giải nén packet
3. Tìm và lấy các file .dll với phiên bản NetFramework mong muốn.

Sau khi làm các bước trên với từng packet, ta sẽ có được danh sách các file .dll như sau:

Sau đó, ta sẽ copy thư mục chứa các file .dll này vào thư mục Asset của project Unity. Unity khi nhận thấy các file này sẽ tự compile, compile xong thì ta sẽ có thể sử dụng được các API để hiện thực client OPC-UA.

4.2. Hiện thực điều khiển cánh tay robot với giao thức OPCUA

Name	Date modified	Type
BouncyCastle.Crypto.dll	19/10/2021 11:53	Application extens...
BouncyCastle.Crypto.xml	19/10/2021 11:27	XML Source File
Opc.UaFx.Client.dll	16/09/2022 14:02	Application extens...
Opc.UaFx.Client.xml	16/09/2022 13:56	XML Source File
System.Buffers.dll	19/02/2020 10:05	Application extens...
System.Buffers.xml	19/02/2020 10:05	XML Source File
System.Memory.dll	08/05/2022 03:31	Application extens...
System.Memory.xml	08/05/2022 03:31	XML Source File
System.Numerics.Vectors.dll	15/05/2018 13:29	Application extens...
System.Numerics.Vectors.xml	15/05/2018 13:29	XML Source File
System.Runtime.CompilerServices.Unsafe....	22/10/2021 23:40	Application extens...
System.Runtime.CompilerServices.Unsafe....	19/10/2021 07:14	XML Source File
System.ServiceModel.Primitives.dll	16/08/2022 21:01	Application extens...
System.ServiceModel.Primitives.pdb	16/08/2022 21:01	Program Debug D...

Figure 4.28: Danh sách các file .dll đầy đủ để sử dụng thư viện Opc.UaFx.Client

4.2.5.4.3 Viết các script (code) cho hoạt động của ứng dụng Script myopc.cs

Trong script này, nhóm sẽ viết các hàm và chức năng để thực hiện kết nối và giao tiếp với server OPC-UA sử dụng thư viện đã cài đặt ở trên. Để sử dụng thư viện, ta thêm dòng sau ở đầu script.

```
1  using Opc.UaFx;  
2  using Opc.UaFx.Client;
```

Ngôn ngữ C# là một ngôn ngữ thuần hướng đối tượng, do đó, trong script này nhóm sẽ tạo ra 1 class (lớp) là myopc. trong lớp này gồm một số hàm quan trọng và chức năng của nó như sau:

- public void BtnConnect(): Hàm này sẽ được gắn với 1 nút nhấn, tùy vào

4.2. Hiện thực điều khiển cánh tay robot với giao thức OPCUA

trạng thái kết nối với server OPC-UA, hàm này có thể khởi động kết nối hoặc ngắt kết nối với server

- `private void OPCCConnect()` Hàm này sẽ thu thập thông tin địa chỉ của server OPC-UA từ một trường input field, sau đó khởi tạo đối tượng myclient là một `opcClient`. Nếu khởi tạo thành công đối tượng (địa chỉ server hợp lệ), hàm sẽ tạo ra 1 thread riêng để thực hiện kết nối với server, đồng thời tạo một Coroutine (1 hàm đặc biệt của Unity, cho phép hàm có thể tạm dừng và quay trở lại thực hiện sau một khoảng thời gian, Coroutine là một loại hàm đặc biệt chạy ở thread chính (main thread)). Trong hàm này, coroutine được gọi sẽ thực hiện quan sát sự thành công của kết nối từ thread thực hiện kết nối với server, và thay đổi giao diện cho phù hợp.

```
1  private void OPCCConnect()
2  {
3      string url = UI_manage.instance.UI_GetUrl();
4      Debug.Log(url);
5      try
6      {
7          myclient = new OpcClient(url);
8      }
9      catch (Exception e)
10     {
11         Debug.Log(e);
12         UI_manage.instance.UI_Blr(false);
13         UI_manage.instance.UI_Text(e.Message, true);
14         return;
15     }
16     Thread connectThread = new Thread(ConnectStart);
17     connectThread.Start();
18
19     StartCoroutine(Connecting());
20 }
21
```

- `private void OPCDisconnect(bool isManual)` Hàm thực hiện ngắt kết nối với server và thay đổi giao diện thành ngắt kết nối.
- `public void ConnectStart()` Hàm thực hiện kết nối với server và lấy thông

