

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA MECÂNICA

MATHEUS ALVES DE PAULA

Desenvolvimento de um aplicativo mobile para controlar bancada eletropneumática via
protocolo MQTT usando Raspberry Pi 4 e ESP32

Uberlândia

2025

MATHEUS ALVES DE PAULA

Desenvolvimento de um aplicativo mobile para controlar bancada eletropneumática via protocolo MQTT usando Raspberry Pi 4 e ESP32

Trabalho de Conclusão de Curso apresentado à Faculdade de Engenharia Mecânica da Universidade Federal de Uberlândia como requisito parcial para obtenção do título de bacharel em Engenharia Mecatrônica.

Área de concentração: Engenharia Mecatrônica

Orientador: Prof. Dr. José Jean-Paul Zanlucchi de Souza Tavares

Uberlândia

2025

Ficha Catalográfica Online do Sistema de Bibliotecas da UFU
com dados informados pelo(a) próprio(a) autor(a).

P324 2025	Paula, Matheus Alves de, 1995- Desenvolvimento de um aplicativo mobile para controlar bancada eletropneumática via protocolo MQTT usando Raspberry Pi 4 e ESP32 [recurso eletrônico] / Matheus Alves de Paula. - 2025. Orientador: José Jean-Paul Zanlucchi de Souza Tavares. Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Uberlândia, Graduação em Engenharia Mecatrônica. Modo de acesso: Internet. Inclui bibliografia. Inclui ilustrações. 1. Mecatrônica. I. Tavares, José Jean-Paul Zanlucchi de Souza, 1972-, (Orient.). II. Universidade Federal de Uberlândia. Graduação em Engenharia Mecatrônica. III. Título. CDU: 621.03
--------------	---

Bibliotecários responsáveis pela estrutura de acordo com o AACR2:

Gizele Cristine Nunes do Couto - CRB6/2091
Nelson Marcos Ferreira - CRB6/3074

MATHEUS ALVES DE PAULA

Desenvolvimento de um aplicativo mobile para controlar bancada eletropneumática via protocolo MQTT usando Raspberry Pi 4 e ESP32

Trabalho de Conclusão de Curso apresentado à Faculdade de Engenharia Mecatrônica da Universidade Federal de Uberlândia como requisito parcial para obtenção do título de bacharel em Engenharia Mecatrônica.

Área de concentração: Engenharia Mecatrônica

Uberlândia, 08/05/2025.

Banca Examinadora:

Prof. Dr. José Jean-Paul Zanlucchi de Souza Tavares

Prof. Dr. Roberto Mendes Finzi

Prof. Ms. Werley Rocherter Borges Ferreira

Dedico este trabalho aos meus pais, pelo
estímulo, carinho e compreensão.

AGRADECIMENTOS

Agradeço ao professor Prof. Dr. José Jean-Paul Zanlucchi de Souza Tavares pelo suporte ao longo do desenvolvimento deste trabalho com ideias, incentivo, motivação, orientação e disponibilidade para sanar todas as minhas dúvidas, ao Laboratório de Planejamento Automático de Manufatura, pelo espaço e pelos materiais cedidos, e ao Laboratório de Ensino de Mecatrônica, pelo espaço disponibilizado durante os testes em bancada.

Agradeço também aos colegas de graduação que frequentaram os laboratórios de ensino ao longo do projeto, em especial ao Álisson Carvalho Vasconcelos e ao Walter Barbosa Soares Júnior, pelo apoio durante os testes em bancada realizados no Laboratório de Ensino de Mecatrônica.

À minha mãe, agradeço pela criação, pelo amor, pelo carinho e pelos ensinamentos ao longo da vida. Esse trabalho não seria conseguível sem a sua ajuda. Agradeço ainda ao meu pai, ao meu irmão e aos meus amigos por todo o apoio e conselhos durante o período de graduação.

Agradeço, em especial, à minha parceira pelo apoio e incentivo nos momentos mais difíceis; suas conversas e ideias foram fundamentais para a conclusão bem-sucedida deste projeto.

“Ninguém ignora tudo. Ninguém sabe tudo. Todos nós sabemos alguma coisa. Todos nós ignoramos alguma coisa.”

(FREIRE, 2002, p. 69)

RESUMO

A pandemia da COVID-19 impossibilitou a realização de diversas tarefas devido às medidas de segurança. Dentre essas atividades, temos as aulas práticas de Automação Industrial da Universidade Federal de Uberlândia utilizando bancada eletropneumática, os alunos ficaram prejudicados por não terem oportunidade de testar os modelos de sequências eletropneumáticas ensinados em aulas teóricas, nas aulas práticas. Entretanto com o desenvolvimento da Indústria 4.0, temos uma arquitetura que pode ser utilizada para resolver problemas como esse, que é a *Industrial Internet of Things* (IIoT). A IIoT permite conectar dispositivos, máquinas e sensores em tempo real, trazendo maior eficiência e aumento da produtividade em variados cenários. Uma das ferramentas para a realização dessa aplicação é o protocolo *Message Queuing Telemetry Transport* (MQTT), que nos permite realizar a comunicação entre os dispositivos conectados à rede. Assim, este projeto apresenta uma solução para realização das aulas práticas controlando a bancada eletropneumática remotamente através de um aplicativo de celular, apresentando várias tecnologias que podem ser adaptadas em ambientes além da universidade. A arquitetura desse projeto é composta de microcontroladores ESP32 para o controle das válvulas 5/2 de simples solenoide e retorno por mola da bancada eletropneumática, microcomputador Raspberry para a hospedagem do broker local MQTT, sensores de fim de curso que enviam as informações de estado dos atuadores e aplicativo mobile para dispositivos Android.

Palavras-chave: Indústria 4.0; IIoT; MQTT; microcontrolador ESP32; Mecatrônica.

ABSTRACT

The COVID-19 pandemic made it impossible to carry out several tasks due to the safety measures. Among those activities, we have the practical classes of Industrial Automation at the Federal University of Uberlândia using an electro-pneumatic bench, the students were hindered because they had no opportunity to test the models of electro-pneumatic sequences, taught in theoretical classes, in the practice classes. However with the development of Industry 4.0, we have an architecture that can be used to solve problems like this, which is the *Industrial Internet of Things* (IIoT). The IIoT allows connecting devices, machines, and sensors in real time, bringing greater efficiency and increased productivity in various scenarios. One of the tools for carrying out this application is the *Message Queuing Telemetry Transport* (MQTT) protocol, which allows us to carry out communication between the devices connected to a network. Thus, this project presents a solution for carrying out practical classes by controlling the electro-pneumatic bench remotely through a cellphone application, presenting several technologies that can be adapted in environments beyond the university. The architecture of this project is composed of ESP32 microcontrollers to control the 5/2 valves with simple solenoid and spring return of the electro-pneumatic bench, Raspberry microcomputer to host the local MQTT broker, limit sensors that send the status information of the actuators and mobile application for Android devices.

Keywords: Industry 4.0; IIoT; MQTT; ESP32 microcontroller; Mechatronics.

LISTA DE ILUSTRAÇÕES

Figura 1 – Representação do número de posições.....	15
Figura 2 – Representação do número de vias	15
Figura 3 – Válvula 5 vias por 2 posições	19
Figura 4 – Representação em corte do atuador pneumático de simples ação.....	20
Figura 5 – Simbologia para o atuador pneumático de simples ação	20
Figura 6 – Representação em corte do atuador pneumático de dupla ação	21
Figura 7 – Simbologia para o atuador pneumático de dupla ação.....	21
Figura 8 – Atuador pneumático de dupla ação	21
Figura 9 – Sensor de fim de curso com acionamento por rolete	22
Figura 10 – Diagrama eletropneumático pelo método cascata.....	25
Figura 11 – Atuadores pneumáticos e sensores fins de curso	26
Figura 12 – Diagrama trajeto-passo	26
Figura 13 – Diagrama de acionamento dos sensores e atuadores.....	27
Figura 14 – Diagrama eletropneumático pelo método trajeto-passo	28
Figura 15 – Cabeçalho do protocolo MQTT	29
Figura 16 – Modelo de publicação e subscrição no MQTT	30
Figura 17 – Pinagem ESP32.....	31
Figura 18 – Diagrama em blocos do chip ESP32.....	32
Figura 19 – Raspberry Pi Modelo B.....	33
Figura 20 – Interface do Android Studio.....	34
Figura 21 – Desenho esquemático da arquitetura do projeto	36
Figura 22 – Interface do Balena Etcher	38
Figura 23 – Interface do Win32 Disk Imager.....	39
Figura 24 – Interface de instalação do sistema operacional do Raspberry Pi	39
Figura 25 – Arquivo de configuração dhcpcd	40
Figura 26 – Arquivo de configuração interfaces	41
Figura 27 – Arquivo de configuração Mosquitto	42
Figura 28 – Estado de funcionamento do Mosquitto.....	42
Figura 29 – Definição de bibliotecas.....	44
Figura 30 – Definição dos parâmetros de conexão Wi-Fi e MQTT	44
Figura 31 – Protótipos das funções	45
Figura 32 – Definição dos pinos para os sensores fim de curso.....	46
Figura 33 – Protótipo das funções	46

Figura 34 – Trecho de código XML para a tela inicial	48
Figura 35 – Tela inicial do aplicativo	49
Figura 36 – Trecho de código Java para a tela inicial	50
Figura 37 – Introduções da biblioteca MQTT no código Java	51
Figura 38 – Subscrição de tópicos em Java	51
Figura 39 – Tela do aplicativo no modo de controle para dois atuadores	52
Figura 40 – Código Java para as funções de avanço e recuo do atuador A.....	53
Figura 41 – Tela do aplicativo no modo de comando para dois atuadores	54
Figura 42 – Tela do aplicativo no modo de dashboard para dois atuadores.....	54
Figura 43 – Bancada eletropneumática	56
Figura 44 – Esquema da ligação do ESP32 com o módulo relé.....	57
Figura 45 – Montagem da ligação do ESP32 com o módulo relé	57
Figura 46 – Esquema da ligação do ESP32 com os sensores de fim de curso	58
Figura 47 – Montagem da ligação do ESP32 com os sensores de fim de curso	58
Figura 48 – Preenchimentos das equações para a sequência A+ B+ A– B–	60
Figura 49 – Sequência A+ B+ A– B– em andamento	60
Figura 50 – Sequência A+ B+ B– A– em andamento	61
Figura 51 – Sequência A+ A– B+ B– em andamento	62
Figura 52 – Sequência A+B+C+C–B–A– em andamento.....	62
Figura 53 – Aplicação em execução no modo “comando”	63
Figura 54 – Aplicação em execução no modo “dashboard”	63

LISTA DE TABELAS

Tabela 1 – Identificação dos orifícios das válvulas pneumáticas	16
Tabela 2 – Exemplos de simbologia para válvulas.....	18
Tabela 3 – Principais tipos de cilindros.....	19
Tabela 4 – Quadro de sequência dos acionamentos	25
Tabela 5 – Tópicos dos microcontroladores das válvulas	45
Tabela 6 – Tópicos do microcontrolador dos sensores de fim de curso.....	47
Tabela 7 – Tópicos do aplicativo.....	55
Tabela 8 – Equações para as sequências eletropneumáticas	59

LISTA DE ABREVIATURAS E SIGLAS

MQTT	Transporte de Filas de Mensagem de Telemetria (em inglês <i>Message Queuing Telemetry Transport</i>)
IoT	Internet das Coisas (em inglês <i>Internet of Things</i>)
IIoT	Internet Industrial das Coisas (em inglês <i>Industrial Internet of Things</i>)
PFC	Projeto de Fim de Curso
UFU	Universidade Federal de Uberlândia
CETOP	Comissão Européia de Transmissões Óleos-Hidráulicos e Pneumáticas
CLP	Controlador Lógico Programável
CTI	Centro de Tecnologia da Informação
LEM	Laboratório de Ensino de Mecatrônica
MAPL	Laboratório de Planejamento Automático de Manufatura (em inglês <i>Manufacturing Automation Planning Lab</i>)
GND	Terra (em inglês <i>Ground</i>)
VDC	Tensão de corrente contínua (em inglês <i>Volts Direct Current</i>)
GPIO	Entrada/saída de propósito geral (em inglês <i>General Purpose Input/Output</i>)
IHM	Interface Homem-Máquina

SUMÁRIO

1	INTRODUÇÃO.....	12
1.1	OBJETIVOS	13
1.1.1	OBJETIVO GERAL	13
1.1.2	OBJETIVOS ESPECÍFICOS	13
1.1.3	JUSTIFICATIVA DO PROJETO	13
2	REVISÃO BIBLIOGRÁFICA	14
2.1	SISTEMAS ELETROPNEUMÁTICOS	14
2.1.1	VÁLVULAS DE COMANDO.....	14
2.1.2	ATUADORES	19
2.1.3	SENSORES	22
2.2	MÉTODOS SEQUENCIAIS	23
2.2.1	MÉTODO INTUITIVO.....	23
2.2.2	MÉTODO CASCATA.....	24
2.2.3	MÉTODO TRAJETO PASSO	25
2.3	PROTOCOLO MQTT	28
2.4	MICROCONTROLADOR ESP32.....	30
2.5	MICROCOMPUTADOR RASPBERRY	32
2.6	ANDROID STUDIO	33
3	DESENVOLVIMENTO.....	35
3.1	ARQUITETURA DO PROJETO	35
3.2	MATERIAIS E PROGRAMAS UTILIZADOS	37
3.3	CONFIGURAÇÕES REQUERIDAS	37
3.3.1	INSTALAÇÃO DO SISTEMA OPERACIONAL RASPBERRY PI.....	38
3.3.2	DEFININDO IP FIXO PARA O RASPBERRY PI	40
3.3.3	IMPLEMENTAÇÃO DO MQTT NO RASPBERRY PI.....	41
3.4	CÓDIGOS DOS MICROCONTROLADORES ESP32.....	43
3.5	CÓDIGO DO APLICATIVO	47
4	RESULTADOS	56
5	CONCLUSÃO.....	64
6	TRABALHOS FUTUROS	65
	REFERÊNCIAS.....	66
7	APÊNDICE A – CÓDIGO DE CONFIGURAÇÃO DHCPD	68

8	APÊNDICE B – CÓDIGO DE CONFIGURAÇÃO INTERFACES	70
9	APÊNDICE C – CÓDIGO DE CONFIGURAÇÃO MOSQUITTO	71
10	APÊNDICE D – CÓDIGO ESP32 DA VÁLVULA A.....	72
11	APÊNDICE E – CÓDIGO ESP32 DA VÁLVULA B	80
12	APÊNDICE F – CÓDIGO ESP32 DA VÁLVULA C	88
13	APÊNDICE G – CÓDIGO ESP32 DOS SENSORES	96
14	APÊNDICE H – CÓDIGO XML DA PRIMEIRA TELA.....	106
15	APÊNDICE I – CÓDIGO JAVA DA PRIMEIRA TELA.....	113

1 INTRODUÇÃO

Com a pandemia da COVID-19 que ocorreu durante os anos 2020 e 2021, as aulas práticas das disciplinas que utilizam a bancada eletropneumática como Automação Industrial, Redes Industriais e Sistemas de Controle Hidráulicos e Pneumáticos, foram impossibilitadas de serem realizadas devido às medidas de segurança. Assim, os alunos puderam realizar apenas simulações em softwares de ensino, o que comprometeu o aprendizado, já que a experiência prática com a bancada oferece situações que os softwares não conseguem reproduzir.

Diante desse contexto, este projeto propõe uma solução aplicável ao ensino à distância. Além disso, as tecnologias utilizadas no projeto podem ser incorporadas ao plano de ensino atual, uma vez que é essencial que o engenheiro mecatrônico esteja atualizado com os avanços tecnológicos, especialmente no que se refere à Internet das Coisas Industrial e à Indústria 4.0.

A Indústria 4.0 inaugurou uma nova era na manufatura e nos processos produtivos, na qual tecnologias embarcadas e sistemas ciber-físicos se unem para criar fábricas inteligentes. Ao integrar dispositivos conectados, esse conceito rompe com a lógica tradicional de produção centralizada, remetendo à Internet das Coisas (IoT).

Nesse contexto, o protocolo de comunicação *Message Queuing Telemetry Transport* (MQTT) surge como elemento-chave para viabilizar a comunicação eficiente entre os diversos componentes da indústria, dando origem ao que vem sendo denominado Internet Industrial das Coisas (IIoT). Sua leveza, baixo consumo de energia e facilidade de implementação permitem que microcontroladores, como o ESP32, e microcomputadores, como o Raspberry Pi, comuniquem-se de forma confiável. Assim, o MQTT estabelece a base para aplicações robustas, escaláveis e com resposta em tempo real.

A Indústria 4.0, portanto, não é apenas uma evolução tecnológica, mas uma revolução na maneira de projetar, entender e gerenciar todo o ciclo produtivo. Ao combinar automação com controle remoto, abre-se espaço para modelos de negócio inovadores. Além disso, essa tecnologia é ferramenta poderosa para o ensino a distância. Este relatório explora esses conceitos e demonstra, por meio de um protótipo com ESP32 e Raspberry Pi, como as práticas da Indústria 4.0 podem ser aplicadas e aprimoradas, otimizando processos industriais e qualificando ferramentas de ensino.

1.1 OBJETIVOS

1.1.1 OBJETIVO GERAL

O objetivo geral deste Projeto de Fim de Curso (PFC) é aplicar a tecnologia IIoT às aulas práticas de Automação Industrial, visando avaliar a viabilidade de realização das aulas práticas de forma remota.

1.1.2 OBJETIVOS ESPECÍFICOS

Além do objetivo geral, foram definidos objetivos específicos, como o estudo do protocolo de comunicação MQTT, responsável pela comunicação entre os dispositivos móveis e os microcontroladores ESP32 conectados aos mecanismos da bancada; a configuração e inicialização dos serviços de mensagens MQTT no Raspberry Pi 4, para que este funcione como broker local na rede da Universidade Federal de Uberlândia (UFU); a ambientação e programação do microcontrolador ESP32; a ambientação e programação do software Android Studio para o desenvolvimento do aplicativo para dispositivos móveis; a revisão dos métodos sequenciais abordados na disciplina de Automação Industrial, com ênfase no método sequencial trajeto-passo, cujas equações serão utilizadas para controlar os atuadores eletropneumáticos da bancada; e a aplicação dessas tecnologias da Indústria 4.0 como ferramenta pedagógica.

1.1.3 JUSTIFICATIVA DO PROJETO

A justificativa deste projeto fundamenta-se na base teórica e prática adquirida ao longo das disciplinas cursadas: em Automação Industrial, consolidaram-se os conceitos de métodos sequenciais; em Sistemas de Controle Hidráulicos e Pneumáticos, interiorizou-se o funcionamento das válvulas e dos atuadores utilizados na bancada; em Redes Industriais, aprendeu-se a estabelecer comunicações seguras e confiáveis entre dispositivos; em Algoritmo e Programação de Computadores, desenvolveu-se a lógica e as estruturas de código da linguagem de programação C/C++; em Eletrônica Básica, adquiriu-se o entendimento de circuitos e componentes eletrônicos; em Sistemas Digitais para Mecatrônica, compreendeu-se a programação em ambiente Linux; e, finalmente, em Sistemas Operacionais e Arquitetura de Redes de Computadores, obteve-se o conhecimento dos conceitos sobre sistemas operacionais, redes e protocolos de comunicação.

2 REVISÃO BIBLIOGRÁFICA

Esta seção apresenta os mecanismos e conceitos teóricos utilizados no desenvolvimento do projeto, visando facilitar a compreensão ao conectar esses elementos que darão forma à arquitetura do projeto, que será discutida na próxima seção.

2.1 SISTEMAS ELETROPNEUMÁTICOS

De uma forma mais geral, define-se pneumática como sendo o conjunto de todas as aplicações que utilizam a energia armazenada e transmitida pelo ar comprimido. Mais especificamente, na engenharia industrial, a pneumática é a realização técnica de acionamentos lineares ou rotativos, através da utilização de atuadores pneumáticos, com os seus respectivos elementos de comando, de sinal e de processamento de sinal (Bollmann, 1997).

Os sistemas eletropneumáticos têm se consolidado como uma solução robusta e versátil na automação industrial, integrando o uso do ar comprimido com componentes eletrônicos para o acionamento preciso de atuadores. Junto de outras tecnologias, consegue-se desenvolver e implementar diversos modelos e métodos que são capazes de serem aplicados na área de ensino, aperfeiçoando o ensino e a qualificação do futuro engenheiro, e na área da indústria, atendendo requisições das linhas de produção.

Os microprocessadores, microcontroladores e controladores programáveis, a preços cada vez mais competitivos, estão substituindo gradativamente os elementos de processamento de sinal pneumáticos e os relés da eletropneumática, provocando consequentemente também o crescente uso dos sensores elétricos ou eletrônicos como elementos de sinal. Essa moderna combinação dos atuadores e suas válvulas de comando pneumáticos com a microeletrônica e a informática, reúne as vantagens típicas de cada uma dessas tecnologias, acarretando seu uso crescente na automação industrial (Bollmann, 1997).

Nos tópicos a seguir, serão detalhados os componentes mencionados, os quais constituem a base dos sistemas eletropneumáticos.

2.1.1 VÁLVULAS DE COMANDO

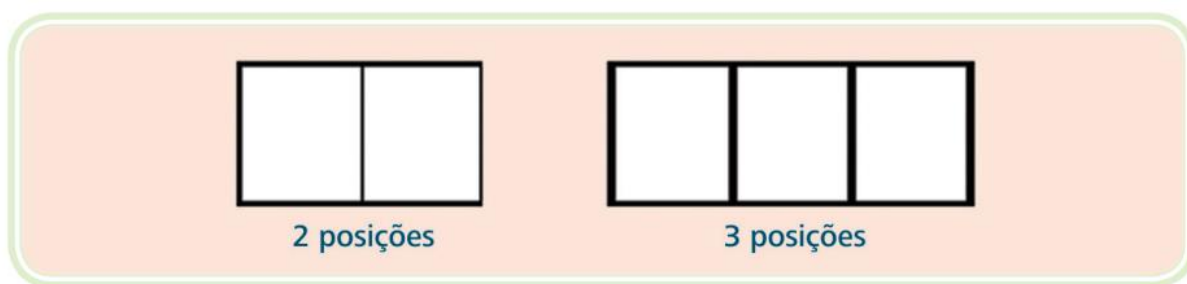
São todas as válvulas que, ao receberem um impulso pneumático, mecânico, ou elétrico, permitem que haja fluxo de ar pressurizado para alimentar determinado(s) elemento(s) do automatismo. Também são válvulas de comando, as que permitem controlar o fluxo do ar para os diversos elementos do sistema, mediante ajuste mecânico ou elétrico, as que permitem o

fluxo em apenas um sentido, os elementos lógicos, as controladoras de pressão e as temporizadas (Fialho, 2012).

Para caracterizar uma válvula direcional, devemos conhecer: número de posições, número de vias, vazão, tipo de acionamento (comando), tipo de retorno e tipo construtivo da válvula (Pavani, 2011).

Para os desenhos dos projetos eletropneumáticos, se fez necessário criar representações esquemáticas para as válvulas, sendo que essas representações simulam o funcionamento interno das mesmas. As válvulas direcionais são representadas por um retângulo, sendo que este retângulo é dividido em quadrados. O número de quadrados é referente ao número de posições da válvula, sendo este que indica a quantidade de movimentos que a válvula executa através de acionamentos. A Figura 1 apresenta o esquema para a representação do número de posições.

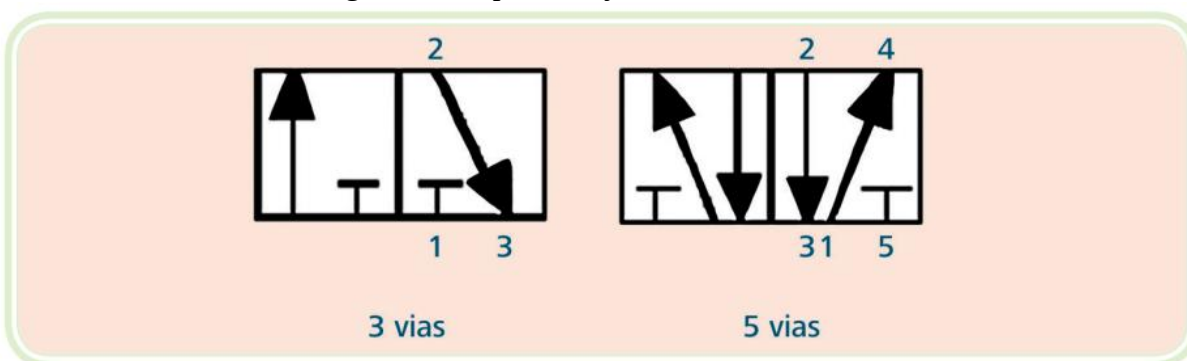
Figura 1 – Representação do número de posições



Fonte: Pavani (2011, p. 79).

O número de vias é referente ao número de conexões que a válvula possui, sendo que os possíveis tipos de conexões consideradas como vias são: a conexão de entrada de pressão, as conexões de utilização e as conexões de escape. A Figura 2 apresenta o esquema para a representação do número de vias.

Figura 2 – Representação do número de vias



Fonte: Pavani (2011, p. 79).

Uma regra prática para a determinação do número de vias consiste em separar um dos quadros (posição) e verificar quantas vezes o(s) símbolo(s) interno(s) toca(m) os lados do quadrado, obtendo-se assim, o número de orifícios e em correspondência o número de vias (Pavani, 2011).

Em 1976, a Comissão Européia de Transmissões Óleos-Hidráulicos e Pneumáticas (CETOP) propôs um método universal de identificação dos orifícios das válvulas aos fabricantes de equipamentos pneumáticos. A finalidade da codificação é fazer com que o usuário tenha uma fácil instalação de componentes, relacionando as marcas dos orifícios no circuito, com as marcas contidas nas válvulas. A CETOP propõe identificação numérica (Pavani, 2011).

A Tabela 1 apresenta a identificação dos orifícios das válvulas pneumáticas com base nas principais normas seguidas no Brasil.

Tabela 1 – Identificação dos orifícios das válvulas pneumáticas

Orifício	Norma DIN 24300			Norma ISO 1219		
Pressão	P			1		
Utilização	AA	BB	C	2	4	6
Escape	RR	SS	T	3	5	7
Pilotagem	XX	YY	Z	10	12	14

Fonte: Pavani (2011, p. 80).

Com base na tabela, os números relacionados à norma ISO 1219 significam:

- **Nº 1** - alimentação: orifício de suprimento principal (pressão).;
- **Nº 2** - utilização, saída: orifício de aplicação em válvulas de 2/2, 3/2 e 3/3;
- **Nºs 2 e 4** - utilização, saída: orifícios de aplicação em válvulas 4/2, 4/3, 5/2 e 5/3. A via 4 normalmente faz o avanço e a 2 o retorno do cilindro;
- **Nº 3** - escape ou exaustão: orifícios de liberação do ar utilizado em válvulas 3/2, 3/3, 4/2 e 4/3;
- **Nºs 3 e 5** - escape ou exaustão: orifício de liberação do ar utilizado em válvulas 5/2 e 5/3.

Os orifícios de pilotagem são identificados da seguinte forma: 10, 12 e 14. Estas referências baseiam-se na identificação do orifício de alimentação 1.

- **Nº 10** - indica um orifício de pilotagem que, ao ser influenciado, isola, bloqueia, o orifício de alimentação;
- **Nº 12** - liga a alimentação 1 com o orifício de utilização 2, quando ocorrer o comando;
- **Nº 14** - comunica a alimentação 1 com o orifício de utilização 4, quando ocorrer a pilotagem.

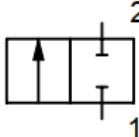
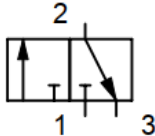
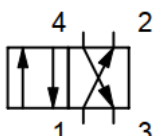
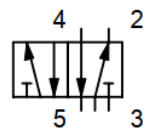
A norma DIN 24300 utiliza uma forma literal para a identificação das vias. As letras relacionadas à mesma significam:

- **A, B, C** - linha de trabalho (utilização);
- **P** - Conexão de pressão (alimentação);
- **R, S, T** - escape ao exterior do ar comprimido utilizado pelos equipamentos pneumáticos (escape, exaustão);
- **L** - drenagem de líquido;
- **X, Y, Z** - linha para transmissão da energia de comando (linhas de pilotagem).

Além dessas letras que são encontradas na tabela, de acordo com as normas da *National Fire Protection Association* (NFPA), os escapes também podem ser representados pela letra E, seguida da respectiva letra que identifica a utilização. Por exemplo, EA significa que os orifícios em questão são a exaustão do ponto de utilização A ou EB significa que o escape do ar utilizado é pelo orifício B. Ademais, a letra D, quando utilizada, representa orifício de escape do ar de comando interno.

Na simbologia das válvulas, além das representações dos números de posições e de vias, constam as setas que indicam a direção e o sentido do fluxo de ar, bem como os T's que indicam o bloqueio das vias. Com base nesses conceitos e nos apresentados anteriormente, alguns exemplos de válvulas são apresentados na Tabela 2.

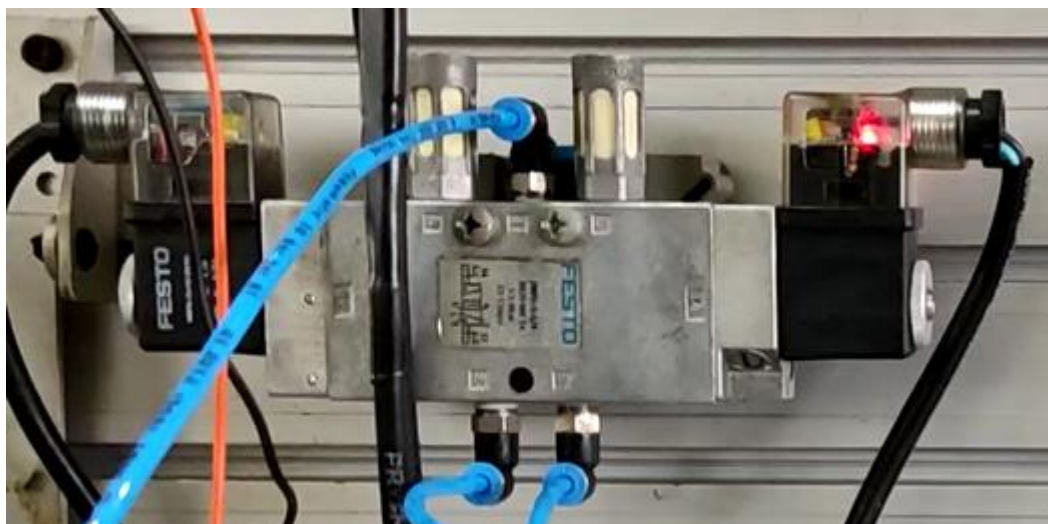
Tabela 2 – Exemplos de simbologia para válvulas

Simbologia	Designação	Descrição
	2/2	2 vias por 2 posições normalmente aberta
	3/2	3 vias por 2 posições normalmente fechado
	4/2	4 vias por 2 posições
	5/2	5 vias por 2 posições

No funcionamento das válvulas, sabe-se que as mesmas requerem uma ação para efetuar mudança de posição e uma outra ação para voltarem ao estado (posição) inicial. Para isso, temos o tipo de comando de retorno, sendo que estes podem ser mecânicos, elétricos ou combinados. Os retornos mecânicos são: retorno por mola, no qual a válvula retorna à posição inicial por meio da energia liberada pela mola, previamente armazenada pela compressão, e retorno por trava, em que a trava é o mecanismo que mantém a válvula na posição de manobra, exigindo uma ação para que ela volte à posição inicial, sendo esse tipo de retorno utilizado junto a acionamentos musculares. Nos retornos elétricos, a operação das válvulas é realizada por meio de sinais elétricos, gerados por sensores de fim de curso, pressostatos, botoeiras, relés, temporizadores, além de microcontroladores, Controlador Lógico Programável (CLP) e sensores de proximidade, como os indutivos, capacitivos e ópticos. Já nos retornos combinados, temos a utilização da própria energia do ar comprimido para acionar as válvulas, podendo comunicar o ar de alimentação da válvula a um acionamento auxiliar que permite a ação do ar sobre o comando de válvula.

A Figura 3 apresenta uma válvula de comando pneumática de 5 vias por 2 posições que foi utilizada no projeto.

Figura 3 – Válvula 5 vias por 2 posições



2.1.2 ATUADORES

Os atuadores pneumáticos são elementos mecânicos que por meio de movimentos lineares ou rotativos transformam a energia cinética gerada pelo ar pressurizado e em expansão, em energia mecânica, produzindo trabalho (Fialho, 2012). Eles são responsáveis pelas ações físicas necessárias para o funcionamento do sistema, como mover, posicionar ou controlar componentes mecânicos. No âmbito dos atuadores temos os cilindros de simples ação e cilindros de dupla ação. A Tabela 3 apresenta um resumo sobre os principais tipos de cilindros.

Tabela 3 – Principais tipos de cilindros

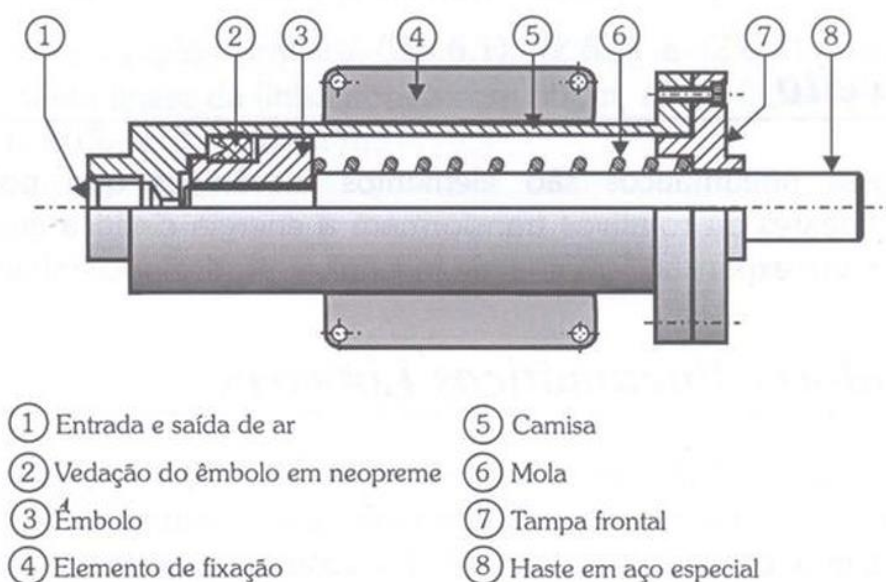
Tipo	Aspecto construtivo	Retorno
Cilindros de ação simples	De êmbolo e haste	Retorno por mola ou por força externa
	De membrana e haste	
	De membrana	
Cilindros de ação dupla	Com haste	Simples
		Dupla
	Sem haste	De cabo
		De cinta (ou tira)
		Magnético

Fonte: Bollmann (1997, p. 65).

Os atuadores de simples ação são atuadores cujo movimento de retração ou expansão é feito pela ação de uma mola interna ao tubo cilíndrico "camisa", podendo ainda ter retorno por

força externa. São normalmente aplicados em dispositivos de fixação, gavetas de moldes de injeção, expulsão, prensagem, elevação e alimentação de componentes (Fialho, 2012). A Figura 4 mostra o desenho de uma representação em corte do atuador de simples ação indicando os seus principais componentes.

Figura 4 – Representação em corte do atuador pneumático de simples ação



Fonte: Fialho (2012, p. 78).

A Figura 5 apresenta a representação simbólica para este tipo de atuador.

Figura 5 – Simbologia para o atuador pneumático de simples ação

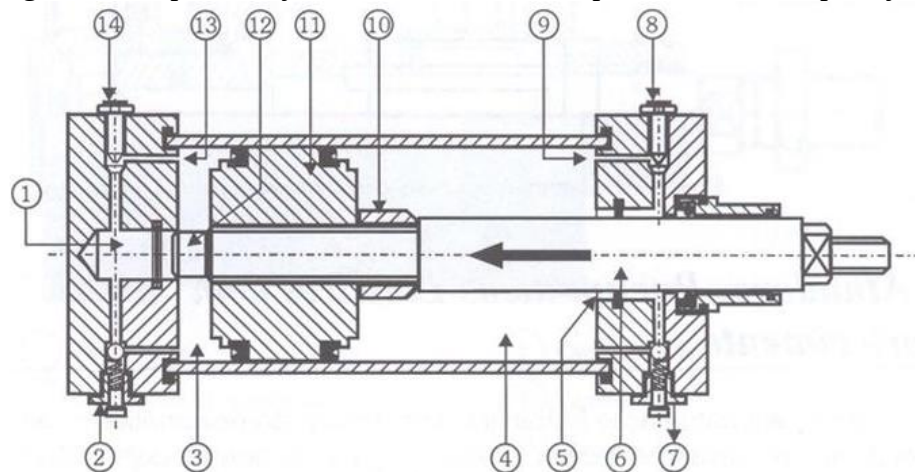


Figura 3.2 - Representação simbólica normalizada - (a) atuador linear de simples efeito normalmente retraído com retorno por mola - (b) atuador linear de simples efeito normalmente distendido com retorno por mola.

Fonte: Fialho (2012, p. 79).

Já os atuadores de dupla ação são atuadores em que a alimentação e exaustão ocorrem por conexões localizadas em ambas extremidades do atuador (Fialho, 2012). A Figura 6 mostra o desenho de uma representação em corte do atuador de dupla ação indicando os seus principais componentes.

Figura 6 – Representação em corte do atuador pneumático de dupla ação

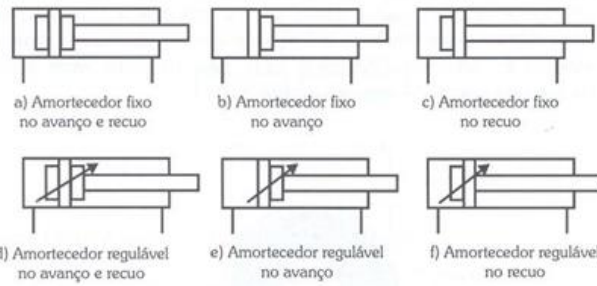


- | | |
|-----------------------------|---|
| ① Caverna traseira | ⑧ Conexão alimentação/exaustão c/ reg. de amortecimento |
| ② Conj. válvula de retenção | ⑨ Orifício de saída para amortecimento no avanço |
| ③ Câmara traseira | ⑩ Bucha amortecedora |
| ④ Câmara frontal | ⑪ Êmbolo |
| ⑤ Caverna frontal | ⑫ Ponta amortecedora da haste |
| ⑥ Haste | ⑬ Orifício de saída para amortecimento no retorno |
| ⑦ Conj. válvula de retenção | ⑭ Conexão alimentação/exaustão c/reg. de amortecimento |

Fonte: Fialho (2012, p. 82).

A Figura 7 apresenta a representação simbólica para o atuador pneumático de dupla ação.

Figura 7 – Simbologia para o atuador pneumático de dupla ação



Fonte: Fialho (2012, p. 83).

A Figura 8 apresenta um atuador pneumático de dupla ação utilizado no projeto.

Figura 8 – Atuador pneumático de dupla ação



2.1.3 SENSORES

Os sensores são dispositivos responsáveis por monitorar variáveis de processo dentro do sistema de controle, fornecendo informações por meio de seu sinal de saída. Dentre os tipos de sensores, destacam-se os sensores elétricos, os sensores fotoelétricos e os sensores eletromecânicos.

Para cada tipo, existe uma variedade de opções. Por exemplo, os sensores elétricos podem ser de proximidade, sendo estes capazes de detectar a presença de elementos de máquina, objetos da linha de produção, embalagens para codificação, fluidos, entre outros. Eles ainda podem ser classificados em sensores indutivos (para a detecção de materiais metálicos) e sensores capacitivos (para a detecção de materiais orgânicos). Além disso, entre os sensores elétricos, também se destaca o pressostato, um dispositivo que converte um pulso pneumático em um pulso elétrico.

Quanto aos sensores ópticos, existem duas opções principais: o sensor por reflexão, que detecta a posição pela luz que retorna a um fotossensor, e o sensor por interrupção, no qual a luz emitida é captada por um fotossensor alinhado, que detecta a presença da peça quando esta intercepta o feixe de luz.

Por fim, os sensores eletromecânicos incluem relés e sensores de fim de curso, que, quando acionados, enviam um sinal elétrico. No presente projeto, utilizou-se o sensor de fim de curso para identificar as posições (avançado ou recuado) dos atuadores pneumáticos. Nesse caso, o sensor possui um rolete que é acionado quando o atuador realiza o contato. A Figura 9 apresenta este tipo de sensor que foi utilizado no projeto.

Figura 9 – Sensor de fim de curso com acionamento por rolete



2.2 MÉTODOS SEQUENCIAIS

No Módulo 1 de Automação Industrial, que trata dos fundamentos da eletropneumática, apresentam-se métodos didáticos para a resolução de problemas que envolvem sequências de comandos dos atuadores em diversas aplicações. Nos tópicos a seguir, serão discutidos os conceitos e a aplicação desses métodos.

2.2.1 MÉTODO INTUITIVO

O método intuitivo é indicado para pequenos circuitos, porém requer experiência em projetos eletropneumáticos. Para utilizá-lo, deve-se verificar se a sequência do circuito é direta. Chamamos de sequência direta aquela em que, ao dividirmos a sequência ao meio e compararmos os lados direito e esquerdo, desconsiderando os sinais, ambos os lados são exatamente iguais.

Para identificar se a sequência é direta ou indireta, deve-se, inicialmente, escrevê-la de forma abreviada, conforme o exemplo a seguir, em que as letras “A” e “B” representam os atuadores e os sinais “+” e “–” indicam os movimentos de avanço e de recuo, respectivamente.

- A+ B+ A– B– (Sequência I)
- A+ A– B+ B– (Sequência II)
- A+ B+ B– A– (Sequência III)
- A+ C+ B– A– C– B+ (Sequência IV)

O próximo passo é inserir um traço vertical que divida a sequência exatamente ao meio, conforme mostrado a seguir.

- A+ B+ | A– B– (Sequência I)
- A+ A– | B+ B– (Sequência II)
- A+ B+ | B– A– (Sequência III)
- A+ C+ B– | A– C– B+ (Sequência IV)

Por fim, analisa-se se os dois lados do traço possuem as mesmas letras na mesma ordem; se sim, trata-se de uma sequência direta, cujo circuito de comando pode ser construído facilmente pelo método intuitivo. No exemplo demonstrado, somente as sequências I e IV representam sequências diretas.

As sequências diretas têm resolução mais simples pelo método intuitivo, pois este é o único que não obedece a uma regra específica e as sequências diretas não apresentam o

problema de sobreposição de sinais. Por depender do pensamento e do raciocínio de quem o emprega, esse método oferece soluções distintas para um mesmo problema.

2.2.2 MÉTODO CASCATA

O método cascata foi criado para evitar o problema da sobreposição de sinais; assim, pode resolver tanto sequências diretas quanto indiretas. No entanto, é mais adequado utilizar o método intuitivo nos casos de sequências diretas. Alguns aspectos desse método são: limita-se a circuitos de porte médio; é inadequado em circuitos compostos; não apresenta garantia operacional.

O princípio do método consiste em interromper a alimentação de ar comprimido nos elementos de sinal que estiverem gerando contrapressão na pilotagem das válvulas de comando, interferindo, assim, na sequência de movimentos dos elementos de trabalho. A contrapressão é evitada porque a sequência é dividida em setores, e cada setor pode conter apenas um movimento de cada cilindro pneumático.

Após verificar se a sequência é direta ou indireta, conforme mostrado na seção anterior, passa-se ao primeiro passo da solução de um circuito pneumático utilizando o método cascata, que consiste em dividir a sequência em setores. Por exemplo, ao resolver o diagrama $A^+ B^+ B^- A^-$ pelo método, obtém-se:

$$\begin{array}{c} A^+ B^+ \mid B^- A^- \\ \text{Setor I} \mid \text{Setor II} \end{array}$$

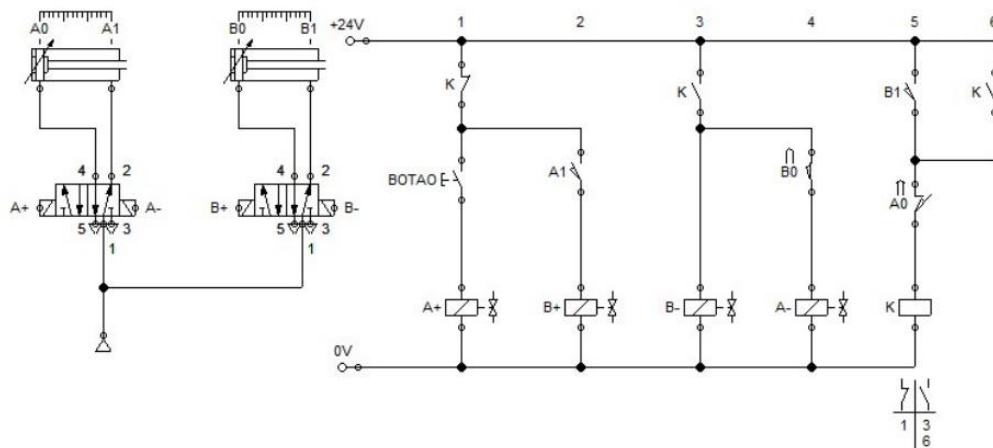
Para estabelecer a divisão de setores, é necessário seguir uma regra: sempre que uma letra se repetir, independentemente da direção, inicia-se um novo setor com um traço vertical. Com base nisso, determina-se o número de linhas pneumáticas necessárias para controlar esses setores, sendo que cada setor deve possuir uma linha pneumática. O número de relés necessários para controlar os setores é igual ao número de linhas menos um. Após a definição da cascata elétrica com base no número de relés, a próxima etapa consiste em preencher o quadro de sequência dos acionamentos para o método cascata. Continuando o exemplo, a Tabela 4 mostra o quadro de sequência dos acionamentos preenchido para a sequência $A^+ B^+ B^- A^-$.

Tabela 4 – Quadro de sequência dos acionamentos

Passo	Comando	Acionamento	Setor
1	Botão de partida	Avança A	I
2	A1 acionado	Avança B	I
3	B1 acionado	Mudança de setor I → II	Rede
4	Setor II energizado	Recua B	II
5	B0 acionado	Recua A	II
6	A0 acionado	Mudança de setor II → I e fim de ciclo	Rede

Por fim, é possível construir o diagrama eletropneumático com base no quadro de sequência. É importante lembrar que apenas válvulas de duplo solenóide devem ser utilizadas. A Figura 10 apresenta o diagrama eletropneumático para a sequência A+ B+ B– A–.

Figura 10 – Diagrama eletropneumático pelo método cascata



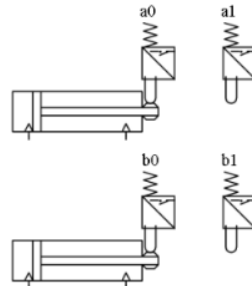
Fonte: Lista de exercícios de Automação Industrial.

2.2.3 MÉTODO TRAJETO PASSO

O método trajeto-passo consiste na elaboração dos diagramas trajeto-passo referentes ao posicionamento dos cilindros, ao diagrama de atuação dos sensores e ao diagrama de comando dos atuadores. Com base nestes diagramas, é possível definir as equações de acionamento para os elementos de trabalho. Alguns aspectos deste método são: exige a comparação dos diagramas de comando e de atuação, não se limita a circuitos de porte médio e, em alguns cenários, requer sintetizar a solução, ou seja, transformar as informações obtidas nos diagramas em um formato otimizado utilizando a ferramenta denominada Diagrama de Karnaugh-Veitch.

Para exemplificar as etapas deste método, será utilizada a sequência A+ B+ A- B-. A Figura 11 apresenta o esquema dos atuadores pneumáticos e seus respectivos sensores fins de curso identificados.

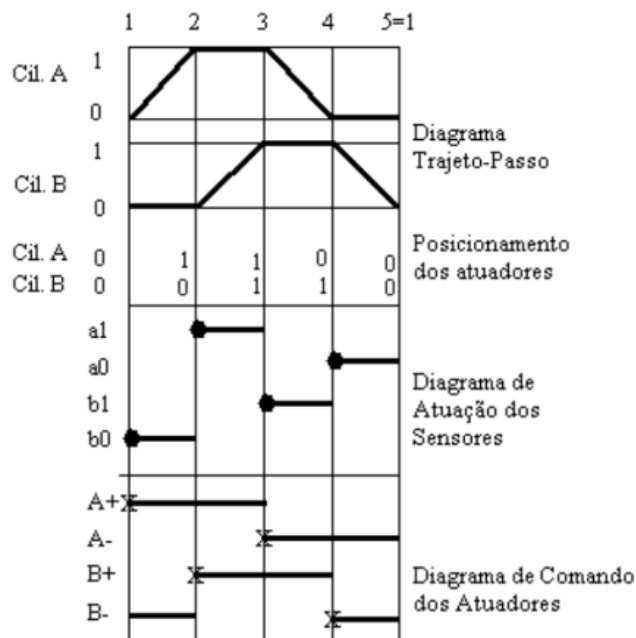
Figura 11 – Atuadores pneumáticos e sensores fins de curso



Fonte: Módulo 1 de Automação Industrial.

A primeira etapa é elaborar o diagrama trajeto-passo do posicionamento dos cilindros. Esse diagrama, além de apresentar as posições dos cilindros, mostra as linhas de mudança de estado de cada um desses elementos de trabalho, sendo que cada elemento assume apenas duas posições: posição 0 (elemento recuado) ou posição 1 (elemento avançado). A Figura 12 apresenta o diagrama trajeto-passo referente à sequência A+ B+ A- B-.

Figura 12 – Diagrama trajeto-passo

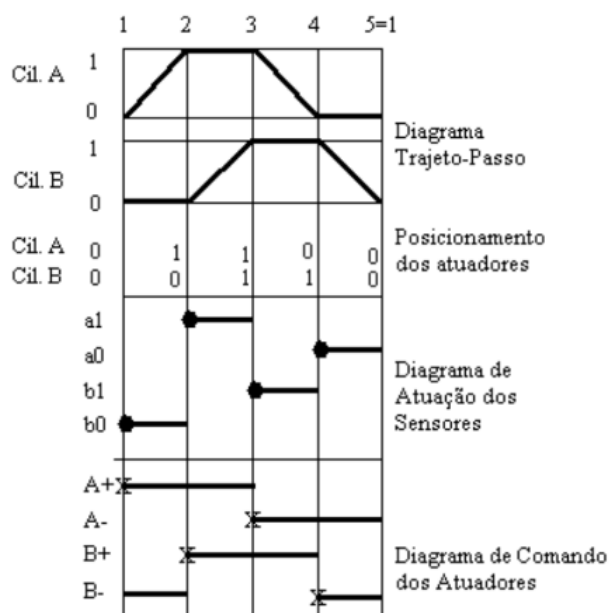


Fonte: Módulo 1 de Automação Industrial.