4.2. Hiện thực điều khiển cánh tay robot với giao thức OPCUA

tin 4 Node đặc biệt điều khiển 4 servo của server là **M, R, L, F**. Do việc thực hiện kết nối khá lâu, hàm này sẽ được chạy ở 1 thread khác, và thay đổi các biến phù hợp để thông báo sự thành công hay thất bại của kết nối.

```
1  public void ConnectStart()
2  {
3      try
4      {
5          myclient.Connect();
6
7          var paramNode = myclient.BrowseNode("ns=2;i=1");
8
9          Debug.Log(paramNode.DisplayName + paramNode.NodeId);
10
11         foreach (var childNode in paramNode.Children())
12         {
13
14             Debug.Log(childNode.DisplayName + childNode.NodeId)
15
16         ;
17
18             switch (childNode.DisplayName)
19             {
20
21                 case "M":
22
23                     midServoID = childNode.NodeId.ToString();
24
25                     break;
26
27                 case "L":
28
29                     leftServoID = childNode.NodeId.ToString();
30
31                     break;
32
33                 case "R":
34
35                     rightServoID = childNode.NodeId.ToString();
36
37                     break;
38
39                 case "F":
40
41                     gripperServoID = childNode.NodeId.ToString()
42
43                     () ;
44
45                     break;
46
47             }
48
49         }
50
51
52         isConnected = true;
53
54
55         midServo = myclient.ReadNode(midServoID).As<int>(90);
56         leftServo = myclient.ReadNode(leftServoID).As<int>(90);
57         rightServo = myclient.ReadNode(rightServoID).As<int
58 >(90);
59
60         gripperServo = myclient.ReadNode(gripperServoID).As<int
```

4.2. Hiện thực điều khiển cánh tay robot với giao thức OPCUA

```
34         >(90);  
35  
36             Debug.Log(isConnected.ToString() + midServo.ToString()  
37             + leftServo.ToString() + rightServo.ToString() + gripperServo.  
38             ToString());  
39         }  
40     catch (OpcException e)  
41     {  
42         isFailed = true;  
43         Debug.Log("Unable to connect to server!" + e);  
44         ConnectErr = e.ToString();  
45     }  
46     catch (Exception e)  
47     {  
48         isFailed = true;  
49         Debug.Log("Unable to connect to server!" + e);  
50         ConnectErr = e.ToString();  
51     }  
52 }  
53 }
```

- `public IEnumerator Connecting()` Một hàm coroutine, hàm này sẽ quan sát các biến trạng thái kết nối từ hàm `connectStart()` và thay đổi giao diện theo trạng thái kết nối

```
1     public IEnumerator Connecting()  
2     {  
3         while (!isConnected)  
4         {  
5             if (isFailed)  
6             {  
7                 UI_manage.instance.UI_Text(ConnectErr, true);  
8                 isFailed = false;  
9                 UI_manage.instance.UI_Blur(false);  
10                break;  
11            }  
12            yield return new WaitForSeconds(0.5f);  
13        }  
14        if (isConnected)  
15        {  
16            UI_manage.instance.UI_Connected();  
17            UI_manage.instance.UI_Blur(false);  
18        }  
19    }  
20 }
```

```

18     }
19 }
20 }
```

- `public void ChangeSpeed(float speed)` speed là giá trị từ 1 tới 5, giúp thay đổi bước nhảy của góc quay các servo trong cánh tay robot, do đó tăng hoặc giảm tốc độ quay của cánh tay robot ở các điểm xoay.
- `public bool armMove(int direction)` Hàm này thực hiện điều khiển chuyển động cánh tay bằng cách thay đổi giá trị các node của OPC-UA sub server mà ứng dụng kết nối tới. Giá trị direction là một số nguyên từ 0 tới 7, tương ứng cho 8 phương hướng di chuyển của cánh tay robot:
 - 0: Di chuyển lên
 - 1: Di chuyển xuống
 - 2: Xoay trái
 - 3: Xoay phải
 - 4: Hướng về trước
 - 5: Lùi về sau
 - 6: Mở đầu gắp
 - 7: Đóng đầu gắp

Với mỗi cách di chuyển, hàm này sẽ thay đổi giá trị các node tương ứng trong sub server OPC-UA