A próxima etapa é elaborar o diagrama de acionamento dos sensores e atuadores, sendo que, no diagrama dos sensores, o início do sinal é representado pelo símbolo • (círculo fechado)

e, no diagrama dos atuadores, o início do comando é representado pelo símbolo x (letra xis). A Figura 13 apresenta os diagramas de acionamento dos sensores e atuadores para a sequência A+ B+ A- B-.

Figura 13 – Diagrama de acionamento dos sensores e atuadores



Fonte: Módulo 1 de Automação Industrial.

Com os diagramas finalizados, é possível definir as equações para cada acionamento. Ao analisar o diagrama da Figura 13, pode-se identificar as seguintes situações:

- A+ e b0 iniciam no passo 1;
- A- e b1 iniciam no passo 3;
- B+ e a1 iniciam no passo 2;
- B- e a0 iniciam no passo 4.

Observa-se que a duração da atuação dos sensores b0 e b1 é menor do que a dos comandos A+ e A-, portanto, o comando do atuador A é bidirecional. Da mesma forma, a duração da atuação dos sensores a1 e a0 é inferior à dos comandos B+ e B-, tornando o comando do atuador B também bidirecional. Assim, em ambos os atuadores, deverão ser utilizadas válvulas de duplo solenóide.

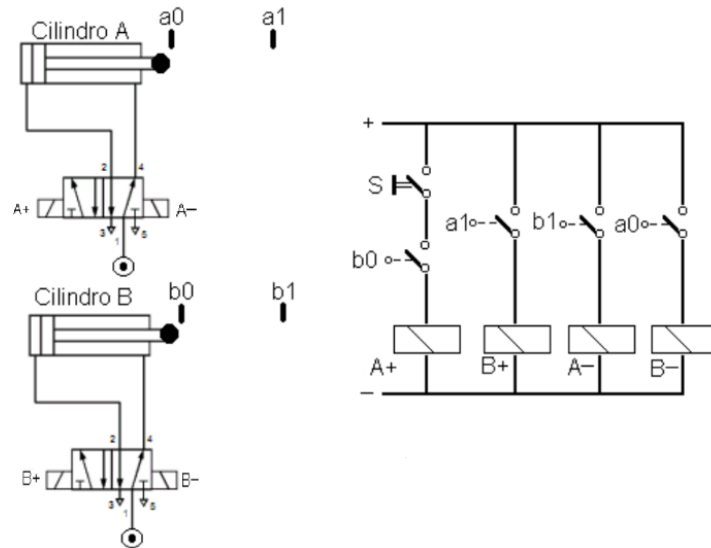
Assim, as equações para os movimentos do atuador A são: $A+ = b0$ e $A- = b1$, e para os movimentos do atuador B: $B+ = a1$ e $B- = a0$.

Para sair do repouso, o início do ciclo de trabalho deve ser acionado por algum dispositivo, por exemplo, uma botoeira ou uma chave liga/desliga. Assim, o comando A+ é

acrescido da letra S, que representa o acionamento para o início do ciclo, resultando na equação: $A+ = b0.S$.

Com as equações de cada acionamento, temos a última etapa do método trajeto-passo: a elaboração do diagrama eletropneumático. A Imagem X apresenta o diagrama eletropneumático para a sequência $A+ B+ A- B-$.

Figura 14 – Diagrama eletropneumático pelo método trajeto-passo



Fonte: Módulo 1 de Automação Industrial.

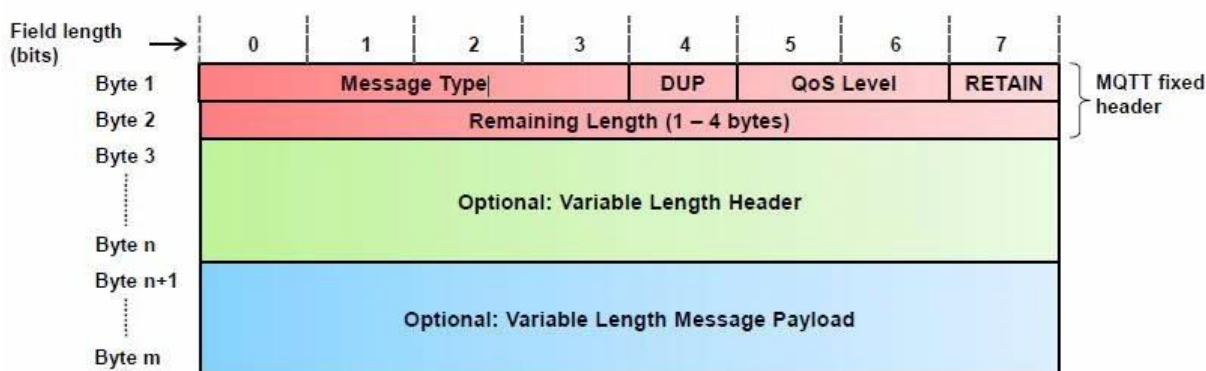
É importante lembrar que, para algumas sequências, será necessário utilizar um relé (representado pela letra K) como elemento auxiliar na resolução do problema. Isso ocorre quando há repetição no diagrama de posicionamento dos atuadores dentro de um mesmo ciclo. Dessa forma, além das equações que descrevem os movimentos dos atuadores, há também uma equação específica para o relé.

2.3 PROTOCOLO MQTT

O *Message Queuing Telemetry Transport* (MQTT), que em português significa Transporte de Filas de Mensagem de Telemetria, foi criado na década de 90 pela IBM e Eurotech. Ele é um protocolo de comunicação máquina para máquina (M2M - Machine to Machine) desenvolvido para a Internet das Coisas (IoT), sendo ideal para conectar dispositivos remotos oferecendo um baixo consumo de rede e banda por conta da sua baixa exigência de recursos de software. O MQTT tem sido utilizado em diversas áreas da indústria, como automotiva, manufatura, telecomunicações, petróleo e gás (MQTT.org, 2024).

Analisando a arquitetura de rede do protocolo MQTT, nota-se que para a transmissão de dados o mesmo foi baseado no modelo TCP/IP. Assim, para otimizar a largura de banda de rede, os cabeçalhos de mensagens MQTT são pequenos podendo variar, geralmente, de dois a cinco bytes, sendo que por padrão todos os cabeçalhos tem os seguintes campos no primeiro byte: tipo de mensagem, indicador de mensagem duplicada, identificador da qualidade de serviço (QoS) do pacote e indicador de mensagem retida ou não. Os demais bytes definirão o tamanho do pacote como um todo, sendo que o tamanho mínimo é de dois bytes e o tamanho máximo do pacote é de 256 MB. A Figura 15 apresenta como é o cabeçalho do protocolo MQTT.

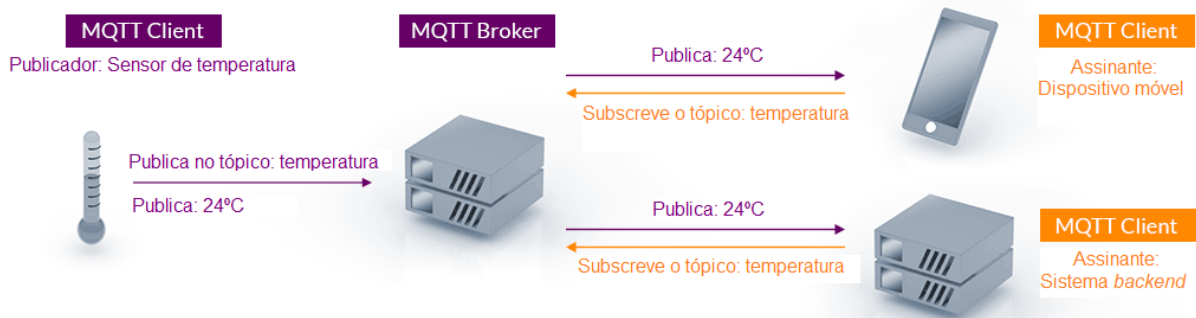
Figura 15 – Cabeçalho do protocolo MQTT



Fonte: <https://www.rfwireless-world.com/Tutorials/MQTT-tutorial.html>

O protocolo MQTT funciona com um modelo de publicação e subscrição, permitindo que os clientes publiquem e/ou recebam informações enquanto o servidor gerencia esse envio e recebimento das respectivas informações. Em uma comunicação MQTT, um publicador é responsável por enviar mensagens para um tópico específico, sendo que um ou mais assinantes podem se inscrever para receber essas mensagens. Além disso, o MQTT suporta comunicações bidirecionais, possibilitando que um dispositivo atue simultaneamente como publicador e assinante, facilitando a troca contínua de informações. A Figura 16 ilustra a arquitetura do protocolo MQTT, destacando seu funcionamento com ênfase no modelo de publicação e subscrição.

Figura 16 – Modelo de publicação e subscrição no MQTT



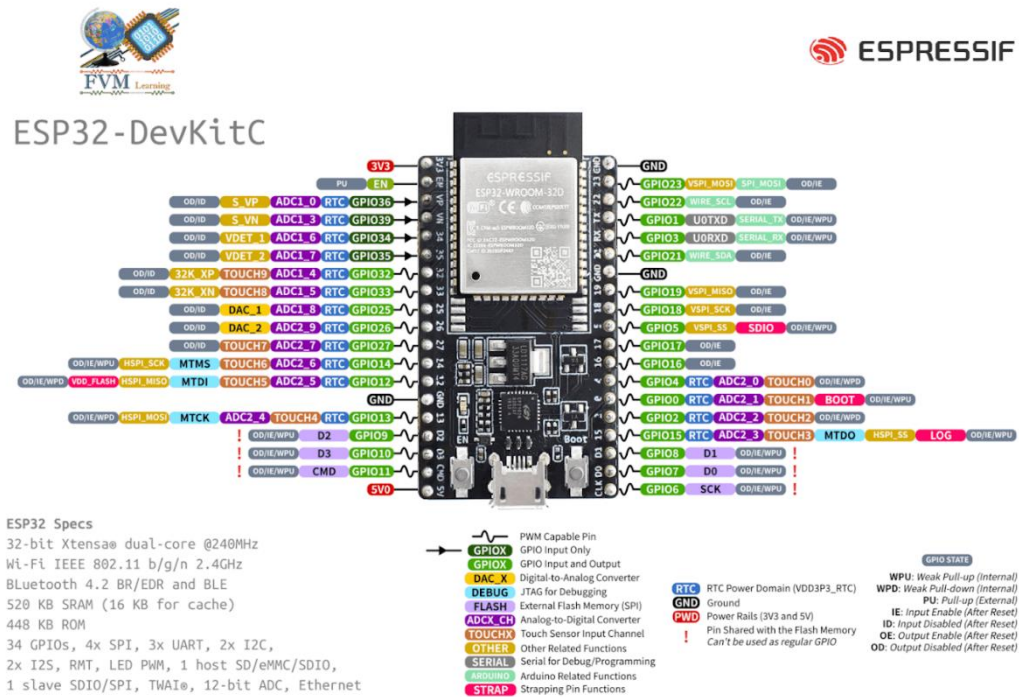
Fonte: <https://mqtt.org/>

2.4 MICROCONTROLADOR ESP32

O microcontrolador NodeMCU-32S ESP32, ou simplesmente ESP32, é uma placa de desenvolvimento fabricada pela empresa Espressif que utiliza o módulo ESP32S Wi-Fi, um componente eletrônico avançado projetado especificamente para facilitar a conexão de projetos robóticos e de automação à Internet, oferecendo uma solução acessível e eficiente (Usinainfo, 2023).

O NodeMCU possui o módulo ESP-WROOM-32 que além de conter o ESP32, conta com um cristal de 40 MHz, antena embutida e porta micro USB para alimentação e programação, além dos pinos GPIOs, Conversor Analógico Digital com 12 bits de resolução (até 18 canais), 2 DAC com 8 bits de resolução. Outro diferencial exclusivo do NodeMCU-32S ESP32 é que ele além de possuir conexão Wi-Fi nativa, conta também com Bluetooth v4.2 embutido e micro processador dual core Xtensa® 32-bit LX6. (Usinainfo, 2023). A Figura 17 mostra um modelo de ESP32 e sua pinagem.

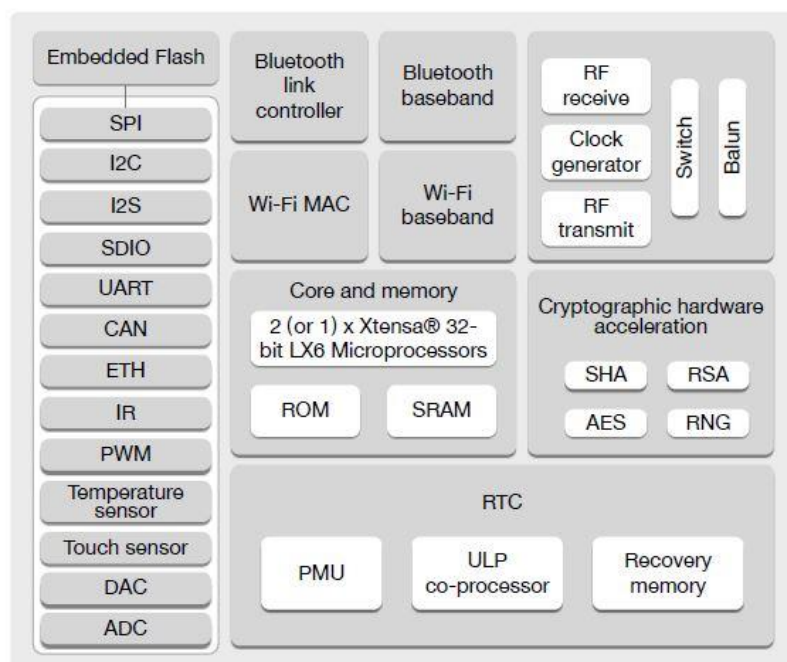
Figura 17 – Pinagem ESP32



Fonte: <https://www.fvml.com.br/2022/04/pinagem-pinout-esp32-devkitc.html>

O ESP32 possui um design robusto, sendo capaz de funcionar de forma confiável em ambientes industriais, com uma temperatura operacional variando de -40°C a $+125^{\circ}\text{C}$. Possui consumo de energia ultra baixo, projetado para dispositivos móveis e aplicações de IoT, incluindo recursos de última geração, como clock gating de granulação fina, vários modos de energia e dimensionamento dinâmico de energia. O ESP32 é altamente integrado com sistemas de antena embutidos, amplificador de potência, amplificador de recepção de baixo ruído, filtros e módulos de gerenciamento de energia. O ESP32 possui chip híbrido Wi-Fi e Bluetooth podendo atuar como um sistema autônomo completo ou como um dispositivo escravo, reduzindo a sobrecarga da pilha de comunicação no processador do aplicativo principal (Espressif, 2024). A Figura 18 apresenta o diagrama em blocos do chip ESP32.

Figura 18 – Diagrama em blocos do chip ESP32



Fonte: <https://arduino.com.br/o-que-e-o-esp32/>

Para a programação dos códigos do ESP32 utilizou-se a interface do Arduino, pois a interface do Arduino tem uma grande comunidade de usuários. Na seção de desenvolvimento será mostrada as configurações e os códigos de cada microcontrolador utilizado no projeto.

2.5 MICROCOMPUTADOR RASPBERRY

O Raspberry é um microcomputador de placa única desenvolvido pela empresa Raspberry Pi Foundation. Este dispositivo tem uma boa capacidade de velocidade e desempenho, baixo consumo de energia e tamanho compacto, podendo substituir um computador padrão em diversas aplicações. Para esse projeto, utilizou-se o Raspberry Pi 4 Modelo B para funcionar como um servidor broker MQTT local que foi configurado e disposto no Laboratório de Ensino para Mecatrônica da universidade. As suas características estão apresentadas a seguir.

Hardware

- Processador ARM-Cortex A72 quad-core de 64 bits rodando a 1,5 GHz;
- Memória RAM de 4 Gigabytes;
- Decodificador de vídeo H.264 e H.265 (HEVC);
- Gráficos 3D VideoCore VI.

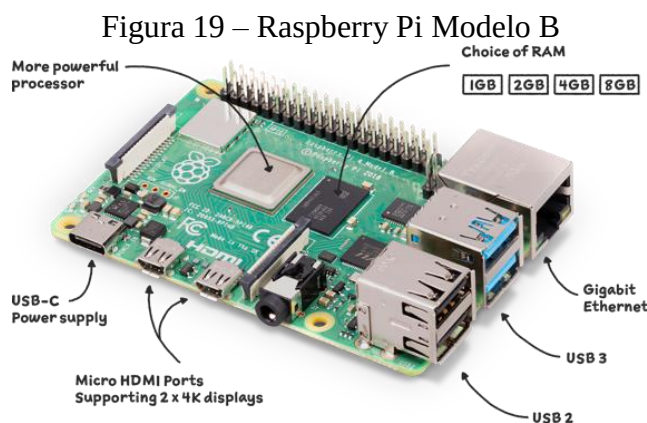
Interfaces

- Rede Wi-Fi no padrão 802.11 b/g/n/ac;
- Bluetooth 5.0 com BLE;
- 1x entrada para cartão de memória SD;
- 2x portas micro-HDMI com suporte para monitores duplos;
- 2x portas USB 2.0;
- 2x portas USB 3.0;
- 1x porta Gigabit Ethernet;
- 1x porta Raspberry Pi camera;
- 1x porta Raspberry Pi display;
- 28x portas GPIO.

Software

- Conjunto de instruções ARMv8;
- Sistema operacional baseado no Linux;
- Atualizações de software constantes.

As configurações realizadas para que o microcomputador atuasse como um broker MQTT e funcionasse diretamente na rede da universidade serão apresentadas na seção de desenvolvimento. A Figura 19 mostra um Raspberry Pi Modelo B.



Fonte: <https://www.raspberrypi.com/products/raspberry-pi-4-model-b/>

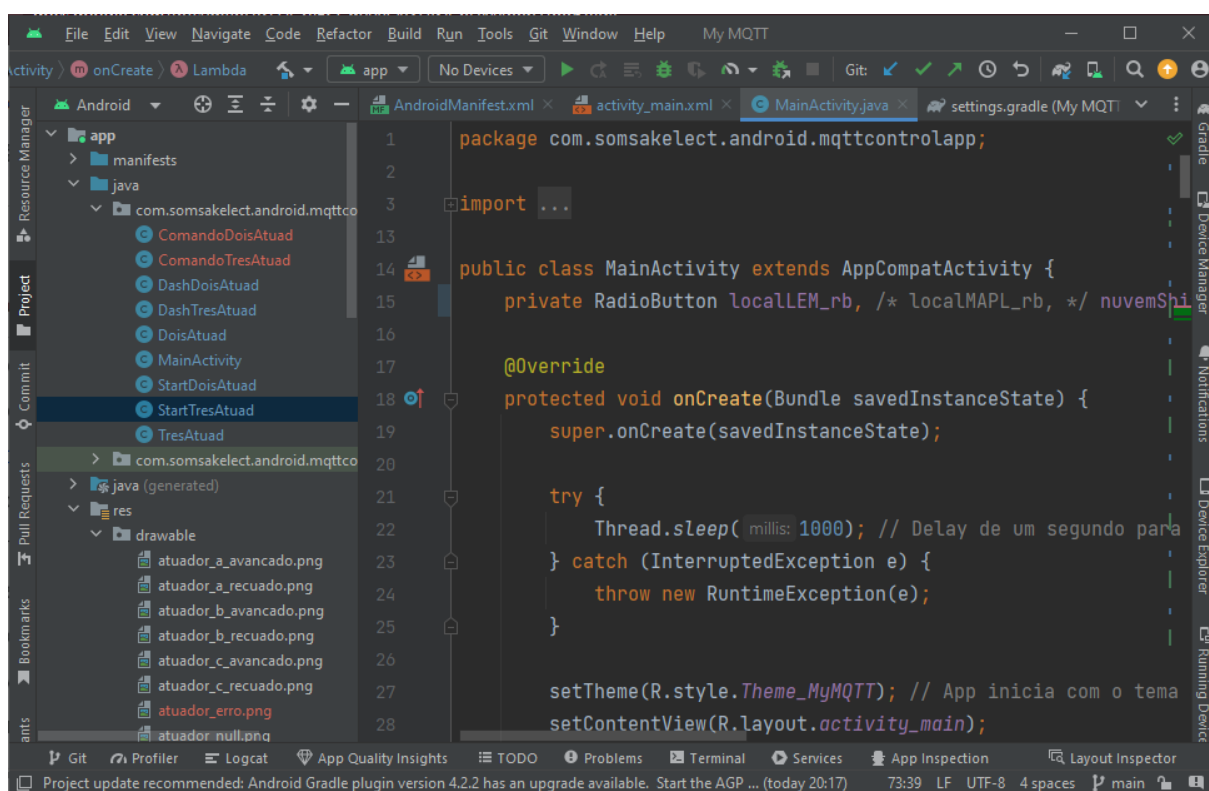
2.6 ANDROID STUDIO

O Android Studio é o ambiente de desenvolvimento integrado oficial para o desenvolvimento de apps Android criado e distribuído pelo Google. Um ambiente de desenvolvimento integrado contém ferramentas que permitem aos desenvolvedores de software projetar, criar, executar e testar softwares. Neste caso, apps para a plataforma Android. O

Android Studio usa o IntelliJ IDEA como base e inclui o plug-in do Android pré-instalado junto de algumas mudanças específicas da plataforma Android (Google Developers, 2024).

Escolheu-se o Android Studio para o desenvolvimento do aplicativo, pois através de pesquisas e testes foi a ferramenta que melhor atendeu às requisições do projeto quanto à conexão e comunicação utilizando o protocolo MQTT. O projeto no Android Studio é formado por linhas de código, sendo que para cada tela do aplicativo temos um código em XML que se refere à parte visual da tela e outro código em Java que se refere às funcionalidades do aplicativo. As configurações e códigos serão apresentados na seção de desenvolvimento. A Figura 20 apresenta a interface do Android Studio.

Figura 20 – Interface do Android Studio



3 DESENVOLVIMENTO

Na presente seção, serão apresentadas a arquitetura do projeto, os materiais e os programas utilizados, as configurações requeridas para as fases de desenvolvimento ao longo do processo do projeto. Adicionalmente, serão mostrados os procedimentos relativos ao desenvolvimento dos códigos dos microcontroladores, bem como do código do aplicativo do projeto.

3.1 ARQUITETURA DO PROJETO

A arquitetura do projeto segue o modelo cliente-servidor, baseado no protocolo de comunicação MQTT para a interconexão entre os dispositivos, sendo que o cliente pode ser do tipo *Publisher*, que publica mensagens no servidor, ou do tipo *Subscriber*, que recebe as publicações feitas no servidor. A aplicação requer o controle das válvulas eletropneumáticas, portanto, há um aplicativo para celular que funciona como controlador das operações. Por meio dele, é possível utilizar três modos: modo controle, no qual o usuário preenche as equações sequenciais obtidas pelo método trajeto-passo, fazendo com que a bancada realize a sequência de movimentos; modo comando, em que o usuário envia comandos para avançar e recuar os atuadores eletropneumáticos por meio de botões dispostos na interface gráfica; e, por fim, modo dashboard, onde o usuário acompanha, em tempo real, o estado (avançado ou recuado) de cada atuador eletropneumático. Para melhor entendimento, segue uma visão geral dos componentes ligados à rede MQTT:

- **Celular (Cliente Publisher/Subscriber):**

O aplicativo móvel atua como cliente MQTT. Ele envia comandos (publica mensagens) em tópicos específicos para controlar as operações da bancada eletropneumática e, se necessário, também recebe atualizações (por meio de assinaturas) sobre o estado dos atuadores.

- **Broker MQTT (Servidor):**

O broker centraliza a troca de mensagens entre todos os clientes. Ele recebe as publicações tanto do celular quanto dos microcontroladores ESP32 e encaminha cada mensagem para os dispositivos que estão inscritos nos respectivos tópicos. Dessa forma, o celular pode enviar comandos e, ao mesmo tempo, receber informações dos ESP32. Esse intermediário garante a escalabilidade e a segurança na comunicação.

- **Microcontroladores ESP32 das válvulas (Clientes Subscriber):**

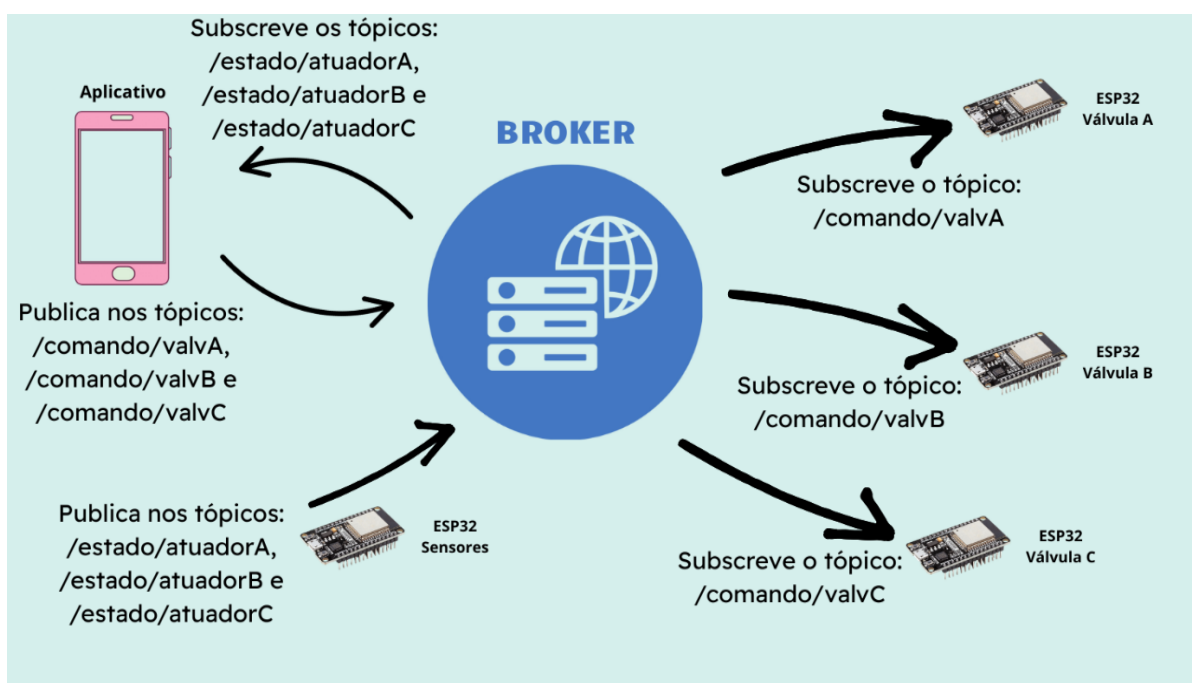
Os ESP32, instalados nas válvulas da bancada eletropneumática, são responsáveis por executar os comandos recebidos (por exemplo, acionar ou desacionar a válvula), resultando no avanço ou recuo dos atuadores eletropneumáticos.

- **Microcontroladores ESP32 dos sensores de fim de curso (Clientes Publisher):**

Os ESP32, instalados nos sensores de fim de curso da bancada eletropneumática, são responsáveis por publicar as posições dos atuadores eletropneumáticos (avançado ou recuado).

A Figura 21 apresenta o desenho esquemático da arquitetura do projeto.

Figura 21 – Desenho esquemático da arquitetura do projeto



Com base na Figura 21, é possível compreender melhor o fluxo de comunicação entre os diversos dispositivos que compõem o sistema. Resumidamente, o aplicativo para celular envia um comando publicando uma mensagem MQTT em um tópico específico. Em seguida, o broker MQTT recebe essa mensagem e a encaminha para todos os dispositivos inscritos neste tópico. Nesse caso, os microcontroladores inscritos no tópico recebem a mensagem e executam as ações correspondentes na bancada eletropneumática.

Essa arquitetura permite uma comunicação assíncrona e flexível, onde os dispositivos não precisam se conhecer diretamente, facilitando a manutenção, escalabilidade e integração com outros sistemas.

3.2 MATERIAIS E PROGRAMAS UTILIZADOS

A seguir, temos uma listagem detalhada dos materiais e programas utilizados no desenvolvimento do projeto.

- 1 microcomputador Raspberry Pi 4;
- 1 cabo HDMI;
- 1 adaptador micro USB - HDMI;
- 1 monitor;
- 1 teclado;
- 1 mouse;
- 1 fonte de alimentação 5V e cabo com saída USB-C;
- 1 cartão de memória micro SD de 32 GB;
- 1 adaptador para cartão de memória;
- 4 microcontroladores ESP32;
- 3 módulos relés;
- 3 atuadores de dupla ação;
- 3 válvulas eletropneumática 5/2 simples solenoide de retorno por mola;
- 6 sensores de fim de curso;
- 1 bancada eletropneumática com fonte de ar comprimido e fonte de 24 VDC;
- 1 computador;
- Brokers em nuvem (Shiftr.io e HiveMQ);
- Software SD Card Formatter;
- Software Win32 Disk Imager ou Balena Etcher;
- Software IDE do Arduino;
- Software Android Studio;
- Software de simulação FluidSIM;
- Software MQTTBox.

3.3 CONFIGURAÇÕES REQUERIDAS

Para o funcionamento do Raspberry como broker MQTT local, foi necessário solicitar ao Centro de Tecnologia da Informação (CTI) da UFU a definição de um endereço IP fixo para o equipamento, sendo obrigatório informar o endereço MAC (MAC Address: eth0 –

E4:5F:01:8F:A8:9F; wlan0 – E4:5F:01:8F:A8:A0). Algumas instruções foram fornecidas pelo CTI, conforme listado a seguir:

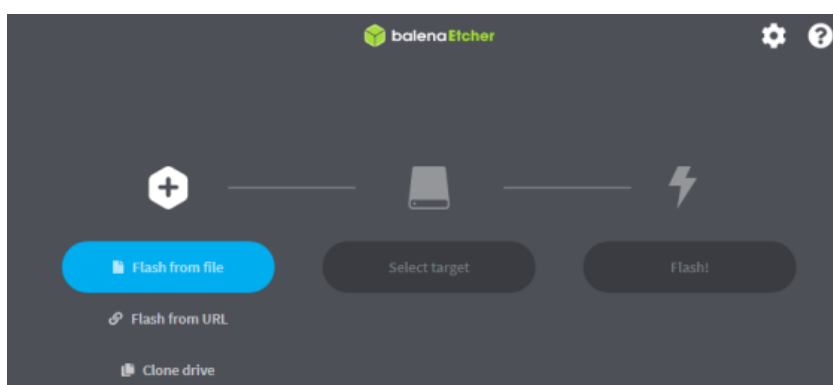
- Configurar para aquisição de IP via DHCP;
- Liberação das portas: TCP/UDP – nº 1883 e 8883; HTTP/HTTPS – nº 80 e 443; FTP – nº 20 e 21;
- IP da rede cabeada: 10.32.80.12;
- IP da rede Wi-Fi: 10.47.0.2;
- IP público: 200.131.193.196.

3.3.1 INSTALAÇÃO DO SISTEMA OPERACIONAL RASPBERRY PI

Para a instalação do SO do Raspberry, o primeiro passo é formatar o cartão de memória para garantir que não há arquivos instalados e/ou partições no cartão. Para esse procedimento utilizou-se o software *SD Card Formatter*. Com o cartão de memória formatado, seguimos para a instalação do sistema operacional. Para esse procedimento é possível utilizar dois softwares: *Win32 Disk Imager* ou *Balena Etcher*. Mas antes de abordarmos sobre cada um deles, é necessário o arquivo que contém a imagem do sistema operacional. No site oficial do Raspberry Pi deve-se fazer o download da opção compatível com o Raspberry que será utilizado no projeto. Portanto, é necessário informar o modelo Raspberry Pi 4 Model B Rev. 1.5 e o tipo de sistema aarch64 (64 bits).

Agora, vamos abordar como é feita a instalação utilizando-se o software Balena Etcher. Com o programa instalado e inicializado, visualizou-se o que está apresentado na Figura 22.

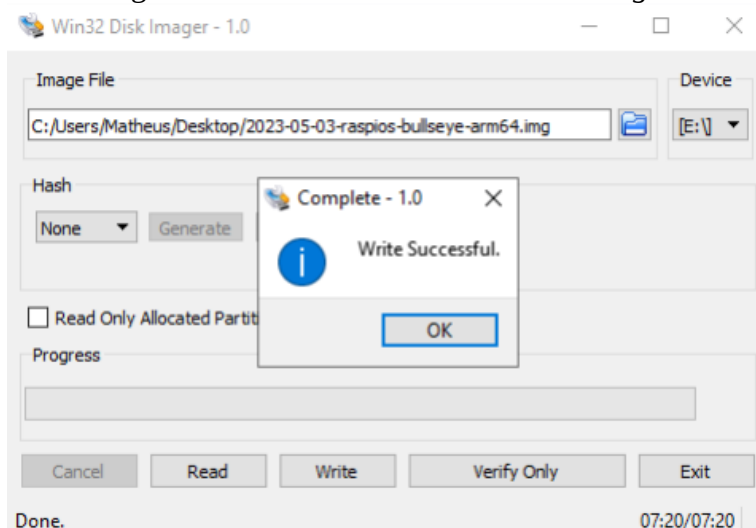
Figura 22 – Interface do Balena Etcher



Assim, clicou-se no botão “Flash from file” para selecionar o arquivo com a imagem do sistema operacional e posteriormente no botão “Select target” para selecionar o cartão de memória. Feito isso, a instalação é iniciada clicando-se no botão “Flash!”. Após o processo ser finalizado, o cartão está pronto para ser inserido no slot do Raspberry.

Outra maneira, é instalar o sistema operacional utilizando o software Win32 Disk Imager. Com o programa instalado e inicializado, escolhe-se o caminho do arquivo com a imagem do sistema operacional no campo “Image File”. Além disso, no campo “Device” deve selecionar o cartão de memória. Depois, clica-se no botão “Write” e aguarde até que o processo de escrita seja finalizado, conforme está apresentado na Figura 23.

Figura 23 – Interface do Win32 Disk Imager



Com o cartão de memória inserido no Raspberry, mouse e teclado conectados via porta USB 2.0, cabo de rede conectado via porta Ethernet, monitor conectado via cabo HDMI com adaptador micro USB e fonte de alimentação conectada via porta USB-C, iniciou-se a configuração do sistema ligando a fonte de alimentação na tomada. Após isso, o sistema iniciou o processo de boot e reboot até chegar na tela mostrada na Figura 24. Ademais, é possível notar que o IP mostrado é o IP da rede cabeada que foi passado pelo CTI.

Figura 24 – Interface de instalação do sistema operacional do Raspberry Pi

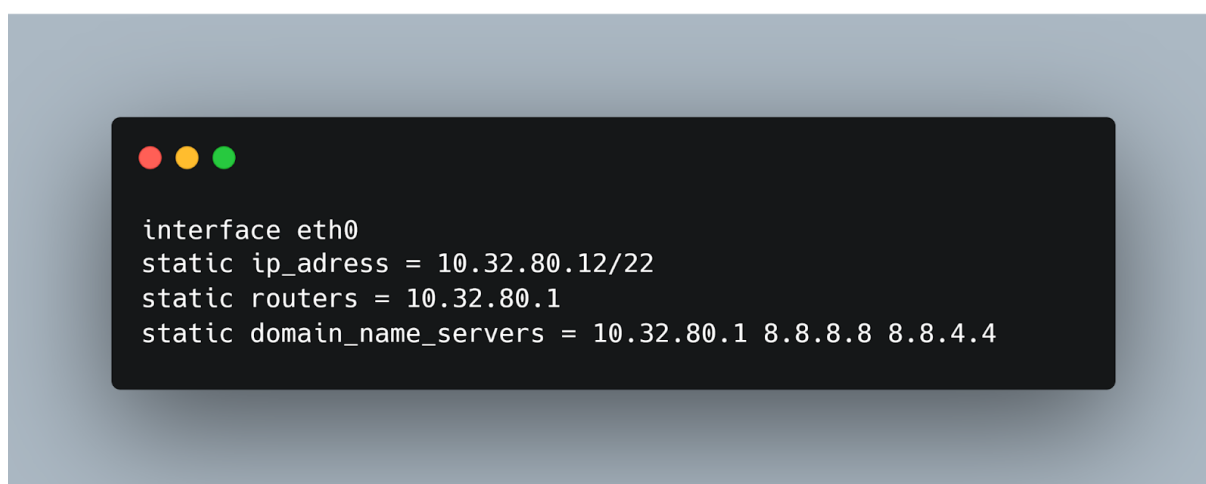


Clicando em “Next” o sistema solicita informações de país, língua, fuso horário, usuário, senha e dimensionamento de tela do desktop. Posteriormente, o sistema pediu para escolher uma rede de internet Wi-Fi, mas como nossa configuração foi realizada com a internet cabeada, continuou-se para os próximos passos clicando em “Skip” até chegar na tela que solicita a reinicialização do sistema. Com o sistema reinicializado foi concluída a instalação do sistema operacional. No próximo tópico será abordado como configurar o IP fixo do Raspberry.

3.3.2 DEFININDO IP FIXO PARA O RASPBERRY PI

Agora, temos que fazer alguns ajustes para o Raspberry adquirir o IP fixo e conseguir conexão com a rede UFU-Portal. Inicialmente, com o terminal do Raspberry aberto digita-se o comando **netstat -nr** para buscar o IP do roteador, pois será necessário informá-lo no arquivo de configuração. Nesse caso, o IP do roteador conectado foi 10.32.80.1. Sabendo os IPs do Raspberry e do roteador, prosseguiu-se para a edição do arquivo de configuração responsável por fixar o IP com o comando **sudo nano /etc/dhcpd.conf**. Com o arquivo aberto deve-se inserir quatro linhas com informações referentes ao tipo de conexão e IP, conforme mostrado na Figura 25. É válido lembrar que a conexão por Wi-Fi está desativada. Assim, todo o processo está sendo realizado por conexão Ethernet e o IP (10.32.80.12) é o mesmo que foi informado pelo CTI referente à conexão cabeada (eth0).

Figura 25 – Arquivo de configuração dhcpd

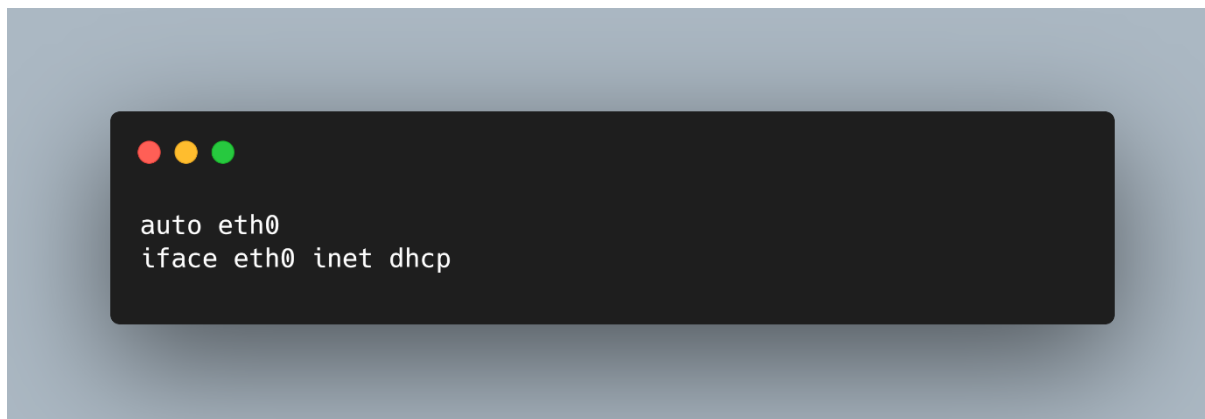


Para salvar as alterações feitas no arquivo, utiliza-se o atalho “Ctrl + O”. Em seguida, deve-se confirmar o nome do arquivo com a tecla “Enter” e em seguida sair com o atalho “Ctrl + X”.

Agora, vamos editar o arquivo responsável pela configuração da interface de rede para a conexão com a internet ser bem-sucedida. O comando para editar o arquivo é **sudo nano**

/etc/network/interfaces. Com o arquivo aberto deve-se inserir duas linhas indicando o tipo de interface a ser utilizada, conforme apresentado na Figura 26.

Figura 26 – Arquivo de configuração interfaces



Após isso, deve-se salvar o arquivo e reiniciar o Raspberry com o comando **sudo reboot**.

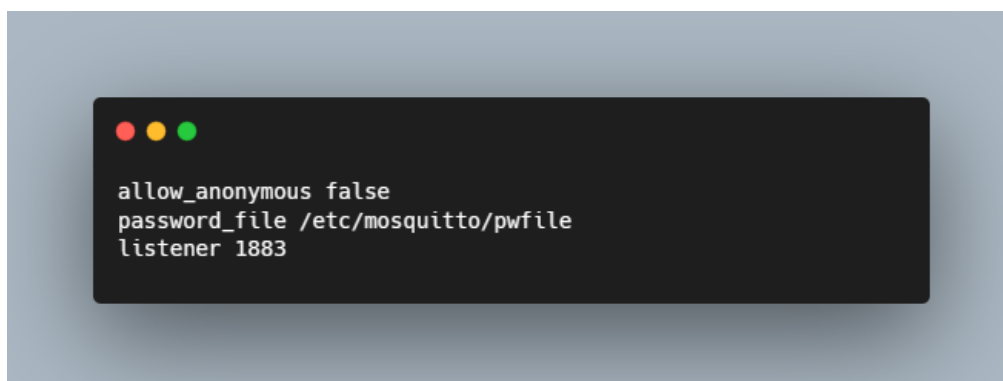
Nos apêndices A e B estão, respectivamente, os códigos de configuração dos arquivos *dhcpcd* e *interfaces*.

3.3.3 IMPLEMENTAÇÃO DO MQTT NO RASPBERRY PI

Para a implementação do MQTT no Raspberry Pi utiliza-se o Eclipse Mosquitto que é um broker de software livre (licença EPL/EDL). Para instalar o Mosquitto no sistema operacional do Raspberry, deve-se executar no terminal de comando **sudo apt-get install mosquitto -y**. O parâmetro “-y” é utilizado para confirmar e instalar diretamente todos os diretórios sem a necessidade de solicitar permissão. Além disso, instalou-se também uma ferramenta auxiliar do Mosquitto responsável por publicar e subscrever tópicos com os comandos *mosquitto_pub* e *mosquitto_sub*. Para instalar essa ferramenta, no terminal do Raspberry deve-se executar o comando **sudo apt-get install mosquitto-clients -y**.

O próximo passo é fazer algumas alterações no arquivo de configuração do Mosquitto. Para acessar o arquivo deve-se executar o comando **sudo nano /etc/mosquitto/mosquitto.conf**. Com o arquivo aberto, adiciona-se ao final do arquivo três linhas, conforme mostrado na Figura 27.

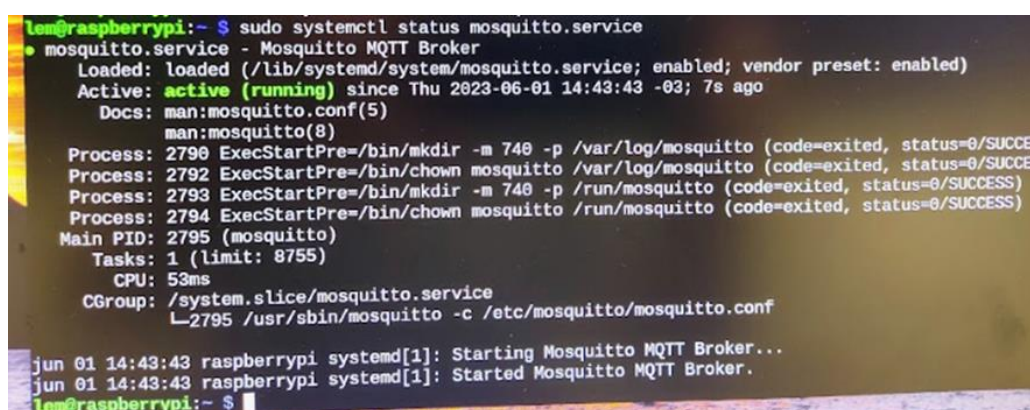
Figura 27 – Arquivo de configuração Mosquitto



A primeira linha é para bloquear conexões sem identificação, na segunda linha temos a definição do arquivo responsável por armazenar usuários e senhas e a terceira linha é a especificação da porta de conexão do broker. Após adicionar essas informações, salva-se o arquivo. A próxima etapa é definir qual será o usuário e senha do broker. Para isso, deve-se executar o seguinte comando **sudo mosquitto_passwd -c /etc/mosquitto/pwfile localufulem**. Sendo que o último parâmetro será o nome de usuário do broker. Nesse caso, “localufulem” é o nome de usuário. Ao pressionar a tecla “Enter” será solicitado a senha e posteriormente a confirmação da mesma. Para visualizar o arquivo contendo usuário e senha, deve-se executar o seguinte comando **nano /etc/mosquitto/pwfile**. Lembrando que a senha é mostrada de modo criptografado.

Agora, para aplicar as alterações deve-se reiniciar o broker com o comando **sudo systemctl restart mosquitto.service**. Após isso, executa-se o comando **sudo systemctl status mosquitto.service** para conferir se o broker está em execução. A Figura 28 apresenta o resultado desse comando. Note a mensagem em verde *active (running)* informando que o servidor está ativo. O apêndice C contém o código de configuração do Mosquitto.

Figura 28 – Estado de funcionamento do Mosquitto



É importante ressaltar que o Raspberry Pi, embora seja uma plataforma de prototipagem de baixo custo e alta flexibilidade, não foi projetado para ambientes industriais onde segurança e robustez são requisitos críticos. Primeiro, a execução de uma distribuição Linux completa demanda atualização e correção contínuas de software, expondo-o a vulnerabilidades conhecidas caso esse ciclo não seja rigorosamente mantido. Além disso, o sistema padrão do Raspbian habilita serviços de rede como SSH (porta 22) e VNC (porta 5900) por padrão, abrindo diversas portas que ampliam a superfície de ataque se não forem devidamente protegidas. Por fim, o hardware do Raspberry Pi não oferece isolamento galvanico, proteção contra interferências eletromagnéticas (EMI) ou faixas de tensão industriais presentes em controladores lógicos programáveis (CLPs) e PCs industriais, tornando-o suscetível a falhas em condições extremas de temperatura, vibração ou surtos de energia. Dessa forma, recomenda-se limitar o uso do Raspberry Pi a provas de conceito e protótipos, migrando para soluções industrializadas em aplicações de produção.

3.4 CÓDIGOS DOS MICROCONTROLADORES ESP32

Para o desenvolvimento dos códigos dos microcontroladores utilizou-se a IDE do Arduino por ser uma ferramenta didática e de fácil instalação. Conforme mostrado na arquitetura do projeto, utilizou-se quatro microcontroladores, sendo utilizado três para as válvulas e um para os sensores de fim de curso. Assim, os códigos dos microcontroladores das válvulas são idênticos, alterando-se somente o ID e os tópicos a serem publicados e subscritos.

Inicialmente, deve-se importar as bibliotecas que serão utilizadas. Nesse caso, foram três: WiFi.h, responsável pela conexão Wi-Fi, esp_wpa2.h, responsável pela conexão em redes Enterprise (rede da universidade) e PubSubClient.h, responsável pela conexão MQTT. A Figura 29 apresenta como foi feito esse processo.

Figura 29 – Definição de bibliotecas



```

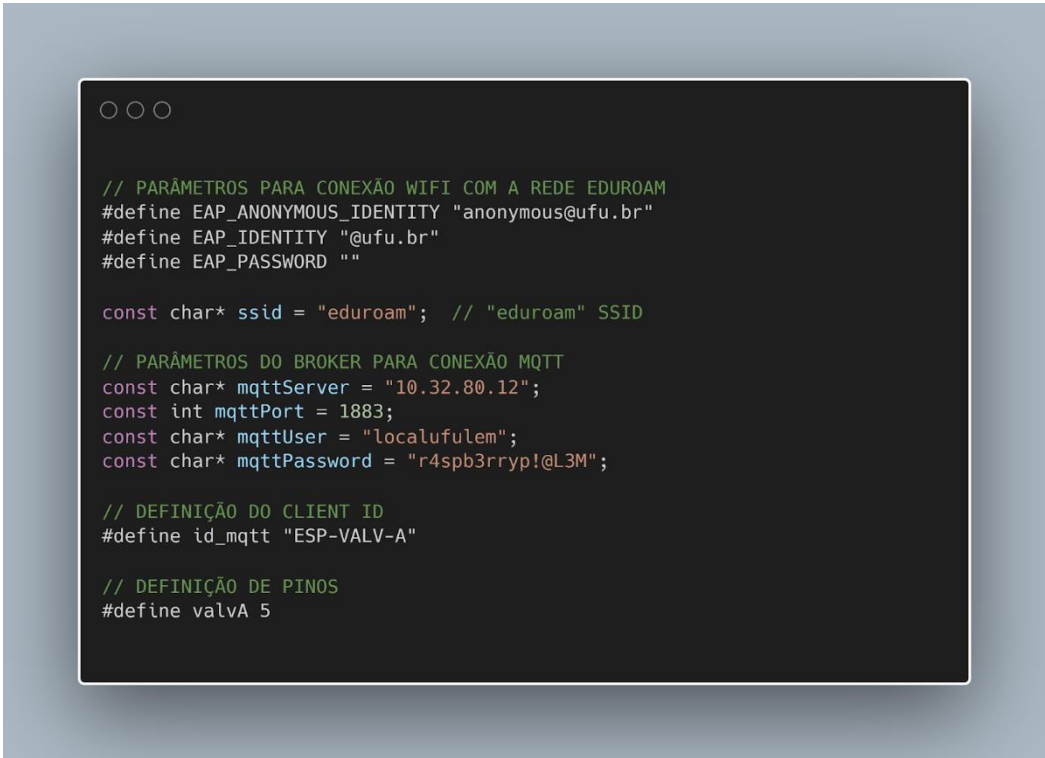
○ ○ ○

#include <WiFi.h>
#include "esp_wpa2.h"
#include <PubSubClient.h>

```

Depois, os parâmetros para conexão Wi-Fi e conexão com o broker MQTT devem ser declarados. Além disso, a definição do ID e da porta GPIO responsável pelo acionamento da válvula são realizados. Essas informações estão apresentadas na Figura 30.

Figura 30 – Definição dos parâmetros de conexão Wi-Fi e MQTT



```

○ ○ ○

// PARÂMETROS PARA CONEXÃO WIFI COM A REDE EDUROAM
#define EAP_ANONYMOUS_IDENTITY "anonymous@ufu.br"
#define EAP_IDENTITY "@ufu.br"
#define EAP_PASSWORD ""

const char* ssid = "eduroam"; // "eduroam" SSID

// PARÂMETROS DO BROKER PARA CONEXÃO MQTT
const char* mqttServer = "10.32.80.12";
const int mqttPort = 1883;
const char* mqttUser = "localufulem";
const char* mqttPassword = "r4spb3rryp!@L3M";

// DEFINIÇÃO DO CLIENT ID
#define id_mqtt "ESP-VALV-A"

// DEFINIÇÃO DE PINOS
#define valvA 5

```

Por questões de segurança, os campos de identidade e senha referentes à conexão Wi-Fi com a rede da universidade foram ocultados. Os parâmetros referentes à conexão com o broker MQTT são relacionados ao broker local instalado no Raspberry Pi do LEM, porém para utilizar os brokers em nuvem tais parâmetros devem ser alterados. O ID para o microcontrolador é dado por ESP-VALV-A e a porta GPIO 5 será utilizada para o controle da válvula.