Script UI_manage.cs

Đây là script giúp quản lý các thành phần giao diện và đưa ra các hàm phù hợp để script `myopc.cs` có thể sử dụng. Tương tự như `myopc.cs` script, `UI_manage` cũng tạo ra một lớp (class) là `UI_manage`, chứa các hàm quan trọng sau:

- `void Start()` Khởi tạo giao diện khi ứng dụng mới chạy, đồng thời lấy các đối tượng giao diện cần thiết.

```

1     void Start()
2     {
3         mytext = loginHolder.GetComponentsInChildren<TMP_Text>()
4         [2];
```

4.2. Hiện thực điều khiển cánh tay robot với giao thức OPCUA

```
4         urlInput = loginHolder.GetComponentInChildren<
5             TMP_InputField>()[0];
6         connectBtn = loginHolder.GetComponentInChildren<Button>()
7             [0];
8         Debug.Log(mytext);
9         Debug.Log(urlInput);
10        Debug.Log(connectBtn);
11        UI_CanvasGroup_enabler(loginHolder, true, 1);
12
13        UI_CanvasGroup_enabler(controlHolder, false, 0.1f);
14
15        loginHolder.transform.DOLocalMoveY(0f, 1, true);
16        UI_Text("Not connected", true);
17        UI_Blr(false);
18    }
19
```

- `public void UI_Blr(bool isBlur)` Dùng để làm mờ màn hình hoặc tắt làm mờ màn hình, dùng khi đang kết nối hoặc mở hợp thoại cài đặt (setting)
- `void UI_CanvasGroup_enabler(CanvasGroup cg, bool enable, float duration)` Giúp ẩn/hiện các Game Object chứa thành phần CanvasGroup sử dụng DOTween tạo ra hiệu ứng mượt mà hơn
- `public void UI_Connected()` Giúp thay đổi giao diện khi kết nối thành công

```
1     public void UI_Connected()
2     {
3         loginHolder.transform.DOLocalMoveY(350f, 1, true);
4         UI_CanvasGroup_enabler(controlHolder, true, 1);
5
6         UI_Text("Connected", false);
7         connectBtn.GetComponentInChildren<Text>().text = "
8             Disconnect";
9     }
```

- `public void UI_Disconnected()` Giúp thay đổi giao diện khi kết nối bị ngắt

```
1     public void UI_Disconnected()
```

4.2. Hiện thực điều khiển cánh tay robot với giao thức OPCUA

```
2     {
3         loginHolder.transform.DOLocalMoveY(0f, 1, true);
4         UI_CanvasGroup_enabler(controlHolder, false, 1);
5
6         connectBtn.GetComponentInChildren<Text>().text = "Connect";
7         UI_Text("Disconnected", true);
8     }
9
```

- `public void UI_Text(string text, bool isError)` Dùng để thay đổi nội dung của một đối tượng text ngay trên input field điền địa chỉ kết nối tới server, thường dùng để thể hiện trạng thái kết nối hoặc thể hiện lỗi.
- `public void UI_changeSpeed(float speed)` Gọi đến hàm `ChangeSpeed(speed)` của class myopc để thay đổi bước nhảy của góc xoay servo, qua đó thay đổi tốc độ của cánh tay

```
1     public void UI_changeSpeed(float speed)
2     {
3         myopc.instance.ChangeSpeed(speed);
4     }
5
```

- `public void UI_toSetting(bool showSetting)` Dùng để ẩn/hiện hộp thoại setting.
- `public string UI_GetUrl()` Dùng để lấy nội dung của địa chỉ trong Input Field, và xử lý khi cần.

```
1     public string UI_GetUrl()
2     {
3         return urlInput.text;
4     }
5
```

Script Button_handler.cs

Script này là 1 script đơn giản được gắn vào mỗi nút nhấn điều khiển cánh tay, nó sẽ chú ý vào các sự kiện khi nút được nhấn và khi được thả ra. Nếu nút được nhấn, một coroutine sẽ được chạy mỗi 50ms và gọi hàm `armMove` của lớp myopc để điều khiển cánh tay theo hướng yêu cầu, kèm theo đó là phát ra một âm thanh đơn giản để tạo sự phản hồi với người dùng.