Para o devido funcionamento do código, algumas funções foram criadas, conforme mostrado na Figura 31.

Figura 31 – Protótipos das funções



As funções `conectaWiFi()` e `conectaMQTT()` são responsáveis por iniciarem as conexões Wi-Fi e MQTT, respectivamente. Ademais, a função `mantemConexoes()` garante que tais conexões fiquem ativas. A função `callback()` é a função de retorno de chamada MQTT, ou seja, toda vez que uma mensagem é recebida via MQTT, passa pelo tratamento desta função. As funções `openValve()` e `closeValve()` são responsáveis por abrir e fechar a válvula através de um relé. Para um melhor entendimento do código, a Tabela 5 apresenta os tópicos de cada microcontrolador referente às válvulas e as suas respectivas atribuições.

Tabela 5 – Tópicos dos microcontroladores das válvulas

Microcontrolador	Tópico	Atribuição
ESP-VALV-A	/status/espValvA	Publica o status de conexão a cada 3 segundos
	/comando/valvA	Subscreve para receber os comandos de abrir e fechar a válvula A
ESP-VALV-B	/status/espValvB	Publica o status de conexão a cada 3 segundos
	/comando/valvB	Subscreve para receber os comandos de abrir e fechar a válvula B
ESP-VALV-C	/status/espValvC	Publica o status de conexão a cada 3 segundos
	/comando/valvC	Subscreve para receber os comandos de abrir e fechar a válvula C

O código do microcontrolador referente aos sensores de fim de curso é similar em relação à importação das bibliotecas e declaração dos parâmetros para conexão Wi-Fi e MQTT. A primeira mudança é a definição do ID e dos pinos, como são seis sensores de fim de curso, utiliza-se seis pinos do ESP32, um para cada fim de curso, conforme mostrado na Figura 32.

Figura 32 – Definição dos pinos para os sensores fim de curso

○ ○ ○

```
// DEFINIÇÃO DE PINOS
#define A0 2 // definindo a ligação do sensor fim de curso na porta 2
#define A1 4 // definindo a ligação do sensor fim de curso na porta 4

#define B0 16 // definindo a ligação do sensor fim de curso na porta 5
#define B1 17 // definindo a ligação do sensor fim de curso na porta 19

#define C0 18 // definindo a ligação do sensor fim de curso na porta 12
#define C1 19 // definindo a ligação do sensor fim de curso na porta 14
```

Além disso, algumas funções são diferentes, conforme mostrado na Figura 33.

Figura 33 – Protótipo das funções

○ ○ ○

```
void conectaWiFi();
void conectaMQTT();
void mantemConexoes();
void publishSensorA0A1State(int a0Value, int a1Value);
void publishSensorB0B1State(int b0Value, int b1Value);
void publishSensorC0C1State(int c0Value, int c1Value);
```

A função `publishSensorA0A1State()` realiza publicações nos tópicos referente aos sensores do atuador A, a função `publishSensorB0B1State()` realiza publicações nos tópicos referente aos sensores do atuador B e a função `publishSensorC0C1State()` realiza publicações

nos tópicos referente aos sensores do atuador C. Para um melhor entendimento do código, a Tabela 6 apresenta os tópicos do microcontrolador referente aos sensores de fim de curso e as suas respectivas atribuições.

Tabela 6 – Tópicos do microcontrolador dos sensores de fim de curso

Microcontrolador	Tópico	Atribuição
ESP-SENSORES	/status/espSensores	Publica o status de conexão a cada 3 segundos
	/estado/atuadorA	Publica o estado do atuador A
	/estado/atuadorB	Publica o estado do atuador B
	/estado/atuadorC	Publica o estado do atuador C

Nos apêndices D, E e F estão os códigos para os microcontroladores responsáveis pelo controle das válvulas e no apêndice G é possível conferir o código completo para o microcontrolador responsável por monitorar os estados dos atuadores através dos sinais dos sensores de fim de curso. Além disso, no repositório do GitHub¹ do *Manufacturing Automation Planning Lab* (MAPL) da UFU estão disponíveis todos os códigos com o passo a passo para conexão com a rede Wi-Fi da universidade e com os diversos brokers MQTT.

3.5 CÓDIGO DO APLICATIVO

A programação do aplicativo foi realizada utilizando o software Android Studio. Inicialmente, fez-se a parte visual do aplicativo, ou seja, a interface com os botões e textos que o usuário consegue interagir. Para isso, é necessário criar um código XML que contém os comandos de títulos, botões, cores das letras, cores de fundo, fontes, tamanhos, etc. A Figura 34 apresenta uma parte do código da tela inicial.

¹ <https://github.com/MAPL-UFU/mqtt-app-control>

Figura 34 – Trecho de código XML para a tela inicial



O trecho do código apresentado acima se refere às opções de brokers que o usuário poderá selecionar na tela do aplicativo. Para melhor entendimento, a interface visual da tela inicial é mostrada na Figura 35. Observe que na primeira parte temos três opções de brokers MQTT, conforme apresentado no trecho do código em XML.

Figura 35 – Tela inicial do aplicativo

MOTT SETUP

BROKER

☐ LOCAL LEM

☐ CLOUD SHIFTR.IO

☐ CLOUD HIVEMQ

ATUADORES

☐ DOIS ☐ TRÊS

MODO

☐ CONTROLE

☐ COMANDO

☐ DASHBOARD

A partir da interface visual, é feita a lógica que dará funcionalidade aos comandos do usuário. Para isso, utilizou-se a linguagem Java. Observando as opções que a tela inicial oferece, o usuário tem 18 caminhos diferentes a seguir por meio do botão “Avançar”. Por exemplo: broker local LEM com dois atuadores e modo de controle é um caminho. Todos os 18 caminhos são verificados no código em Java, além dos cliques nos botões, o botão “Sair” fecha o aplicativo e o botão “Avançar” leva à próxima tela. A Figura 36 apresenta um trecho do código em Java referente à tela inicial.

Figura 36 – Trecho de código Java para a tela inicial

```

○ ○ ○

findViewById(R.id.confirm_button).setOnClickListener(v -> {
if (localLEM_rb.isChecked() && doisAtuad_rb.isChecked() &&
controle_rb.isChecked()) {
    // Crie um Intent para iniciar DoisAtuad
    Intent localLEM_DoisAtuad = new
Intent(MainActivity.this, DoisAtuad.class);
    localLEM_DoisAtuad.putExtra("MQTT_BROKER",
"10.32.80.12");
    localLEM_DoisAtuad.putExtra("MQTT_USUARIO",
"localufulem");
    localLEM_DoisAtuad.putExtra("MQTT_SENHA",
"r4spb3rryp!@L3M");
    // Inicie a DoisAtuad com o Intent
    startActivity(localLEM_DoisAtuad);
} else if (localLEM_rb.isChecked() && doisAtuad_rb.isChecked() &&
comando_rb.isChecked()) {
    // Crie um Intent para iniciar ComandoDoisAtuad
    Intent localLEM_ComandoDoisAtuad = new
Intent(MainActivity.this, ComandoDoisAtuad.class);
    localLEM_ComandoDoisAtuad.putExtra("MQTT_BROKER",
"10.32.80.12");
    localLEM_ComandoDoisAtuad.putExtra("MQTT_USUARIO",
"localufulem");
    localLEM_ComandoDoisAtuad.putExtra("MQTT_SENHA",
"r4spb3rryp!@L3M");
    // Inicie a ComandoDoisAtuad com o Intent
    startActivity(localLEM_ComandoDoisAtuad);
}
}

```

O processo é análogo para todas as telas: inicialmente, cria-se a interface visual com código XML e, a partir daí, desenvolvem-se as funções em Java. Ao clicar em “Avançar”, o aplicativo inicia a conexão com o broker MQTT escolhido pelo usuário. Para isso, foi necessário utilizar a biblioteca *Eclipse Paho Client* MQTT, incluindo algumas configurações no código Java, conforme mostrado na Figura 37.

Figura 37 – Introduções da biblioteca MQTT no código Java

```

import org.eclipse.paho.client.mqttv3.IMqttActionListener;
import org.eclipse.paho.client.mqttv3.IMqttDeliveryToken;
import org.eclipse.paho.client.mqttv3.IMqttToken;
import org.eclipse.paho.client.mqttv3.MqttCallbackExtended;
import org.eclipse.paho.client.mqttv3.MqttClient;
import org.eclipse.paho.client.mqttv3.MqttConnectOptions;
import org.eclipse.paho.client.mqttv3.MqttException;
import org.eclipse.paho.client.mqttv3.MqttMessage;

```

Após iniciar a conexão com o broker MQTT, o aplicativo subscreve-se nos tópicos necessários, dependendo do modo (controle, comando ou dashboard) escolhido pelo usuário. Por exemplo, se o modo selecionado for “comando” com dois atuadores, o aplicativo subscreve-se nos tópicos referentes aos atuadores A e B. A Figura 38 mostra a função com a conexão MQTT e a subscrição nos tópicos para esse caso.

Figura 38 – Subscrição de tópicos em Java

```

mqtt = new MqttAndroidClient(this, MQTT_URL, MQTT_ID);
mqtt.setCallback(new MqttCallbackExtended() {
    @Override
    public void connectComplete(boolean reconnect, String
serverURI) {
        Log.w(TAG, "Reconexão MQTT..." + reconnect);
        statusConexaoDoisAtuad.setText(reconnect ?
"Reconectando..." : "Conectado!");
        if (mqtt.isConnected()) {
            String tsx = "CONECTADO!";
            statusConexaoDoisAtuad.setText(tsx);
            btnConectarDA.setVisibility(View.INVISIBLE);
            subscribe("/status/espSensores");
            subscribe("/status/espValvA");
            subscribe("/status/espValvB");
            subscribe("/estado/atuadorA");
            subscribe("/estado/atuadorB");
        }
    }
}

```

No exemplo acima, os tópicos que começam com “/status/” são usados para verificar se os microcontroladores estão conectados ao broker MQTT. Já os tópicos “/estado/atuadorA” e “/estado/atuadorB” recebem os estados (avançado ou recuado) dos atuadores por meio das mensagens publicadas pelo microcontrolador referente aos sensores de fim de curso.

O modo “controle” é o principal do aplicativo; nele, o usuário deve inserir as equações elaboradas utilizando o método trajeto-passo. Após o preenchimento e a confirmação, se as equações estiverem corretas, a sequência é iniciada, e o aplicativo realiza as publicações nos tópicos correspondentes, acionando a abertura e o fechamento das válvulas conforme a sequência programada. A Figura 39 apresenta a tela de controle para dois atuadores.

Figura 39 – Tela do aplicativo no modo de controle para dois atuadores

CONTROLE MQTT

A+ = Digite a equação de A+

A- = Digite a equação de A-

B+ = Digite a equação de B+

B- = Digite a equação de B-

K = Digite a equação de K

STATUS BROKER

VOLTAR

CONFIRMAR

CONECTAR

Para ilustrar como são realizadas as publicações nos tópicos, a Figura 40 apresenta o código Java das funções publishAmais() e publishAmenos(), responsáveis pelo avanço e recuo do atuador A, respectivamente.

Figura 40 – Código Java para as funções de avanço e recuo do atuador A

```
○ ○ ○

    public void publishAmais() {
        String topic = "/comando/valvA";
        String payload = "1"; // Avança o atuador A
        try {
            byte[] encodedPayload =
payload.getBytes(StandardCharsets.UTF_8);
            MqttMessage message = new MqttMessage(encodedPayload);
            mqtt.publish(topic, message);
        } catch (MqttException e) {
            e.printStackTrace();
        }
    }

    public void publishAmenos() {
        String topic = "/comando/valvA";
        String payload = "0"; // Recua o atuador A
        try {
            byte[] encodedPayload =
payload.getBytes(StandardCharsets.UTF_8);
            MqttMessage message = new MqttMessage(encodedPayload);
            mqtt.publish(topic, message);
        } catch (MqttException e) {
            e.printStackTrace();
        }
    }
}
```

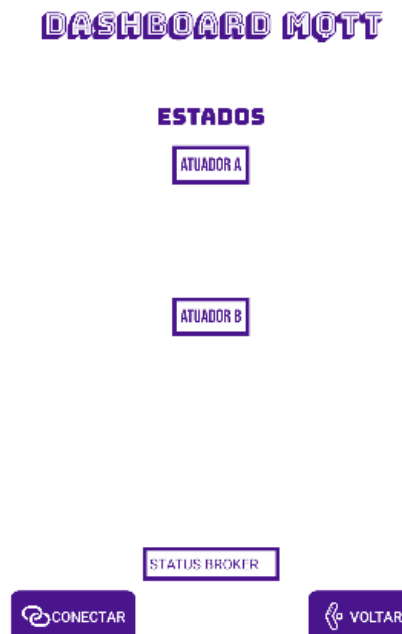
No modo “comando”, além de observar o estado de cada atuador, o usuário consegue avançar e recuar os mesmos por meio de botões. Assim, uma função é atribuída aos eventos de clique de cada botão. Ao clicar no botão “Avançar A”, o aplicativo publica a mensagem “1” no tópico “/comando/valvA” e, para o recuo, publica a mensagem “0” no mesmo tópico. A Figura 41 apresenta a tela de comando para dois atuadores.

Figura 41 – Tela do aplicativo no modo de comando para dois atuadores



Por fim, o modo “dashboard” é responsável por exibir, em tempo real, o estado de cada atuador, mostrando um desenho esquemático dos atuadores, indicando se estão avançados ou recuados. A Figura 42 mostra a tela do modo “dashboard” para dois atuadores.

Figura 42 – Tela do aplicativo no modo de dashboard para dois atuadores



Ao todo, foi necessário criar nove telas com botões funcionais para atender aos 18 diferentes caminhos que o usuário pode escolher. Para melhor compreensão do funcionamento

do aplicativo, a Tabela 7 apresenta os tópicos que são publicados e subscritos, bem como suas respectivas atribuições.

Tabela 7 – Tópicos do aplicativo

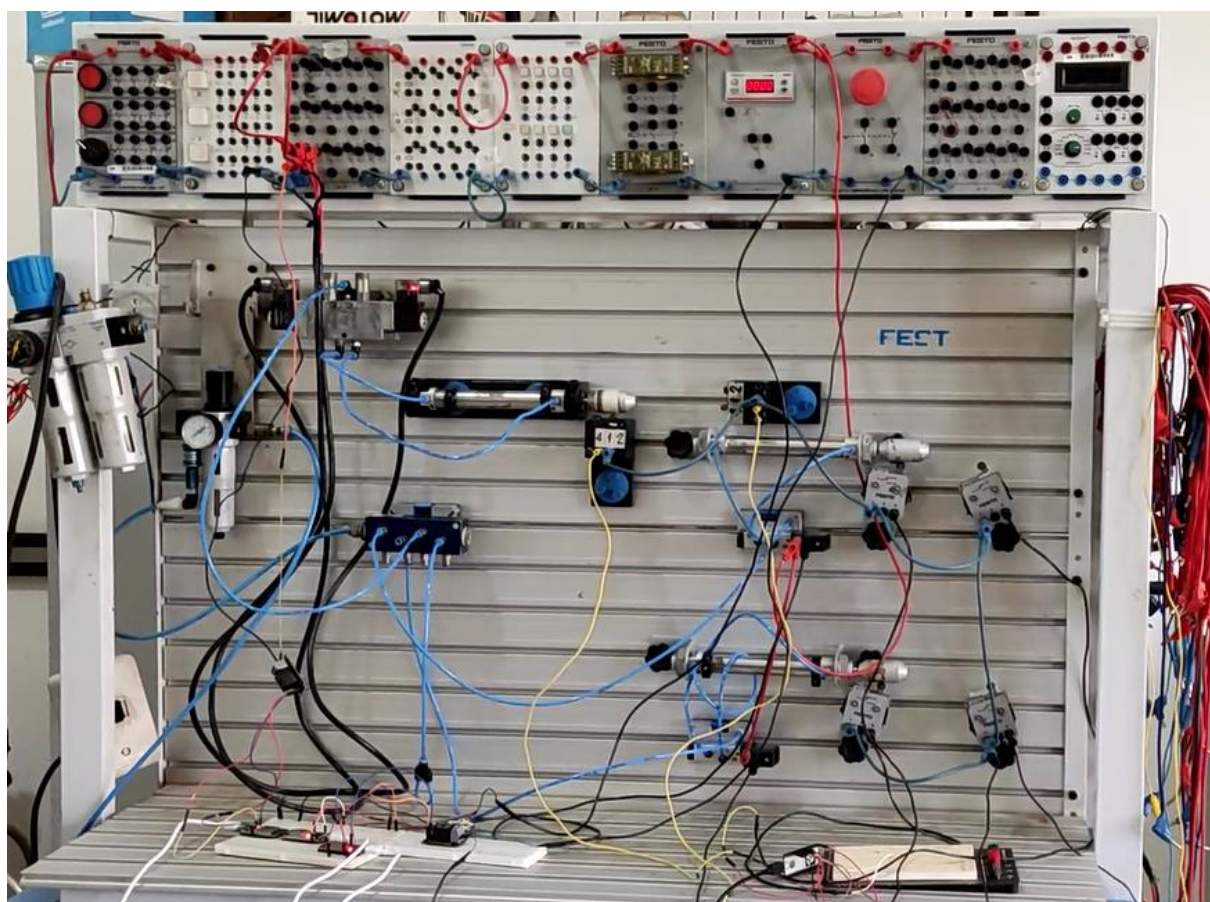
Tópico	Atribuição
/status/espValvA	Recebe o status de conexão a cada 3 segundos
/comando/valvA	Publica no tópico os comandos para abrir e fechar a válvula A
/status/espValvB	Recebe o status de conexão a cada 3 segundos
/comando/valvB	Publica no tópico os comandos de abrir e fechar a válvula B
/status/espValvC	Recebe o status de conexão a cada 3 segundos
/comando/valvC	Publica no tópico os comandos de abrir e fechar a válvula C
/status/espSensores	Recebe o status de conexão a cada 3 segundos
/estado/atuadorA	Recebe o estado do atuador A (avançado ou recuado)
/estado/atuadorB	Recebe o estado do atuador B (avançado ou recuado)
/estado/atuadorC	Recebe o estado do atuador C (avançado ou recuado)

Nos apêndices H e I estão, respectivamente, os códigos XML e Java da primeira tela. O repositório do GitHub mencionado anteriormente contém os códigos de todas as telas e o código-fonte do aplicativo.

4 RESULTADOS

Nesta seção serão apresentados os resultados obtidos na prática com a bancada eletropneumática sendo controlada pelo aplicativo. Para isso, é necessário, primeiramente, montar a bancada eletropneumática com todos os dispositivos apresentados ao longo deste relatório, conectando-os de modo a formar o sistema eletropneumático. A Figura 43 ilustra a montagem da bancada eletropneumática instalada no LEM-04 da UFU.

Figura 43 – Bancada eletropneumática



As válvulas eletropneumáticas de cada atuador foram alimentadas pela fonte de 24 V da bancada. Além disso, essa mesma fonte fornece alimentação aos módulos de relé responsáveis pelo acionamento das válvulas.

Após a realização das ligações dos componentes eletropneumáticos, estabeleceram-se as conexões com os microcontroladores para implementar a comunicação MQTT. Para cada válvula, há um microcontrolador e um módulo relé responsáveis pelo controle de sua abertura e fechamento, conforme descrito na arquitetura do projeto. O microcontrolador envia o sinal ao relé, que, por sua vez, fecha o circuito da válvula — previamente alimentada com 24 V no polo positivo — conectando-o ao polo negativo (0 V) e, assim, acionando-a. Todos os GND das

ligações são comuns, ou seja, provenientes da mesma fonte da bancada eletropneumática. Além disso, os microcontroladores são alimentados com uma fonte de 5 VDC. As Figuras 44 e 45 mostram, respectivamente, o esquema da ligação do ESP32 com o módulo relé e a montagem prática desse esquema. Lembre-se de que a ligação é análoga para as três válvulas.

Figura 44 – Esquema da ligação do ESP32 com o módulo relé

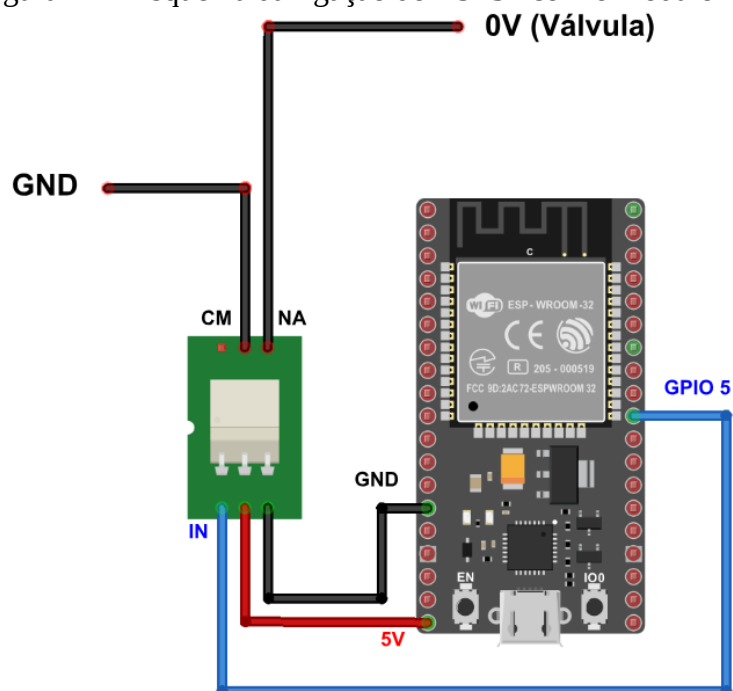
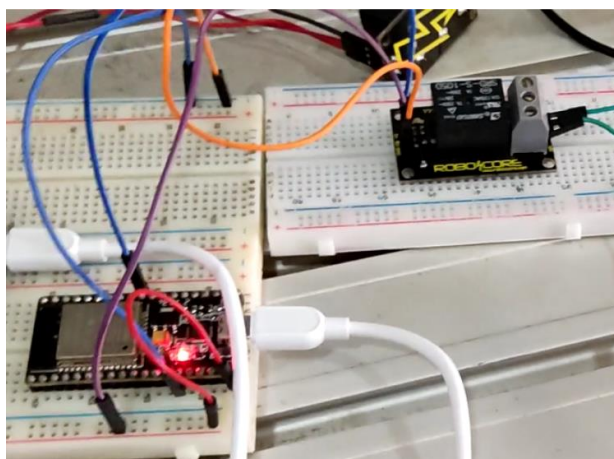


Figura 45 – Montagem da ligação do ESP32 com o módulo relé



Por fim, realizam-se as conexões do microcontrolador referentes aos sensores de fim de curso. O sinal de cada sensor de fim de curso é conectado a uma porta digital do ESP32. Ademais, os terminais de GND são comuns e compartilhados com a fonte da bancada eletropneumática, e o ESP32 é alimentado por uma fonte de 5 VDC. As Figuras 46 e 47 mostram, respectivamente, o esquema de ligação dos sensores de fim de curso ao ESP32 e sua montagem prática.

Figura 46 – Esquema da ligação do ESP32 com os sensores de fim de curso

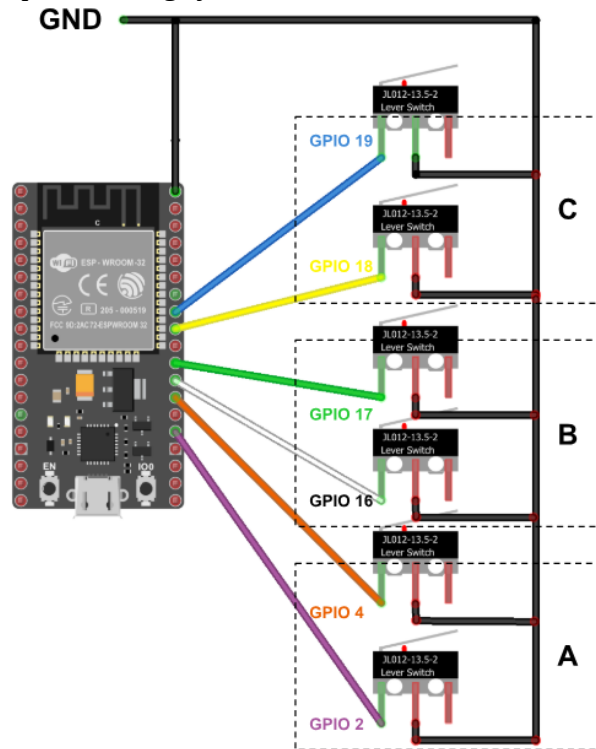
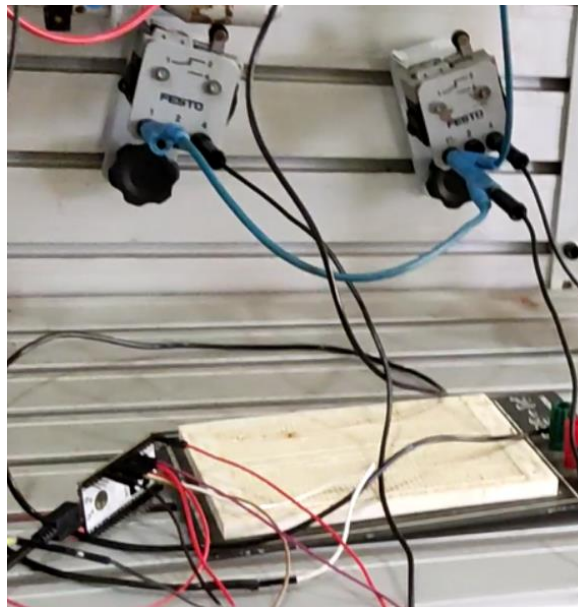


Figura 47 – Montagem da ligação do ESP32 com os sensores de fim de curso



Com o broker ligado e o aplicativo inicializado, iniciou-se o controle da bancada eletropneumática utilizando as equações obtidas pelo método trajeto-passo das seguintes sequências: $A+ B+ A- B-$, $A+ B+ B- A-$, $A+ A- B+ B-$, e $A+ B+ C+ C- B- A-$. Aplicando o método, obteve-se as equações apresentadas na Tabela 8.