4.2.5.4.4 Demo ứng dụng

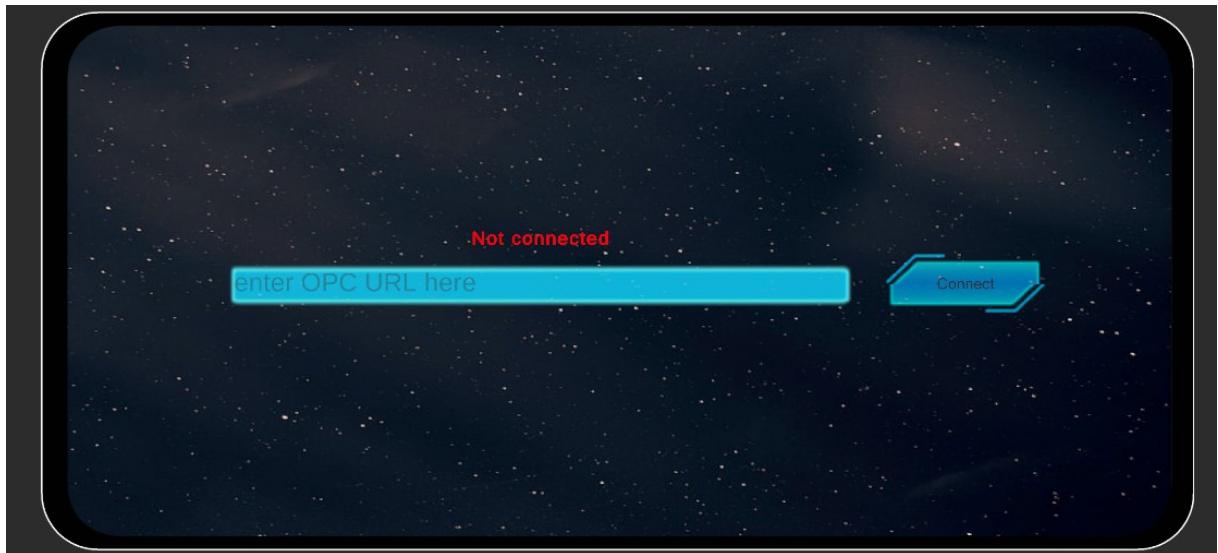


Figure 4.29: Giao diện ứng dụng khi vừa khởi động, chưa kết nối



Figure 4.30: Giao diện ứng dụng khi đã kết nối

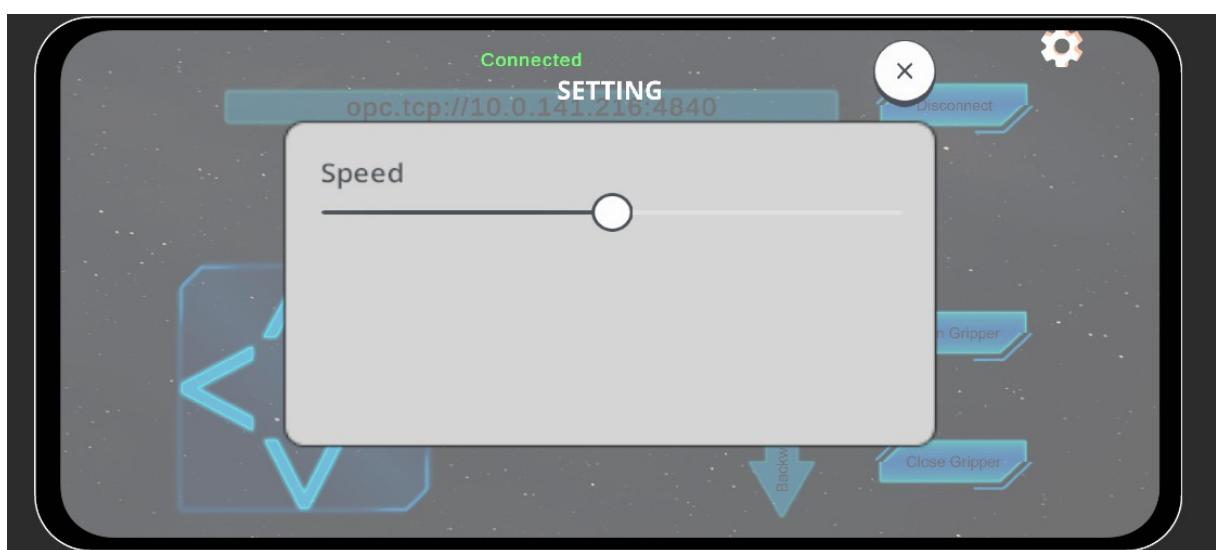


Figure 4.31: Giao diện ứng dụng khi bật hộp thoại setting

CHAPTER 5

TỔNG KẾT VÀ HƯỚNG PHÁT TRIỂN CHO ĐỒ ÁN TỐT NGHIỆP

5.1 Kết quả đạt được trong quá trình thực hiện

Trong suốt quá trình thực hiện đồ án về giao thức OPC-UA nhóm đã đạt được một số kết quả như sau:

5.1.1 Tìm hiểu và hiện thực giao thức OPC-UA

Tìm hiểu cấu trúc cách tổ chức dữ liệu, cách hoạt động trong giao thức OPC-UA và tính ứng dụng của giao thức OPC-UA trong môi trường công nghiệp thông minh 4.0, nơi mà sự giao tiếp giữa các thiết bị là một điều cần được quan tâm rất nhiều, từ những dữ liệu đơn giản như nhiệt độ, độ ẩm, diện tích cản xưởng cho đến năng lượng tiêu thụ, điện năng, hiệu suất làm việc của các thiết bị. Tất cả những dữ liệu này sẽ góp phần cải thiện năng suất làm việc của nhà máy, đồng thời đánh giá hiệu năng của nhà máy một cách chính xác nhất, góp phần tạo nên một nền công nghiệp thông minh hiện đại.

Hiện thực được giao tiếp và gửi các dữ liệu cơ bản giữa các thiết bị thông qua giao thức OPC-UA dựa trên thư viện **opcua** được phát triển trên ngôn ngữ lập trình Python

5.1.2 Thực nghiệm và đưa ra những đánh giá nhận xét giữa OPC-UA và MQTT

Thực hiện các thực nghiệm và đưa ra những đánh giá nhận xét giữa OPC-UA và giao thức khá phổ biến trong IOT hiện nay là MQTT.

Dựa trên kết quả thực nghiệm có thể suy rằng, nhiều lĩnh vực cần sự trao đổi rất lớn về kích thước của dữ liệu, về tốc độ truyền tải của dữ liệu, do đó OPC UA sẽ đáp ứng hiệu quả và đảm bảo cho việc trao đổi trên. Tuy nhiên, không phải tất cả các giao tiếp đều cần phải đáp ứng những nhu cầu về kích thước dữ liệu hay tốc độ, vậy nên đối với những yêu cầu như vậy, MQTT sẽ phù hợp hơn trong việc áp dụng, đồng thời dễ dàng tích hợp các dịch vụ cao cấp hơn phục vụ cho các ứng dụng bên ngoài như quan trắc, điều khiển.

5.1.3 Đề xuất kiến trúc và hiện thực điều khiển cánh tay robot

Kiến trúc hệ thống là một mô hình khái niệm xác định cấu trúc của hệ thống và hành vi tương tác của các thành phần trong hệ thống với nhau. Để tiếp cận với yêu cầu của bài toán điều khiển cánh tay robot bằng mobile app, chúng ta sẽ có nhiều hướng tiếp cận khác nhau và đơn giản nhất sẽ là mô hình một client thiết lập trên mobile app kết nối với một server thiết lập trên một máy tính phục vụ điều khiển cánh tay.

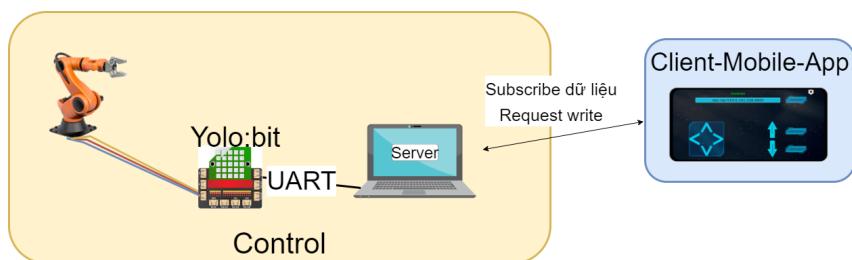


Figure 5.1: Kiến trúc đơn giản với 1 Client - 1 Server

Tuy nhiên với kiến trúc đơn giản này thì sẽ khó có thể phát triển thêm các chức năng kết nối giao tiếp, đồng bộ dữ liệu giữa nhiều thiết bị khác nhau, chính vì thế mà nhóm đã đề xuất ra kiến trúc có tính khả mở cao hơn cho việc kết nối đồng bộ nhiều thiết bị, mở rộng hướng phát triển cho luận văn

5.1. Kết quả đạt được trong quá trình thực hiện

đó là thêm vào một module DataCenter ở giữa để khả năng mở rộng của hệ thống cao hơn.