Tabela 8 – Equações para as sequências eletropneumáticas

Sequência	Equações
A+B+A-B-	A+ b0.S
	A- b1
	B+ a1
	B- a0
	K
A+B+B-A-	A+ not(A-)
	A- b0.KNA
	B+ a1.KNF
	B- not(B+)
	K (b1+KNA).not(a0)
A+A-B+B-	A+ b0.KNF
	A- not(A+)
	B+ a0.KNA
	B- not(B+)
	K (a1+KNA).not(b1)
A+B+C+C-B-A-	A+ S
	A- b0.KNF
	B+ a1.KNA
	B- c0.KNF
	C+ b1.KNA
	C- KNF+not(b1)
	K (a0+KNA).not(c1)

A Figura 48 mostra o preenchimento das equações para a sequência A+ B+ A- B- nos campos de entrada do aplicativo. Ao clicar em “Confirmar”, ocorre uma transição de tela na qual é solicitado o número de ciclos desejado; após inserir essa informação, o ciclo é iniciado. A Figura 49 apresenta o sistema em funcionamento, permitindo visualizar em tempo real, na

bancada eletropneumática, os dados exibidos na tela. Neste frame específico, observa-se o recuo do atuador A.

Figura 48 – Preenchimentos das equações para a sequência A+ B+ A- B-

CONTROLE MQTT

A+ = b0.S

A- = b1

B+ = a1

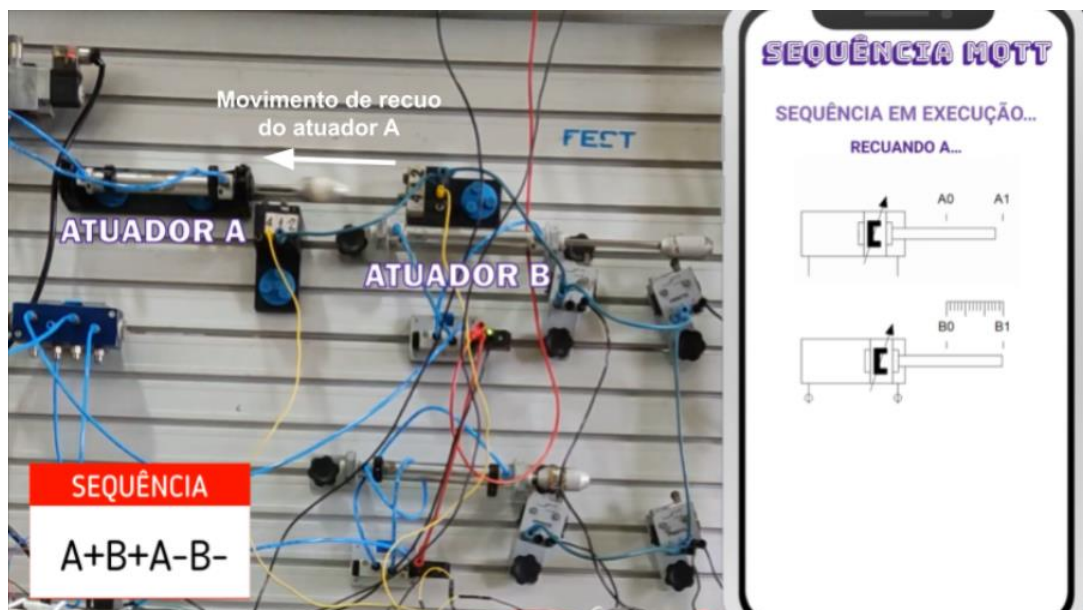
B- = a0

K = Digite a equação de K

STATUS BROKER CONECTADO!

🔙 VOLTAR
CONFIRMAR ✓

Figura 49 – Sequência A+ B+ A- B- em andamento



A aplicação atendeu aos requisitos de projeto, realizando a troca de mensagens entre os dispositivos conectados por meio do protocolo MQTT utilizando o broker local instalado no Raspberry Pi. Observou-se também que o sistema opera em malha fechada, ou seja, o próximo movimento só ocorre após a confirmação de que o anterior foi executado. No frame apresentado anteriormente, o recuo do atuador A corresponde à publicação da mensagem “0” no tópico “/comando/valvA” feita pelo aplicativo, mas essa publicação só ocorre quando o aplicativo recebe a mensagem “Atuador B Avançado” no tópico “/estado/atuadorB”, enviada pelo

microcontrolador dos sensores de fim de curso referente ao movimento anterior da sequência. Ademais, não houve atraso no envio das publicações; contudo, devido ao tempo de processamento do estado dos atuadores – de um segundo –, alguns movimentos sofreram atraso, pois o movimento seguinte só é iniciado após a verificação do anterior. Ao finalizar a sequência, o aplicativo retorna à tela inicial.

O procedimento é análogo para as demais sequências. Deve-se, inicialmente, inserir as equações obtidas pelo método trajeto-passo; se estiverem corretas, a confirmação é aceita e a sequência é iniciada após a escolha do número de ciclos. As Figuras 50, 51 e 52 apresentam um frame da aplicação em andamento, referentes às sequências $A+ B+ B- A-$, $A+ A- B+ B-$ e $A+ B+ C+ C- B- A-$.

Figura 50 – Sequência $A+ B+ B- A-$ em andamento

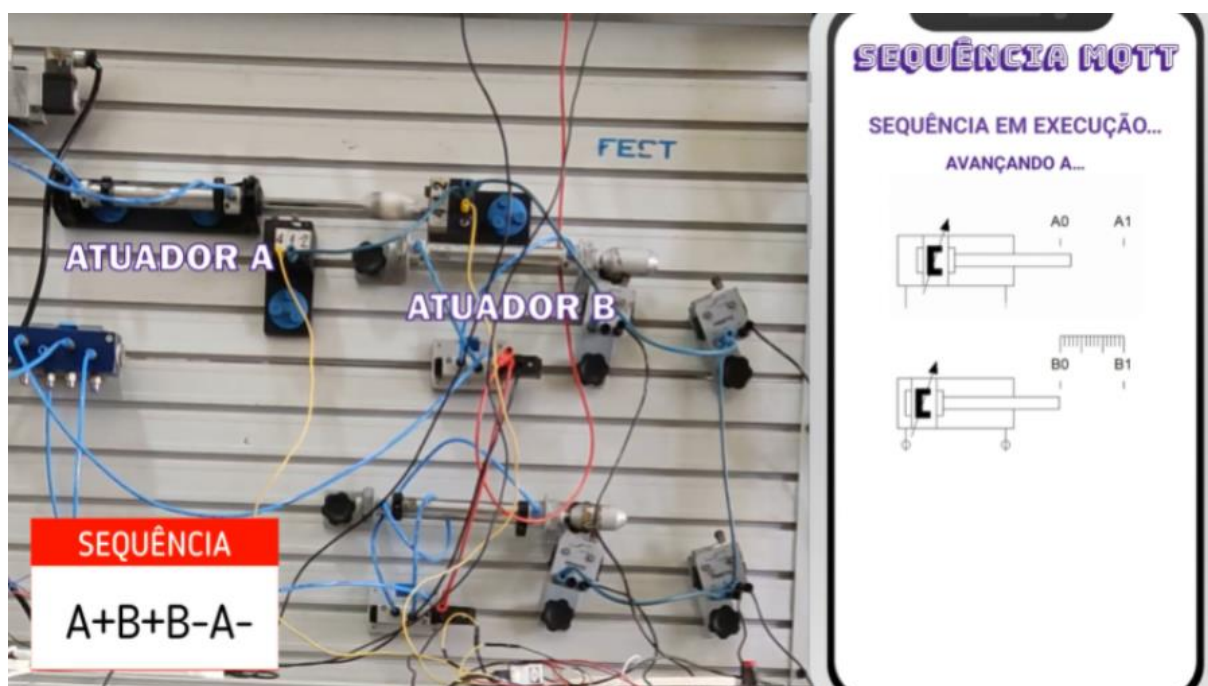


Figura 51 – Sequência A+ A– B+ B– em andamento

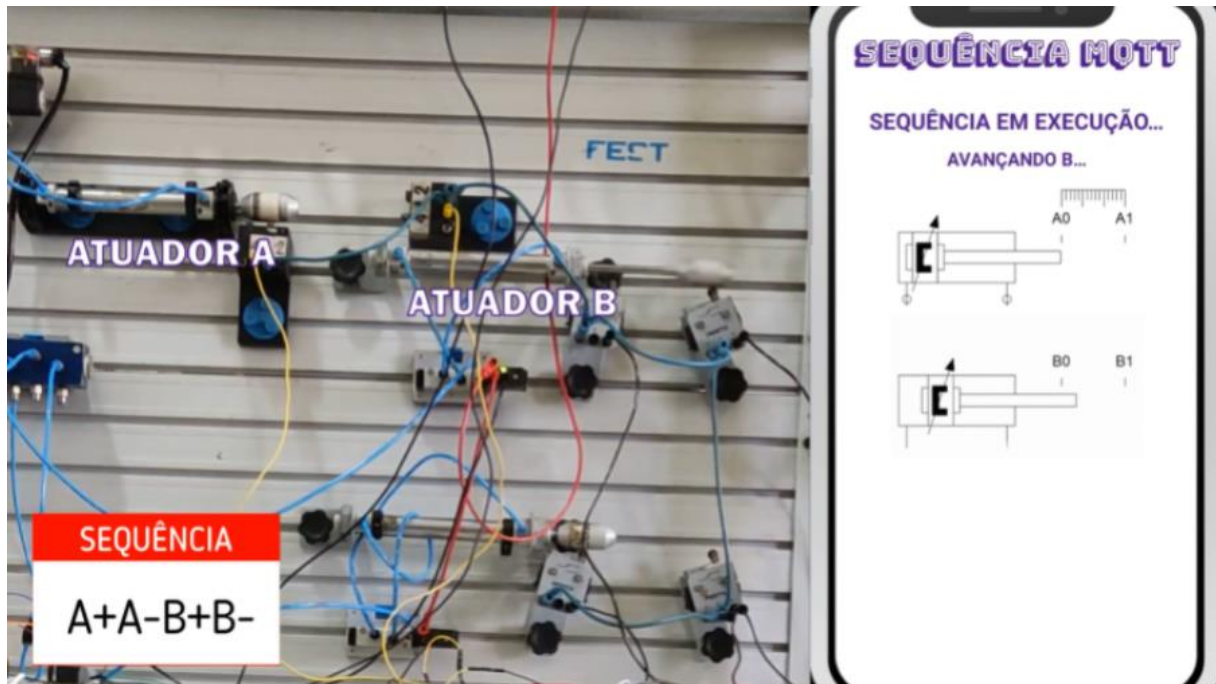
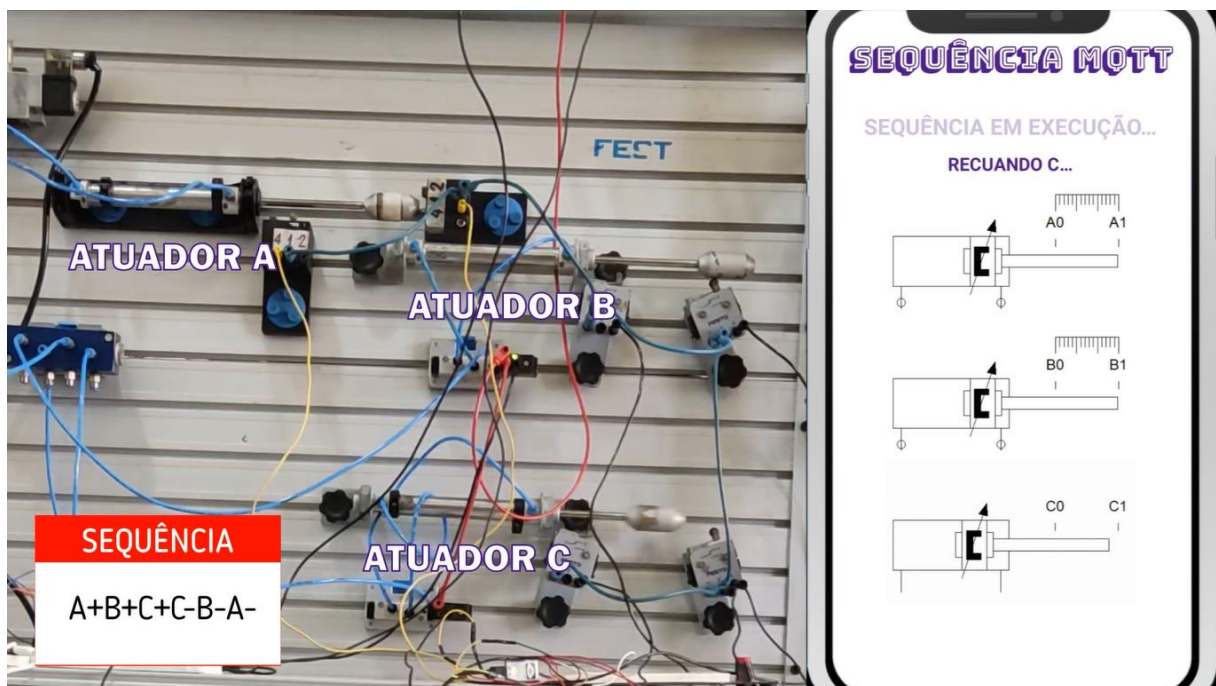


Figura 52 – Sequência A+B+C+C–B–A– em andamento



A aplicação nos modos “comando” e “dashboard” é executada em uma única tela para cada um desses modos. Nesses casos, utilizou-se um broker em nuvem para realizar a conexão e a transmissão de mensagens pelo protocolo MQTT. Observou-se que, mesmo utilizando um broker externo à universidade, não houve atraso perceptível no envio das publicações.

As Figuras 53 e 54 apresentam um frame da aplicação em execução nesses modos, respectivamente.

Figura 53 – Aplicação em execução no modo “comando”

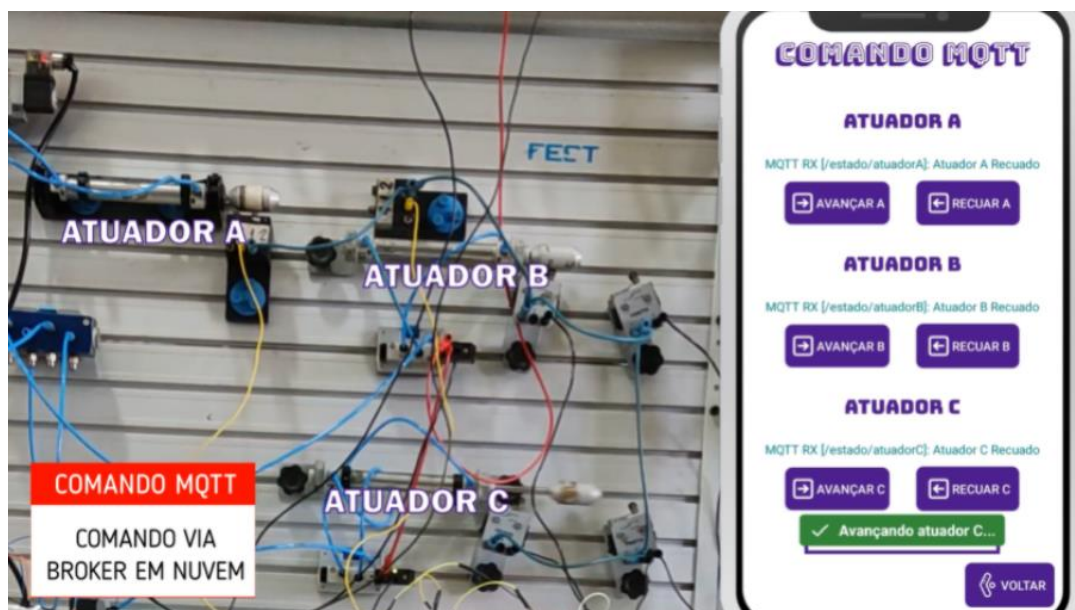
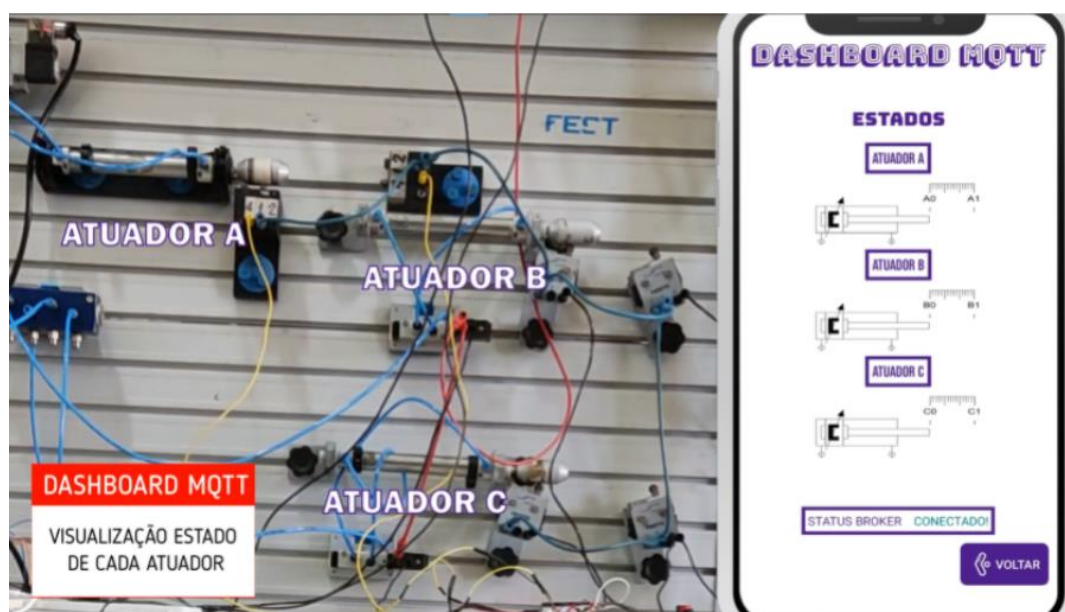


Figura 54 – Aplicação em execução no modo “dashboard”



Os resultados em vídeos da aplicação podem ser encontrados na plataforma do YouTube².

² <https://www.youtube.com/playlist?list=PLzDBC7KLK1vOb4CWltdMiIS3cEcgx-8gT>

5 CONCLUSÃO

O desenvolvimento deste projeto demonstrou que o protocolo de comunicação MQTT é um pilar da Internet Industrial das Coisas, devido à sua flexibilidade e facilidade de implementação. A demonstração prática, com o microcontrolador ESP32 e o microcomputador Raspberry Pi controlando a bancada eletropneumática, evidencia como o MQTT pode ser implementado com rapidez, permitindo ao engenheiro mecatrônico explorar as possibilidades da IIoT.

A partir dos estudos realizados e dos resultados obtidos, concluiu-se o êxito no desenvolvimento de um aplicativo móvel para controlar a bancada eletropneumática via MQTT, utilizando Raspberry Pi 4 e ESP32, alcançando os objetivos descritos no início deste relatório. Para tal, foi necessário aprofundar os estudos sobre comunicação MQTT entre dispositivos, metodologias sequenciais para formulação de equações e programação em Java para a criação do aplicativo. A implementação dessas tecnologias mostrou-se bastante eficaz quanto ao tempo de resposta e à qualidade de conexão, comprovando sua capacidade de viabilizar o controle remoto da bancada.

Os resultados, obtidos tanto em conexão com o broker local quanto com o broker em nuvem, demonstram que o protocolo MQTT, aliado às plataformas ESP32 e Raspberry Pi, atende às necessidades de aplicações em ambientes industriais e acadêmicos. Essas tecnologias são fundamentais para a formação do estudante de engenharia mecatrônica, aperfeiçoando os métodos didáticos das disciplinas com os conceitos da Indústria 4.0.

É importante destacar que o objetivo principal deste projeto foi puramente didático, concentrando-se no entendimento dos mecanismos de troca de mensagens e na implementação prática das funcionalidades básicas do protocolo MQTT. Por essa razão, não foram abordadas as camadas de segurança, tais como autenticação de clientes, autorização de tópicos ou criptografia de payload e de canal (TLS/SSL). Assim, recomenda-se que, em cenários de produção ou em aplicações que exijam maior proteção de dados e controle de acesso, sejam incorporadas soluções adequadas de segurança ao protocolo MQTT.

6 TRABALHOS FUTUROS

Apesar de terem sido atingidos os objetivos do projeto, observou-se que alguns aspectos podem ser aprimorados para otimizar seu funcionamento. Os trabalhos a serem desenvolvidos, estão citados nos itens abaixo:

1. Interface Homem–Máquina (IHM) para microcontroladores

Implementar uma IHM que permita ao usuário selecionar, por meio de interface gráfica, o broker a ser utilizado. No projeto atual, essa escolha só é feita recompilando e enviando o código para o ESP32. O ESP32, equipado com dois núcleos e memória interna para armazenamento de dados, revela-se robusto e adequado para hospedar essa IHM.

2. Configuração de modo *bridge* no broker local

Habilitar o modo *bridge* no broker local para permitir que dispositivos conectados a outros brokers (por exemplo, na nuvem) comuniquem-se entre si. Durante os estudos para o desenvolvimento deste projeto, verificou-se que a configuração do modo bridge precisa ser aplicada em ambos os brokers envolvidos.

3. Aprimoramentos no aplicativo móvel e disponibilidade para sistema iOS

Incluir novas sequências eletropneumáticas para ampliar as funcionalidades de controle da bancada. Refinar a lógica de verificação das equações, garantindo maior robustez e flexibilidade na interpretação dos comandos. Implementar o aplicativo para o sistema iOS, sistema operacional móvel da Apple, utilizado nos iPhones.

4. Descentralização da lógica nos ESP's (sensores ou válvulas)

Um trabalho futuro sobre a descentralização da lógica em ESPs (sensores ou válvulas) envolveria o deslocamento das funções de controle e decisão do servidor central para os próprios nós embarcados, aproveitando a capacidade de computação local dos microcontroladores ESP32.

Este conjunto de melhorias não só elevará a usabilidade e a flexibilidade do sistema, mas também servirá de base para futuras atualizações voltadas a aplicações mais complexas na Indústria 4.0.

REFERÊNCIAS

BOLLMANN, A. Fundamentos da Automação Industrial Pneumática – Projeto de comandos binários eletropneumáticos. São Paulo: ABHP, 1997.

FIALHO, A. B. Automação Pneumática – Projetos, dimensionamento e análise de circuitos. 7. ed. São Paulo: Editora Érica, 2012.

PAVANI, S. A. Comandos Pneumáticos e Hidráulicos. Santa Maria: e-Tec Brasil, 2011.

TAVARES, J. J.-P. Z. D. S. Módulo 1 de Automação Industrial. Uberlândia, 2022.

MQTT. MQTT: The Standard for IoT Messaging, 2024. Disponível em: <<https://mqtt.org/>>. Acesso em: 10 fev. 2024.

Eclipse Mosquitto™. An open source MQTT broker, 2024. Disponível em: <<https://mosquitto.org/>>. Acesso em: 10 fev. 2024.

Alfacomp. MQTT – Conheça o protocolo padrão do IoT, 2024. Disponível em: <<https://alfacomp.net/2022/03/24/mqtt-conheca-o-protocolo-padrao-do-iot/>>. Acesso em: 10 fev. 2024.

Embarcados. MQTT – Protocolos para IoT, 2024. Disponível em: <<https://embarcados.com.br/mqtt-protocolos-para-iot/>>. Acesso em: 10 fev. 2024.

Automação Industrial. Protocolo MQTT: O Que é, Como Funciona e Vantagens, 2024. Disponível em: <<https://www.automacaoindustrial.info/mqtt/>>. Acesso em: 10 fev. 2024.

RFWirelessWorld. MQTT Architecture: Message Types, Format, and Flow, 2024. Disponível em: <<https://www.rfwireless-world.com/tutorials/iot/mqtt-architecture-message-types-format-flow>>. Acesso em: 10 fev. 2024.

OASIS. MQTT Version 3.1.1, 2024. Disponível em: <https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/cs01/mqtt-v3.1.1-cs01.html#_MQTT_Control_Packet>. Acesso em: 10 fev. 2024.

Escola Politecnica UFRJ. MQTT – Como funciona?, 2024. Disponível em: <<https://www.gta.ufrj.br/ensino/eel878/redes1-2019-1/vf/mqtt/>>. Acesso em: 11 fev. 2024.

Instituto NCB. Comunicando-se via MQTT com o ESP32 (MIC404), 2024. Disponível em: <<https://www.newtoncbraga.com.br/index.php/microcontroladores/143-tecnologia/17117-comunicando-se-via-mqtt-com-o-esp32-mic404.html>>. Acesso em: 11 fev. 2024.