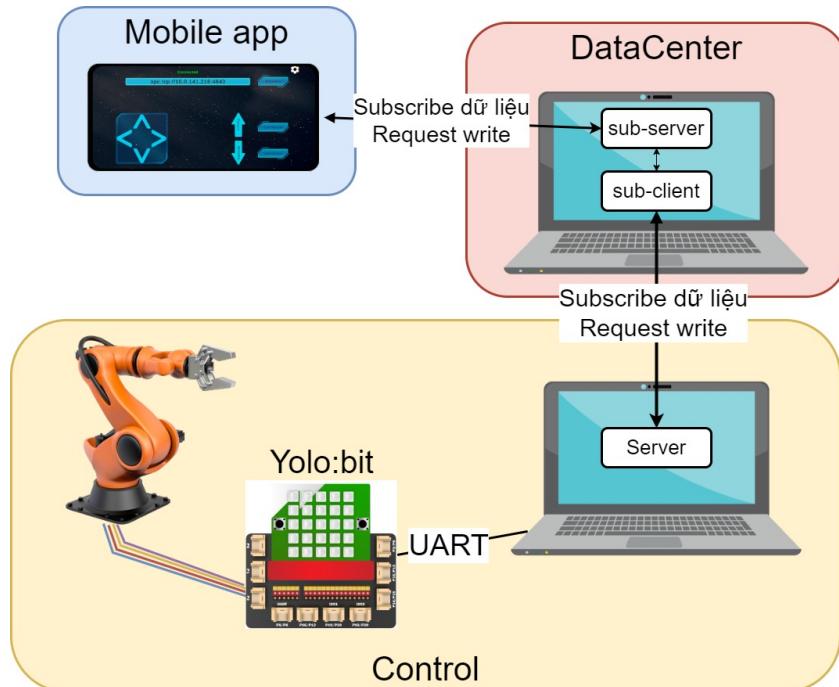


Figure 5.2: Kiến trúc đề xuất

Với kiến trúc đã đề xuất, nhóm đã thành công hiện thực việc điều khiển cánh tay thông qua app mobile, hoạt động tốt với các khớp xoay đơn giản. Dưới đây là giao diện của app mobile:

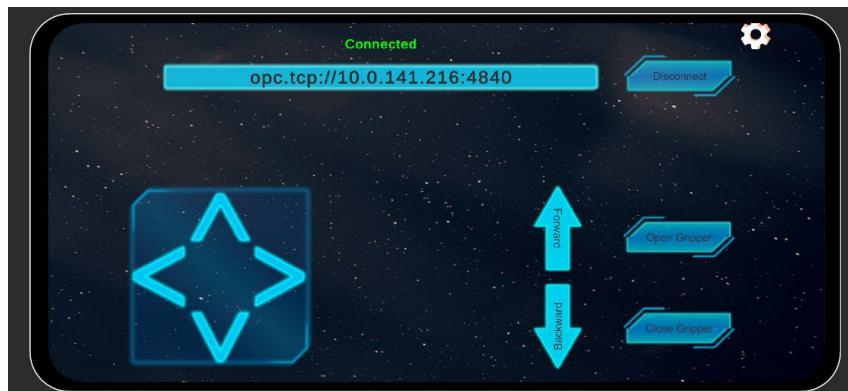


Figure 5.3: Giao diện app mobile

Truy cập video demo điều khiển [tại đây](#).

5.2 Hướng phát triển và mở rộng cho luận văn

Smart Factory là nơi máy móc và thiết bị có thể cải thiện quy trình thông qua tự động hoá và tối ưu hoá, nó không chỉ gói gọn trong 1 màn hình 1 cánh tay, mà đó là sự phối hợp và đồng bộ dữ liệu lẫn nhau giữa nhiều thiết bị (băng chuyền, robot vận chuyển, ...), vì vậy để chuẩn bị cho giai đoạn luận văn nhóm đề xuất ra một số hướng phát triển mở rộng để phù hợp hơn với mô hình của một **Smart Factory**, cụ thể như sau:

5.2.1 Tích hợp thêm cảm biến cho cánh tay

Ở giai đoạn hiện tại, do cánh tay được tích hợp sẵn các driver và việc điều khiển khớp xoay chỉ đơn giản là nhận lệnh từ DataCenter sau đó các khớp tương ứng sẽ xoay những sẽ không thể phản hồi lại về gốc xoay hay vị trí hiện tại của cánh tay, vì thế mà việc đồng bộ dữ liệu của cánh tay đôi khi sẽ gặp chút khó khăn. Để giải quyết được yếu điểm trên, nhóm đề xuất tích hợp thêm các cảm biến về gốc xoay cho cánh tay, để có thể nhận được những phản hồi về gốc xoay cũng như vị trí hiện tại của cánh tay điều này giúp ta dễ dàng thực hiện việc đồng bộ về dữ liệu của cánh tay hơn.

Ngoài ra để kiểm soát độ ổn định của cánh tay, cũng như giúp các mô hình AI đưa ra những cảnh báo về tình trạng hư hỏng của cánh tay hoặc cánh tay cần được bảo trì sửa chữa, nhóm đề xuất gắn thêm vào cánh tay các cảm biến về độ run. Điều này giúp cho ta có cơ sở dữ liệu đưa cho các mô hình AI tính toán đưa ra những dự đoán chính xác về tính ổn định của cánh tay và phát đi những cảnh báo khi cánh tay có dấu hiệu hư hỏng.

5.2.2 Mô phỏng cánh tay trong 3D (Digital Twin)

Từ những dữ liệu đồng được thông qua việc sử dụng giá trị gửi về của cảm biến gốc xoay và dữ liệu phát đi từ DataCenter, ta có thể dùng nó vào việc phát triển một bản sao 3D của cánh tay robot từ đó có thể quan sát được hoạt động của cánh tay thông qua mô hình 3D này.

5.2.3 Gắn thêm camera vận hành các tính năng nhận diện vật phẩm

Trong một Smart Factory các thiết bị hầu như hoàn toàn tự động hóa, hoạt động độc lập mà không cần sự can thiệp hay điều khiển của con người (ngoài việc bảo trì sửa chữa hoặc nâng cấp), để mô phỏng khả năng tự hoạt động của cánh tay, nhóm đề xuất gắn thêm vào cánh tay 1 camera có khả năng xử lý hình ảnh thông qua AI để nhận diện vật thể theo hình dạng hoặc màu sắc và gấp vật thể được nhận diện, đây là một bài toán phân loại vật thể khá điển hình trong hầu hết các Smart Factory.

5.2.4 Hoàn thiện DataCenter kết nối đồng bộ dữ liệu nhiều thiết bị

Cuối cùng là hoàn thiện lại module DataCenter cho phép nhiều thiết bị kết nối và đồng bộ dữ liệu, giúp ta có thể thiết lập một chuỗi hoạt động dây chuyền của các thiết bị khác nhau, ví dụ như sau khi cánh tay phân loại sẽ đưa vật thể lên băng chuyền, dữ liệu từ DataCenter sẽ giúp băng chuyền đưa vật thể tới nơi được quy định tương ứng.

Tham khảo

- [1] B. Chen, J. Wan, L. Shu, P. Li, M. Mukherjee, and B. Yin, “Smart factory of industry 4.0: Key technologies, application case, and challenges,” *IEEE Access*, vol. 6, pp. 6505–6519, 2018. DOI: 10.1109/ACCESS.2017.2783682.
- [2] SAP. “What is a smart factory? | sap insights.” (), [Online]. Available: <https://www.sap.com/insights/what-is-a-smart-factory.html>.
- [3] TWI. “What is a smart factory? (a complete guide) - twi.” (), [Online]. Available: <https://www.twi-global.com/technical-knowledge/faqs/what-is-a-smart-factory>.

THÔNG TIN SINH VIÊN

Danh sách tác giả Đồ Án:

1. **Trần Ngọc Cát** - ID: 1912750

- Số điện thoại: (84)975.598.049
- Email: cat.tran03@hcmut.edu.vn

2. **Diệp Trần Nam** - ID: 1914213

- Số điện thoại: (84)899.939.802
- Email: nam.diep.239@hcmut.edu.vn

3. **Nguyễn Văn Trọng** - ID: 1915677

- Số điện thoại: (84)708375003
- Email: trong.nguyen2492001@hcmut.edu.vn