Raspberry Pi. Raspberry Pi 4, 2024. Disponível em: <<https://www.raspberrypi.com/products/raspberry-pi-4-model-b/>>. Acesso em: 19 abr. 2024.

Tecnoblog. O que é o Raspberry Pi? Saiba para que serve e como funciona o minicomputador, 2024. Disponível em: <<https://tecnoblog.net/responde/o-que-e-o-raspberry-pi/>>. Acesso em: 19 abr. 2024.

The Engineering Projects. What is Raspberry Pi 4? Pinout, Specs, Projects & Datasheet, 2024. Disponível em: <<https://www.theengineeringprojects.com/2021/03/what-is-raspberry-pi-4-pinout-specs-projects-datasheet.html>>. Acesso em: 19 abr. 2024.

Espressif. ESP32, 2024. Disponível em: <<https://www.espressif.com/en/products/socs/esp32>>. Acesso em: 25 abr. 2024.

UsinaInfo. ESP32 Tutorial com Primeiros Passos, 2024. Disponível em: <<https://www.usinainfo.com.br/blog/esp32-tutorial-com-primeiros-passos/>>. Acesso em: 25 abr. 2024.

Arduino. Arduino IDE 2.0, 2024. Disponível em: <<https://www.arduino.cc/en/software/>>. Acesso em: 25 abr. 2024.

RoboCore. Instalando o Arduino IDE, 2024. Disponível em: <<https://www.robocore.net/tutoriais/instalando-arduino-ide>>. Acesso em: 25 abr. 2024.

Shiftr.io. The IoT Platform for Interconnected Projects, 2024. Disponível em: <<https://www.shiftr.io/>>. Acesso em: 7 mai. 2024.

Random Nerd Tutorials. Install Mosquitto MQTT Broker on Raspberry Pi, 2024. Disponível em: <<https://randomnerdtutorials.com/how-to-install-mosquitto-broker-on-raspberry-pi/>>. Acesso em: 13 mai. 2024.

Mosquitto. mosquitto_pub man page, 2024. Disponível em: <https://mosquitto.org/man/mosquitto_pub-1.html>. Acesso em: 15 mai. 2024.

Mosquitto. mosquitto_sub man page, 2024. Disponível em: <https://mosquitto.org/man/mosquitto_sub-1.html>. Acesso em: 15 mai. 2024.

Blog Eletrogate. Automação com Broker MQTT e Raspberry Pi, 2024. Disponível em: <<https://blog.eletrogate.com/automacao-residencial-com-broker-mqtt-local-com-raspberry-pi/>>. Acesso em: 25 mai. 2024.

Arduino e Cia. Enviando mensagens MQTT com módulo ESP32, 2024. Disponível em: <<https://www.arduinoecia.com.br/enviando-mensagens-mqtt-modulo-esp32-wifi/>>. Acesso em: 4 jun. 2024.

EMQX. Android and MQTT: Step by Step Guide, 2024. Disponível em: <<https://www.emqx.com/en/blog/android-connects-mqtt-using-kotlin>>. Acesso em: 18 ago. 2024.

EMQX. MQTT with Java: A Beginner's Guide with Examples & FAQs, 2024. Disponível em: <<https://www.emqx.com/en/blog/how-to-use-mqtt-in-java>>. Acesso em: 18 ago. 2024.

GitHub. MQTT Clients Examples, 2024. Disponível em: <<https://github.com/emqx/MQTT-Client-Examples/tree/master/mqtt-client-Java>>. Acesso em: 24 ago. 2024.

GitHub. Public Brokers, 2024. Disponível em: <https://github.com/mqtt/mqtt.org/wiki/public_brokers>. Acesso em: 19 set. 2024.

7 APÊNDICE A – CÓDIGO DE CONFIGURAÇÃO DHCPD

```
#####
## Seção com as nossas configurações ##
#####

interface eth0
static ip_address=10.32.80.12/22
static routers=10.32.80.1
static domain_name_servers=10.32.80.1 8.8.8.8 8.8.4.4

#####
## Seção com as configurações padrão ##
#####

# A sample configuration for dhcpd.
# See dhcpd.conf(5) for details.

# Allow users of this group to interact with dhcpd via the control socket.
#controlgroup wheel

# Inform the DHCP server of our hostname for DDNS.
hostname

# Use the hardware address of the interface for the Client ID.
clientid
# or
# Use the same DUID + IAID as set in DHCPv6 for DHCPv4 ClientID as per RFC4361.
# Some non-RFC compliant DHCP servers do not reply with this set.
# In this case, comment out duid and enable clientid above.
#duid

# Persist interface configuration when dhcpd exits.
persistent
```

```

# Rapid commit support.
# Safe to enable by default because it requires the equivalent option set
# on the server to actually work.
option rapid_commit

# A list of options to request from the DHCP server.
option domain_name_servers, domain_name, domain_search, host_name
option classless_static_routes
# Respect the network MTU. This is applied to DHCP routes.
option interface_mtu

# Most distributions have NTP support.
#option ntp_servers

# A ServerID is required by RFC2131.
require dhcp_server_identifier

# Generate SLAAC address using the Hardware Address of the interface
#slaac hwaddr
# OR generate Stable Private IPv6 Addresses based from the DUID
slaac private

# Example static IP configuration:
#interface eth0
#static ip_address=192.168.0.10/24
#static ip6_address=fd51:42f8:caae:d92e::ff/64
#static routers=192.168.0.1
#static domain_name_servers=192.168.0.1 8.8.8.8 fd51:42f8:caae:d92e::1

```

8 APÊNDICE B – CÓDIGO DE CONFIGURAÇÃO INTERFACES

```
# interfaces(5) file used by ifup(8) and ifdown(8)
# Include files from /etc/network/interfaces.d:
source /etc/network/interfaces.d/*

auto eth0
iface eth0 inet dhcp
```

9 APÊNDICE C – CÓDIGO DE CONFIGURAÇÃO MOSQUITTO

```
# Place your local configuration in /etc/mosquitto/conf.d/
#
# A full description of the configuration file is at
# /usr/share/doc/mosquitto/examples/mosquitto.conf.example

pid_file /run/mosquitto/mosquitto.pid

persistence true
persistence_location /var/lib/mosquitto/

log_dest file /var/log/mosquitto/mosquitto.log

include_dir /etc/mosquitto/conf.d

allow_anonymous true
password_file /etc/mosquitto/pwfile
listener 1883
```

10 APÊNDICE D – CÓDIGO ESP32 DA VÁLVULA A

```
/*
```

```
ESP_ValvA
```

PROJETO: Desenvolvimento de um sistema, utilizando protocolo MQTT, com Raspberry Pi 4 e ESP32 para controlar bancada eletropneumática através de um aplicativo mobile.

LINK: <https://github.com/MAPL-UFU/mqtt-app-controle>

ALUNO: Matheus Alves de Paula

PROF.: Prof. Dr. José Jean-Paul Zanlucchi de Souza Tavares

DATA: 05/01/2024

```
*/
```

```
// INCLUSÃO DE BIBLIOTECAS
```

```
#include <WiFi.h>          // biblioteca WiFi
```

```
#include "esp_wpa2.h"      // biblioteca wpa2 para conexão em redes Enterprise
```

```
#include <PubSubClient.h> // biblioteca PubSubClient para conexão MQTT
```

```
byte mac[6];
```

```
// PARÂMETROS PARA CONEXÃO WIFI COM A REDE EDUROAM
```

```
#define EAP_ANONYMOUS_IDENTITY "anonymous@ufu.br" //
```

anonymous@example.com, or you can use also nickname@example.com

```
#define EAP_IDENTITY "@ufu.br" // nickname@example.com, at some organizations  
should work nickname only without realm, but it is not recommended
```

```
#define EAP_PASSWORD "" // password for eduroam account
```

```
const char* ssid = "eduroam"; // "eduroam" SSID
```

```
// PARÂMETROS DO BROKER PARA CONEXÃO MQTT
```

```
const char* mqttServer = "broker.hivemq.com"; // endereço do broker a ser conectado.
```

Nuvem: "ufumapl.cloud.shiftr.io"; LEM: "10.32.80.12"; Casa: "179.104.38.174"

```
const int mqttPort = 1883; // porta mqtt
```

```
const char* mqttUser = ""; // usuário mqtt do referido broker. Nuvem: "ufumapl"; LEM:  
"localufulem"; Casa: "localufumapl"
```

```
const char* mqttPassword = ""; // senha mqtt do referido broker. Nuvem:
"p0dy0EfwVhUSzXlC"; LEM: "r4spb3rryp!@L3M"; Casa: "r4spb3rryp!@M4PL"
```

```
// INSTANCIANDO OBJETOS DAS CLASSES DAS BIBLIOTECAS
```

```
WiFiClient espClient; // definindo objeto espClient
```

```
PubSubClient MQTT(espClient); // cliente MQTT passando o objeto espClient
```

```
// INSTANCIANDO OBJETOS TASKHANDLE PARA PROCESSAMENTO DUAL CORE
```

```
TaskHandle_t Task1;
```

```
TaskHandle_t Task2;
```

```
// DEFINIÇÃO DO CLIENT ID
```

```
#define id_mqtt "ESP-VALV-A" // definindo identificador para o esp32. "ESP-
B0A732DA91C8" "ESP-E05A1B779924" "ESP-AC67B23BF1A0"
```

```
// DEFINIÇÃO DE PINOS
```

```
#define valvA 5 // definindo a ligação da válvula na porta 5
```

```
// DECLARAÇÃO DOS PROTÓTIPOS DAS FUNÇÕES
```

```
void conectaWiFi(); // realiza conexão com wifi
```

```
void conectaMQTT(); // realiza conexão com broker mqtt
```

```
void mantemConexoes(); // garantir as conexoes wifi e broker mqtt
ativas
```

```
void callback(char* topic, byte* message, unsigned int length); // função de retorno de chamada
MQTT
```

```
void openValve(); // função para abrir a válvula
```

```
void closeValve(); // função para fechar a válvula
```

```
void setup() {
```

```
Serial.begin(115200); // inicia a comunicação serial em 115200 bauds
```

```
delay(10);
```

```
Serial.print(F("Connecting to network: "));
```

```
Serial.println(ssid);
```

```

WiFi.disconnect(true);                                // disconnect from WiFi to set
new WiFi connection

```

```

WiFi.mode(WIFI_STA);                                // init wifi mode
WiFi.begin(ssid, WPA2_AUTH_PEAP, EAP_ANONYMOUS_IDENTITY,
EAP_IDENTITY, EAP_PASSWORD); // WITHOUT CERTIFICATE - WORKING WITH
EXCEPTION ON RADIUS SERVER

```

```

while (WiFi.status() != WL_CONNECTED) {
  delay(500);
  Serial.print(F("."));
  if (WiFi.status() == WL_CONNECT_FAILED || WiFi.status() ==
WL_CONNECTION_LOST) {
    conectaWiFi(); // reconecta Wi-Fi em caso de falha
  }
}

```

```

Serial.println("");
Serial.println(F("WiFi is connected!"));
Serial.println(F("IP address set: "));
Serial.println(WiFi.localIP()); // exibe LAN IP

```

```

WiFi.macAddress(mac);
Serial.print("MAC address: ");
Serial.print(mac[0], HEX);
Serial.print(":");
Serial.print(mac[1], HEX);
Serial.print(":");
Serial.print(mac[2], HEX);
Serial.print(":");
Serial.print(mac[3], HEX);
Serial.print(":");
Serial.print(mac[4], HEX);
Serial.print(":");

```

```
Serial.println(mac[5], HEX); // exibe MAC Address
```

```
// DECLARANDO OS PINOS
```

```
pinMode(valvA, OUTPUT); // define a saída para a válvula A como OUTPUT
```

```
digitalWrite(valvA, LOW); // estado inicial, a válvula A está fechada (LOW)
```

```
MQTT.setServer(mqttServer, mqttPort);
```

```
MQTT.setCallback(callback); // configura a função de retorno de chamada MQTT
```

```
conectaMQTT(); // chama a função para conexão com o broker MQTT
```

```
// cria uma tarefa que será executada na função Task1code(), com prioridade 1 e execução  
no core 0
```

```
xTaskCreatePinnedToCore(
```

```
    Task1code, /* função que implementa a tarefa. */
```

```
    "Task1", /* nome da tarefa. */
```

```
    10000, /* número de palavras a serem alocadas para uso com a pilha da tarefa */
```

```
    NULL, /* parâmetro de entrada para a tarefa (pode ser NULL) */
```

```
    1, /* prioridade da tarefa (0 a N) */
```

```
    &Task1, /* referência para a tarefa (pode ser NULL) */
```

```
    0); /* núcleo que executará a tarefa (0 ou 1) */
```

```
delay(500);
```

```
// cria uma tarefa que será executada na função Task2code(), com prioridade 1 e execução no  
core 1
```

```
xTaskCreatePinnedToCore(
```

```
    Task2code, /* função que implementa a tarefa. */
```

```
    "Task2", /* nome da tarefa. */
```

```
    10000, /* número de palavras a serem alocadas para uso com a pilha da tarefa */
```

```
    NULL, /* parâmetro de entrada para a tarefa (pode ser NULL) */
```

```
    1, /* prioridade da tarefa (0 a N) */
```

```
    &Task2, /* referência para a tarefa (pode ser NULL) */
```

```
    1); /* núcleo que executará a tarefa (0 ou 1) */
```

```
delay(500);
```

```
}
```

```
// FUNÇÃO EXECUTADA NO NÚCLEO 0
```

```
void Task1code(void* pvParameters) {
```

```
    Serial.print("Publish status running on core ");
```

```
    Serial.println(xPortGetCoreID());
```

```
    for (;;) {
```

```
        const char* msg_status = "ESP32 VALVULA A CONECTADO";
```

```
        MQTT.publish("/status/espValvA", msg_status); // publica o status do ESP32
```

```
        //Serial.print("Mensagem publicada no tópico: /status/espValvA");
```

```
        //Serial.print(" Mensagem: ");
```

```
        //Serial.println(msg_status);
```

```
        delay(3000);
```

```
    }
```

```
}
```

```
// FUNÇÃO EXECUTADA NO NÚCLEO 1
```

```
void Task2code(void* pvParameters) {
```

```
    Serial.print("Conexões (Wi-Fi e MQTT) running on core ");
```

```
    Serial.println(xPortGetCoreID());
```

```
    for (;;) {
```

```
        mantemConexoes(); // função para reconectar wifi ou broker em caso de queda
```

```
        MQTT.loop(); // mantém a comunicação MQTT ativa
```

```
        //delay(10);
```

```
    }
```

```
}
```

```
void loop() {
```

```
}
```

```
// FUNÇÃO DE RECONEXÃO MQTT E WIFI
```

```
void mantemConexoes() {
```

```
    if (!MQTT.connected()) {
```

```

    conectaMQTT();
}

conectaWiFi();
}

// FUNÇÃO CONEXÃO REDE WIFI
void conectaWiFi() {

    if (WiFi.status() == WL_CONNECTED) {
        return;
    }

    Serial.println("Falha na conexão com a rede WiFi.");
    Serial.print("Reconectando-se na rede: ");
    Serial.print(ssid);

    WiFi.disconnect(true); //disconnect from WiFi to set new WiFi connection
    WiFi.mode(WIFI_STA); //init wifi mode
    WiFi.begin(ssid, WPA2_AUTH_PEAP, EAP_ANONYMOUS_IDENTITY,
EAP_IDENTITY, EAP_PASSWORD);

    int counter = 0;
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
        counter++;

        if (counter >= 60) {
            ESP.restart(); // Reinicia o ESP32 após 30 segundos de tentativas de conexão
        }

        Serial.println();
        Serial.print("Conectado com sucesso na rede: ");

```

```

    Serial.println(ssid);
}
}

// FUNÇÃO CONEXÃO MQTT
void conectaMQTT() {
    while (!MQTT.connected()) {
        Serial.print("Conectando ao Broker MQTT: ");
        Serial.println(mqttServer);

        if (MQTT.connect(id_mqtt, mqttUser, mqttPassword)) {
            Serial.println("Conexao realizada com sucesso!");

            MQTT.subscribe("/comando/valvA");          // subscreve para receber comandos

        } else {
            Serial.println("Falha na conexão com o broker MQTT.");
            Serial.println("Nova tentativa de conexão em 5s...");
            delay(5000);
        }
    }
}

// FUNÇÃO DE RETORNO DE CHAMADA MQTT
void callback(char* topic, byte* message, unsigned int length) {
    Serial.print("Mensagem recebida no tópico: ");
    Serial.print(topic);
    Serial.print(". Mensagem: ");

    for (int i = 0; i < length; i++) {
        Serial.print((char)message[i]);
    }
    Serial.println();
}

```

```
int command = message[0] - '0'; // converte o primeiro caractere da mensagem em um valor inteiro
```

```
if (command == 1) {  
    openValve(); // avança o atuador  
} else if (command == 0) {  
    closeValve(); // recua o atuador  
}  
}
```

```
// FUNÇÃO ABRIR VÁLVULA
```

```
void openValve() {  
    digitalWrite(valvA, HIGH); // define o pino da válvula A como HIGH (aberto)  
}
```

```
// FUNÇÃO FECHAR VÁLVULA
```

```
void closeValve() {  
    digitalWrite(valvA, LOW); // define o pino da válvula A como LOW (fechado)  
}
```

11 APÊNDICE E – CÓDIGO ESP32 DA VÁLVULA B

```
/*
```

```
ESP_ValvB
```

PROJETO: Desenvolvimento de um sistema, utilizando protocolo MQTT, com Raspberry Pi 4 e ESP32 para controlar bancada eletropneumática através de um aplicativo mobile.

LINK: <https://github.com/MAPL-UFU/mqtt-app-controle>

ALUNO: Matheus Alves de Paula

PROF.: Prof. Dr. José Jean-Paul Zanlucchi de Souza Tavares

DATA: 05/01/2024

```
*/
```

```
// INCLUSÃO DE BIBLIOTECAS
```

```
#include <WiFi.h> // biblioteca WiFi
```

```
#include "esp_wpa2.h" // biblioteca wpa2 para conexão em redes Enterprise
```

```
#include <PubSubClient.h> // biblioteca PubSubClient para conexão MQTT
```

```
byte mac[6];
```

```
// PARÂMETROS PARA CONEXÃO WIFI COM A REDE EDUROAM
```

```
#define EAP_ANONYMOUS_IDENTITY "anonymous@ufu.br" //
```

anonymous@example.com, or you can use also nickname@example.com

```
#define EAP_IDENTITY " @ufu.br" // nickname@example.com, at some organizations  
should work nickname only without realm, but it is not recommended
```

```
#define EAP_PASSWORD "" // password for eduroam account
```

```
const char* ssid = "eduroam"; // "eduroam" SSID
```

```
// PARÂMETROS DO BROKER PARA CONEXÃO MQTT
```

```
const char* mqttServer = "10.32.80.12"; // endereço do broker a ser conectado. Nuvem:  
"ufumapl.cloud.shiftr.io"; LEM: "10.32.80.12"; Casa: "179.104.38.174"
```

```
const int mqttPort = 1883; // porta mqtt
```

```
const char* mqttUser = "localufulem"; // usuário mqtt do referido broker. Nuvem:  
"ufumapl"; LEM: "localufulem"; Casa: "localufumapl"
```

```
const char* mqttPassword = "r4spb3rryp!@L3M"; // senha mqtt do referido broker. Nuvem:
"p0dy0EfwVhUSzXlC"; LEM: "r4spb3rryp!@L3M"; Casa: "r4spb3rryp!@M4PL"
```

```
// INSTANCIANDO OBJETOS DAS CLASSES DAS BIBLIOTECAS
```

```
WiFiClient espClient; // definindo objeto espClient
```

```
PubSubClient MQTT(espClient); // cliente MQTT passando o objeto espClient
```

```
// INSTANCIANDO OBJETOS TASKHANDLE PARA PROCESSAMENTO DUAL CORE
```

```
TaskHandle_t Task1;
```

```
TaskHandle_t Task2;
```

```
// DEFINIÇÃO DO CLIENT ID
```

```
#define id_mqtt "ESP-VALV-B" // definindo identificador para o esp32. "ESP-
B0A732DA91C8" "ESP-E05A1B779924" "ESP-AC67B23BF1A0"
```

```
// DEFINIÇÃO DE PINOS
```

```
#define valvB 5 // definindo a ligação da válvula na porta 5
```

```
// DECLARAÇÃO DOS PROTÓTIPOS DAS FUNÇÕES
```

```
void conectaWiFi(); // realiza conexão com wifi
```

```
void conectaMQTT(); // realiza conexão com broker mqtt
```

```
void mantemConexoes(); // garantir as conexoes wifi e broker mqtt
ativas
```

```
void callback(char* topic, byte* message, unsigned int length); // função de retorno de chamada
MQTT
```

```
void openValve(); // função para abrir a válvula
```

```
void closeValve(); // função para fechar a válvula
```

```
void setup() {
```

```
Serial.begin(115200); // inicia a comunicação serial em 115200 bauds
```

```
delay(10);
```

```
Serial.print(F("Connecting to network: "));
```

```
Serial.println(ssid);
```

```

    WiFi.disconnect(true); // disconnect from WiFi to set
new WiFi connection
    WiFi.mode(WIFI_STA); // init wifi mode
    WiFi.begin(ssid, WPA2_AUTH_PEAP, EAP_ANONYMOUS_IDENTITY,
EAP_IDENTITY, EAP_PASSWORD); // WITHOUT CERTIFICATE - WORKING WITH
EXCEPTION ON RADIUS SERVER

while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(F("."));
    if (WiFi.status() == WL_CONNECT_FAILED || WiFi.status() ==
WL_CONNECTION_LOST) {
        conectaWiFi(); // reconecta Wi-Fi em caso de falha
    }
}

Serial.println("");
Serial.println(F("WiFi is connected!"));
Serial.println(F("IP address set: "));
Serial.println(WiFi.localIP()); // exibe LAN IP

WiFi.macAddress(mac);
Serial.print("MAC address: ");
Serial.print(mac[0], HEX);
Serial.print(":");
Serial.print(mac[1], HEX);
Serial.print(":");
Serial.print(mac[2], HEX);
Serial.print(":");
Serial.print(mac[3], HEX);
Serial.print(":");
Serial.print(mac[4], HEX);
Serial.print(":");
Serial.println(mac[5], HEX); // exibe MAC Address

```

```

// DECLARANDO OS PINOS
pinMode(valvB, OUTPUT); // define a saída para a válvula B como OUTPUT
digitalWrite(valvB, LOW); // estado inicial, a válvula B está fechada (LOW)

MQTT.setServer(mqttServer, mqttPort);
MQTT.setCallback(callback); // configura a função de retorno de chamada MQTT
conectaMQTT(); // chama a função para conexão com o broker MQTT

// cria uma tarefa que será executada na função Task1code(), com prioridade 1 e execução
no core 0
xTaskCreatePinnedToCore(
    Task1code, /* função que implementa a tarefa. */
    "Task1", /* nome da tarefa. */
    10000, /* número de palavras a serem alocadas para uso com a pilha da tarefa */
    NULL, /* parâmetro de entrada para a tarefa (pode ser NULL) */
    1, /* prioridade da tarefa (0 a N) */
    &Task1, /* referência para a tarefa (pode ser NULL) */
    0); /* núcleo que executará a tarefa (0 ou 1) */
delay(500);

// cria uma tarefa que será executada na função Task2code(), com prioridade 1 e execução no
core 1
xTaskCreatePinnedToCore(
    Task2code, /* função que implementa a tarefa. */
    "Task2", /* nome da tarefa. */
    10000, /* número de palavras a serem alocadas para uso com a pilha da tarefa */
    NULL, /* parâmetro de entrada para a tarefa (pode ser NULL) */
    1, /* prioridade da tarefa (0 a N) */
    &Task2, /* referência para a tarefa (pode ser NULL) */
    1); /* núcleo que executará a tarefa (0 ou 1) */
delay(500);
}

```

```
// FUNÇÃO EXECUTADA NO NÚCLEO 0
```

```
void Task1code(void* pvParameters) {
```

```
    Serial.print("Publish status running on core ");
```

```
    Serial.println(xPortGetCoreID());
```

```
    for (;;) {
```

```
        const char* msg_status = "ESP32 VALVULA B CONECTADO";
```

```
        MQTT.publish("/status/espValvB", msg_status); // publica o status do ESP32
```

```
        //Serial.print("Mensagem publicada no tópico: /status/espValvB");
```

```
        //Serial.print(" Mensagem: ");
```

```
        //Serial.println(msg_status);
```

```
        delay(3000);
```

```
    }
```

```
}
```

```
// FUNÇÃO EXECUTADA NO NÚCLEO 1
```

```
void Task2code(void* pvParameters) {
```

```
    Serial.print("Conexões (Wi-Fi e MQTT) running on core ");
```

```
    Serial.println(xPortGetCoreID());
```

```
    for (;;) {
```

```
        mantemConexoes(); // função para reconectar wifi ou broker em caso de queda
```

```
        MQTT.loop(); // mantém a comunicação MQTT ativa
```

```
        //delay(10);
```

```
    }
```

```
}
```

```
void loop() {
```

```
}
```

```
// FUNÇÃO DE RECONEXÃO MQTT E WIFI
```

```
void mantemConexoes() {
```

```
    if (!MQTT.connected()) {
```

```
        conectaMQTT();
```

```

}

conectaWiFi();
}

// FUNÇÃO CONEXÃO REDE WIFI
void conectaWiFi() {

    if (WiFi.status() == WL_CONNECTED) {
        return;
    }

    Serial.println("Falha na conexão com a rede WiFi.");
    Serial.print("Reconectando-se na rede: ");
    Serial.print(ssid);

    WiFi.disconnect(true); //disconnect from WiFi to set new WiFi connection
    WiFi.mode(WIFI_STA); //init wifi mode
    WiFi.begin(ssid, WPA2_AUTH_PEAP, EAP_ANONYMOUS_IDENTITY,
EAP_IDENTITY, EAP_PASSWORD);

    int counter = 0;
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
        counter++;

        if (counter >= 60) {
            ESP.restart(); // Reinicia o ESP32 após 30 segundos de tentativas de conexão
        }

        Serial.println();
        Serial.print("Conectado com sucesso na rede: ");
        Serial.println(ssid);

```

```

    }
}

// FUNÇÃO CONEXÃO MQTT
void conectaMQTT() {
    while (!MQTT.connected()) {
        Serial.print("Conectando ao Broker MQTT: ");
        Serial.println(mqttServer);

        if (MQTT.connect(id_mqtt, mqttUser, mqttPassword)) {
            Serial.println("Conexao realizada com sucesso!");

            MQTT.subscribe("/comando/valvB");           // subscreve para receber comandos

        } else {
            Serial.println("Falha na conexão com o broker MQTT.");
            Serial.println("Nova tentativa de conexão em 5s...");
            delay(5000);
        }
    }
}

// FUNÇÃO DE RETORNO DE CHAMADA MQTT
void callback(char* topic, byte* message, unsigned int length) {
    Serial.print("Mensagem recebida no tópico: ");
    Serial.print(topic);
    Serial.print(". Mensagem: ");

    for (int i = 0; i < length; i++) {
        Serial.print((char)message[i]);
    }

    Serial.println();
}

```

```
int command = message[0] - '0'; // converte o primeiro caractere da mensagem em um valor inteiro
```

```
if (command == 1) {  
    openValve(); // avança o atuador  
} else if (command == 0) {  
    closeValve(); // recua o atuador  
}  
}
```

```
// FUNÇÃO ABRIR VÁLVULA
```

```
void openValve() {  
    digitalWrite(valvB, HIGH); // define o pino da válvula B como HIGH (aberto)  
}
```

```
// FUNÇÃO FECHAR VÁLVULA
```

```
void closeValve() {  
    digitalWrite(valvB, LOW); // define o pino da válvula B como LOW (fechado)  
}
```

12 APÊNDICE F – CÓDIGO ESP32 DA VÁLVULA C

```
/*
```

```
ESP_ValvC
```

PROJETO: Desenvolvimento de um sistema, utilizando protocolo MQTT, com Raspberry Pi 4 e ESP32 para controlar bancada eletropneumática através de um aplicativo mobile.

LINK: <https://github.com/MAPL-UFU/mqtt-app-controle>

ALUNO: Matheus Alves de Paula

PROF.: Prof. Dr. José Jean-Paul Zanlucchi de Souza Tavares

DATA: 05/01/2024

```
*/
```

```
// INCLUSÃO DE BIBLIOTECAS
```

```
#include <WiFi.h> // biblioteca WiFi
```

```
#include "esp_wpa2.h" // biblioteca wpa2 para conexão em redes Enterprise
```

```
#include <PubSubClient.h> // biblioteca PubSubClient para conexão MQTT
```

```
byte mac[6];
```

```
// PARÂMETROS PARA CONEXÃO WIFI COM A REDE EDUROAM
```

```
#define EAP_ANONYMOUS_IDENTITY "anonymous@ufu.br" // anonymous@example.com, or you can use also nickname@example.com
```

```
#define EAP_IDENTITY "@ufu.br" // nickname@example.com, at some organizations should work nickname only without realm, but it is not recommended
```

```
#define EAP_PASSWORD "" // password for eduroam account
```

```
const char* ssid = "eduroam"; // "eduroam" SSID
```

```
// PARÂMETROS DO BROKER PARA CONEXÃO MQTT
```

```
const char* mqttServer = "10.32.80.12"; // endereço do broker a ser conectado. Nuvem: "ufumapl.cloud.shiftr.io" ou "broker.hivemq.com"; LEM: "10.32.80.12"
```

```
const int mqttPort = 1883; // porta mqtt
```

```
const char* mqttUser = "localufulem"; // usuário mqtt do referido broker. Nuvem: "ufumapl" ou ""; LEM: "localufulem"
```

```
const char* mqttPassword = "r4spb3rryp!@L3M"; // senha mqtt do referido broker. Nuvem:
"p0dy0EfwVhUSzXlC" ou ""; LEM: "r4spb3rryp!@L3M"
```

```
// INSTANCIANDO OBJETOS DAS CLASSES DAS BIBLIOTECAS
```

```
WiFiClient espClient;      // definindo objeto espClient
```

```
PubSubClient MQTT(espClient); // cliente MQTT passando o objeto espClient
```

```
// INSTANCIANDO OBJETOS TASKHANDLE PARA PROCESSAMENTO DUAL CORE
```

```
TaskHandle_t Task1;
```

```
TaskHandle_t Task2;
```

```
// DEFINIÇÃO DO CLIENT ID
```

```
#define id_mqtt "ESP-VALV-C" // definindo identificador para o esp32. "ESP-
B0A732DA91C8" "ESP-E05A1B779924" "ESP-AC67B23BF1A0"
```

```
// DEFINIÇÃO DE PINOS
```

```
#define valvC 5 // definindo a ligação da válvula na porta 5
```

```
// DECLARAÇÃO DOS PROTÓTIPOS DAS FUNÇÕES
```

```
void conectaWiFi();           // realiza conexão com wifi
```

```
void conectaMQTT();           // realiza conexão com broker mqtt
```

```
void mantemConexoes();        // garantir as conexoes wifi e broker mqtt
ativas
```

```
void callback(char* topic, byte* message, unsigned int length); // função de retorno de chamada
MQTT
```

```
void openValve();              // função para abrir a válvula
```

```
void closeValve();             // função para fechar a válvula
```

```
void setup() {
```

```
    Serial.begin(115200); // inicia a comunicação serial em 115200 bauds
```

```
    delay(10);
```

```
    Serial.print(F("Connecting to network: "));
```

```
    Serial.println(ssid);
```

```

    WiFi.disconnect(true); // disconnect from WiFi to set
new WiFi connection
    WiFi.mode(WIFI_STA); // init wifi mode
    WiFi.begin(ssid, WPA2_AUTH_PEAP, EAP_ANONYMOUS_IDENTITY,
EAP_IDENTITY, EAP_PASSWORD); // WITHOUT CERTIFICATE - WORKING WITH
EXCEPTION ON RADIUS SERVER

while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(F("."));
    if (WiFi.status() == WL_CONNECT_FAILED || WiFi.status() ==
WL_CONNECTION_LOST) {
        conectaWiFi(); // reconecta Wi-Fi em caso de falha
    }
}

Serial.println("");
Serial.println(F("WiFi is connected!"));
Serial.println(F("IP address set: "));
Serial.println(WiFi.localIP()); // exibe LAN IP

WiFi.macAddress(mac);
Serial.print("MAC address: ");
Serial.print(mac[0], HEX);
Serial.print(":");
Serial.print(mac[1], HEX);
Serial.print(":");
Serial.print(mac[2], HEX);
Serial.print(":");
Serial.print(mac[3], HEX);
Serial.print(":");
Serial.print(mac[4], HEX);
Serial.print(":");
Serial.println(mac[5], HEX); // exibe MAC Address

```

```

// DECLARANDO OS PINOS
pinMode(valvC, OUTPUT); // define a saída para a válvula C como OUTPUT
digitalWrite(valvC, LOW); // estado inicial, a válvula C está fechada (LOW)

MQTT.setServer(mqttServer, mqttPort);
MQTT.setCallback(callback); // configura a função de retorno de chamada MQTT
conectaMQTT(); // chama a função para conexão com o broker MQTT

// cria uma tarefa que será executada na função Task1code(), com prioridade 1 e execução
no core 0
xTaskCreatePinnedToCore(
    Task1code, /* função que implementa a tarefa. */
    "Task1", /* nome da tarefa. */
    10000, /* número de palavras a serem alocadas para uso com a pilha da tarefa */
    NULL, /* parâmetro de entrada para a tarefa (pode ser NULL) */
    1, /* prioridade da tarefa (0 a N) */
    &Task1, /* referência para a tarefa (pode ser NULL) */
    0); /* núcleo que executará a tarefa (0 ou 1) */
delay(500);

// cria uma tarefa que será executada na função Task2code(), com prioridade 1 e execução no
core 1
xTaskCreatePinnedToCore(
    Task2code, /* função que implementa a tarefa. */
    "Task2", /* nome da tarefa. */
    10000, /* número de palavras a serem alocadas para uso com a pilha da tarefa */
    NULL, /* parâmetro de entrada para a tarefa (pode ser NULL) */
    1, /* prioridade da tarefa (0 a N) */
    &Task2, /* referência para a tarefa (pode ser NULL) */
    1); /* núcleo que executará a tarefa (0 ou 1) */
delay(500);
}

```

```
// FUNÇÃO EXECUTADA NO NÚCLEO 0
```

```
void Task1code(void* pvParameters) {
```

```
    Serial.print("Publish status running on core ");
```

```
    Serial.println(xPortGetCoreID());
```

```
    for (;;) {
```

```
        const char* msg_status = "ESP32 VALVULA C CONECTADO";
```

```
        MQTT.publish("/status/espValvC", msg_status); // publica o status do ESP32
```

```
        Serial.print("Mensagem publicada no tópico: /status/espValvC");
```

```
        Serial.print(" Mensagem: ");
```

```
        Serial.println(msg_status);
```

```
        delay(3000);
```

```
    }
```

```
}
```

```
// FUNÇÃO EXECUTADA NO NÚCLEO 1
```

```
void Task2code(void* pvParameters) {
```

```
    Serial.print("Conexões (Wi-Fi e MQTT) running on core ");
```

```
    Serial.println(xPortGetCoreID());
```

```
    for (;;) {
```

```
        mantemConexoes(); // função para reconectar wifi ou broker em caso de queda
```

```
        MQTT.loop(); // mantém a comunicação MQTT ativa
```

```
        //delay(10);
```

```
    }
```

```
}
```

```
void loop() {
```

```
}
```

```
// FUNÇÃO DE RECONEXÃO MQTT E WIFI
```

```
void mantemConexoes() {
```

```
    if (!MQTT.connected()) {
```

```
        conectaMQTT();
```

```

}

conectaWiFi();
}

// FUNÇÃO CONEXÃO REDE WIFI
void conectaWiFi() {

    if (WiFi.status() == WL_CONNECTED) {
        return;
    }

    Serial.println("Falha na conexão com a rede WiFi.");
    Serial.print("Reconectando-se na rede: ");
    Serial.print(ssid);

    WiFi.disconnect(true); //disconnect from WiFi to set new WiFi connection
    WiFi.mode(WIFI_STA); //init wifi mode
    WiFi.begin(ssid, WPA2_AUTH_PEAP, EAP_ANONYMOUS_IDENTITY,
EAP_IDENTITY, EAP_PASSWORD);

    int counter = 0;
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
        counter++;

        if (counter >= 60) {
            ESP.restart(); // Reinicia o ESP32 após 30 segundos de tentativas de conexão
        }

        Serial.println();
        Serial.print("Conectado com sucesso na rede: ");
        Serial.println(ssid);

```

```

    }
}

// FUNÇÃO CONEXÃO MQTT
void conectaMQTT() {
    while (!MQTT.connected()) {
        Serial.print("Conectando ao Broker MQTT: ");
        Serial.println(mqttServer);

        if (MQTT.connect(id_mqtt, mqttUser, mqttPassword)) {
            Serial.println("Conexao realizada com sucesso!");

            MQTT.subscribe("/comando/valvC");          // subscreve para receber comandos

        } else {
            Serial.println("Falha na conexão com o broker MQTT.");
            Serial.println("Nova tentativa de conexão em 5s...");
            delay(5000);
        }
    }
}

// FUNÇÃO DE RETORNO DE CHAMADA MQTT
void callback(char* topic, byte* message, unsigned int length) {
    Serial.print("Mensagem recebida no tópico: ");
    Serial.print(topic);
    Serial.print(". Mensagem: ");

    for (int i = 0; i < length; i++) {
        Serial.print((char)message[i]);
    }

    Serial.println();
}

```

```
int command = message[0] - '0'; // converte o primeiro caractere da mensagem em um valor inteiro
```

```
if (command == 1) {  
    openValve(); // avança o atuador  
} else if (command == 0) {  
    closeValve(); // recua o atuador  
}  
}
```

```
// FUNÇÃO ABRIR VÁLVULA
```

```
void openValve() {  
    digitalWrite(valvC, HIGH); // define o pino da válvula C como HIGH (aberto)  
}
```

```
// FUNÇÃO FECHAR VÁLVULA
```

```
void closeValve() {  
    digitalWrite(valvC, LOW); // define o pino da válvula C como LOW (fechado)  
}
```

13 APÊNDICE G – CÓDIGO ESP32 DOS SENSORES

```

/*
    ESP_SensoresFdC

    PROJETO: Desenvolvimento de um sistema, utilizando protocolo MQTT, com Raspberry
    Pi 4 e ESP32 para controlar bancada eletropneumática através de um aplicativo mobile.

    LINK:   https://github.com/MAPL-UFU/mqtt-app-controle

    ALUNO:   Matheus Alves de Paula

    PROF.:   Prof. Dr. José Jean-Paul Zanlucchi de Souza Tavares

    DATA:   05/01/2024
*/

// INCLUSÃO DE BIBLIOTECAS
#include <WiFi.h>           // biblioteca WiFi
#include "esp_wpa2.h"       // biblioteca wpa2 para conexão em redes Enterprise
#include <PubSubClient.h>   // biblioteca PubSubClient para conexão MQTT

byte mac[6];

// PARÂMETROS PARA CONEXÃO WIFI COM A REDE EDUROAM
#define EAP_ANONYMOUS_IDENTITY "anonymous@ufu.br" //
anonymous@example.com, or you can use also nickname@example.com
#define EAP_IDENTITY "@ufu.br" // nickname@example.com, at some organizations
should work nickname only without realm, but it is not recommended
#define EAP_PASSWORD "" // password for eduroam account

const char* ssid = "eduroam"; // "eduroam" SSID

// PARÂMETROS DO BROKER PARA CONEXÃO MQTT
const char* mqttServer = "10.32.80.12"; // endereço do broker a ser conectado. Nuvem:
"ufumapl.cloud.shiftr.io"; LEM: "10.32.80.12"; Casa: "192.168.100.122"
const int mqttPort = 1883; // porta mqtt
const char* mqttUser = "localufulem"; // usuário mqtt do referido broker. Nuvem:
"ufumapl"; LEM: "localufulem"; Casa: "localufumapl"

```

```
const char* mqttPassword = "r4spb3rryp!@L3M"; // senha mqtt do referido broker. Nuvem:
"p0dy0EfwVhUSzXlC"; LEM: "r4spb3rryp!@L3M"; Casa: "r4spb3rryp!@M4PL"
```

```
// INSTANCIANDO OBJETOS DAS CLASSES DAS BIBLIOTECAS
```

```
WiFiClient espClient; // definindo objeto espClient
```

```
PubSubClient MQTT(espClient); // cliente MQTT passando o objeto espClient
```

```
// INSTANCIANDO OBJETOS TASKHANDLE PARA PROCESSAMENTO DUAL CORE
```

```
TaskHandle_t Task1;
```

```
TaskHandle_t Task2;
```

```
// DEFINIÇÃO DO CLIENT ID
```

```
#define id_mqtt "ESP-SENSORES" // definindo identificador para o esp32. "ESP-
E05A1B5FF59C" "ESP-E05A1B779924" "ESP-AC67B23BF1A0"
```

```
// DEFINIÇÃO DE PINOS
```

```
#define A0 2 // definindo a ligação do sensor fim de curso na porta 2
```

```
#define A1 4 // definindo a ligação do sensor fim de curso na porta 4
```

```
#define B0 16 // definindo a ligação do sensor fim de curso na porta 5
```

```
#define B1 17 // definindo a ligação do sensor fim de curso na porta 19
```

```
#define C0 18 // definindo a ligação do sensor fim de curso na porta 12
```

```
#define C1 19 // definindo a ligação do sensor fim de curso na porta 14
```

```
// DECLARAÇÃO DOS PROTÓTIPOS DAS FUNÇÕES
```

```
void conectaWiFi(); // realiza conexão com wifi
```

```
void conectaMQTT(); // realiza conexão com broker mqtt
```

```
void mantemConexoes(); // garantir as conexoes wifi e broker mqtt ativas
```

```
void publishSensorA0A1State(int a0Value, int a1Value); // realiza publicações nos tópicos
referente aos sensores do atuador A
```

```
void publishSensorB0B1State(int b0Value, int b1Value); // realiza publicações nos tópicos
referente aos sensores do atuador B
```

void publishSensorC0C1State(int c0Value, int c1Value); // realiza publicações nos tópicos referente aos sensores do atuador C

void setup() {

Serial.begin(115200); // inicia a comunicação serial em 115200 bauds

delay(10);

Serial.print(F("Conectando à rede: "));

Serial.println(ssid);

Serial.println("\n");

WiFi.disconnect(true); // disconnect from WiFi to set

new WiFi connection

WiFi.mode(WIFI_STA); // init wifi mode

WiFi.begin(ssid, WPA2_AUTH_PEAP, EAP_ANONYMOUS_IDENTITY, EAP_IDENTITY, EAP_PASSWORD); // WITHOUT CERTIFICATE - WORKING WITH EXCEPTION ON RADIUS SERVER

while (WiFi.status() != WL_CONNECTED) {

delay(500);

Serial.print(F("."));

if (WiFi.status() == WL_CONNECT_FAILED || WiFi.status() == WL_CONNECTION_LOST) {

conectaWiFi(); // Reconecta Wi-Fi em caso de falha

}

}

Serial.println("\n");

Serial.println(F("Rede WiFi está conectada!"));

Serial.println(F("Endereço IP recebido: "));

Serial.println(WiFi.localIP()); // print LAN IP

WiFi.macAddress(mac);

Serial.print("MAC address: ");

Serial.print(mac[0], HEX);

```

Serial.print(":");
Serial.print(mac[1], HEX);
Serial.print(":");
Serial.print(mac[2], HEX);
Serial.print(":");
Serial.print(mac[3], HEX);
Serial.print(":");
Serial.print(mac[4], HEX);
Serial.print(":");
Serial.println(mac[5], HEX); // print MAC Address

```

```

// DECLARANDO PINOS DE INPUT

```

```

pinMode(A0, INPUT_PULLUP); // inicializando a variável do sensor A0 como entrada com
resistor pull-up interno habilitado

```

```

pinMode(A1, INPUT_PULLUP); // inicializando a variável do sensor A1 como entrada com
resistor pull-up interno habilitado

```

```

pinMode(B0, INPUT_PULLUP); // inicializando a variável do sensor B0 como entrada com
resistor pull-up interno habilitado

```

```

pinMode(B1, INPUT_PULLUP); // inicializando a variável do sensor B1 como entrada com
resistor pull-up interno habilitado

```

```

pinMode(C0, INPUT_PULLUP); // inicializando a variável do sensor C0 como entrada com
resistor pull-up interno habilitado

```

```

pinMode(C1, INPUT_PULLUP); // inicializando a variável do sensor C1 como entrada com
resistor pull-up interno habilitado

```

```

MQTT.setServer(mqttServer, mqttPort);
conectaMQTT();

```

```

// cria uma tarefa que será executada na função Task1code(), com prioridade 1 e execução
no core 0

```

```

xTaskCreatePinnedToCore(
    Task1code, /* função que implementa a tarefa. */

```

```

"Task1", /* nome da tarefa. */
10000, /* número de palavras a serem alocadas para uso com a pilha da tarefa */
NULL, /* parâmetro de entrada para a tarefa (pode ser NULL) */
0, /* prioridade da tarefa (0 a N) */
&Task1, /* referência para a tarefa (pode ser NULL) */
0); /* núcleo que executará a tarefa (0 ou 1) */
delay(500);

```

// cria uma tarefa que será executada na função Task2code(), com prioridade 1 e execução no core 1

```

xTaskCreatePinnedToCore(
    Task2code, /* função que implementa a tarefa. */
    "Task2", /* nome da tarefa. */
    10000, /* número de palavras a serem alocadas para uso com a pilha da tarefa */
    NULL, /* parâmetro de entrada para a tarefa (pode ser NULL) */
    1, /* prioridade da tarefa (0 a N) */
    &Task2, /* referência para a tarefa (pode ser NULL) */
    1); /* núcleo que executará a tarefa (0 ou 1) */
delay(500);
}

```

// FUNÇÃO EXECUTADA NO NÚCLEO 0

```

void Task1code(void* pvParameters) {
    Serial.print("Publish status running on core ");
    Serial.println(xPortGetCoreID());

    for (;;) {
        const char* msg_status = "ESP32 SENSORES CONECTADO";
        MQTT.publish("/status/espSensores", msg_status); // publica o status do ESP32
        //Serial.print("Mensagem publicada no tópico: /status/espSensores");
        //Serial.print(" Mensagem: ");
        //Serial.println(msg_status);
        delay(3000);
    }
}

```

```
}
```

```
// FUNÇÃO EXECUTADA NO NÚCLEO 1
```

```
void Task2code(void* pvParameters) {
```

```
    Serial.print("Conexões (Wi-Fi e MQTT) running on core ");
```

```
    Serial.println(xPortGetCoreID());
```

```
    for (;;) {
```

```
        mantemConexoes(); // função para reconectar wifi ou broker em caso de queda
```

```
        MQTT.loop();      // mantém a comunicação MQTT ativa
```

```
        // variavel auxiliar para tratar dados do sensores
```

```
        int a0Value = digitalRead(A0); // Lê o estado do sensor fim de curso a0
```

```
        int a1Value = digitalRead(A1); // Lê o estado do sensor fim de curso a1
```

```
        int b0Value = digitalRead(B0); // Lê o estado do sensor fim de curso b0
```

```
        int b1Value = digitalRead(B1); // Lê o estado do sensor fim de curso b1
```

```
        int c0Value = digitalRead(C0); // Lê o estado do sensor fim de curso c0
```

```
        int c1Value = digitalRead(C1); // Lê o estado do sensor fim de curso c1
```

```
        publishSensorA0A1State(a0Value, a1Value); // Publica o estado do sensor
```

```
        publishSensorB0B1State(b0Value, b1Value); // Publica o estado do sensor
```

```
        publishSensorC0C1State(c0Value, c1Value); // Publica o estado do sensor
```

```
        delay(1000); // intervalo de 1 segundo entre as leituras
```

```
    }
```

```
}
```

```
void loop() {
```

```
}
```

```
// FUNÇÃO DE RECONEXÃO MQTT E WIFI
```

```
void mantemConexoes() {
```

```

if (!MQTT.connected()) {
    conectaMQTT();
}

conectaWiFi();
}

// FUNÇÃO CONEXÃO REDE WIFI
void conectaWiFi() {

    if (WiFi.status() == WL_CONNECTED) {
        return;
    }

    Serial.println("Falha na conexão com a rede WiFi.");
    Serial.print("Reconectando-se na rede: ");
    Serial.print(ssid);

    WiFi.disconnect(true); //disconnect from WiFi to set new WiFi connection
    WiFi.mode(WIFI_STA); //init wifi mode
    WiFi.begin(ssid, WPA2_AUTH_PEAP, EAP_ANONYMOUS_IDENTITY,
EAP_IDENTITY, EAP_PASSWORD);

    int counter = 0;
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
        counter++;

        if (counter >= 60) {
            ESP.restart(); // Reinicia o ESP32 após 30 segundos de tentativas de conexão
        }

        Serial.println();

```

```

    Serial.print("Conectado com sucesso na rede: ");
    Serial.println(ssid);
}
}

// FUNÇÃO CONEXÃO MQTT
void conectaMQTT() {
    while (!MQTT.connected()) {
        Serial.print("Conectando ao Broker MQTT: ");
        Serial.println(mqttServer);

        if (MQTT.connect(id_mqtt, mqttUser, mqttPassword)) {
            Serial.println("Conexao realizada com sucesso!");
        } else {
            Serial.println("Falha na conexão com o broker MQTT.");
            Serial.println("Nova tentativa de conexão em 5s...");
            delay(5000);
        }
    }
}

// FUNÇÃO SENSORES ATUADOR A
void publishSensorA0A1State(int a0Value, int a1Value) {
    if (a0Value == LOW && a1Value == HIGH) {
        MQTT.publish("/estado/atuadorA", "Atuador A Recuado");
        Serial.println("Mensagem publicada no tópico: /estado/atuadorA Mensagem: Atuador A Recuado");

    } else if (a0Value == HIGH && a1Value == LOW) {
        MQTT.publish("/estado/atuadorA", "Atuador A Avançado");
        Serial.println("Mensagem publicada no tópico: /estado/atuadorA Mensagem: Atuador A Avançado");

    } else if (a0Value == LOW && a1Value == LOW) {

```

```

MQTT.publish("/estado/atuadorA", "Sensor(es) do Atuador A com mau contato!");
Serial.println("Mensagem publicada no t3pico: /estado/atuadorA Mensagem: Sensor(es) do
Atuador A com mau contato!");

} else {
    //Serial.println("\n ... \n");
}
}

// FUNÇÃO SENSORES ATUADOR B
void publishSensorB0B1State(int b0Value, int b1Value) {
    if (b0Value == LOW && b1Value == HIGH) {
        MQTT.publish("/estado/atuadorB", "Atuador B Recuado");
        Serial.println("Mensagem publicada no t3pico: /estado/atuadorB Mensagem: Atuador B
Recuado");

    } else if (b0Value == HIGH && b1Value == LOW) {
        MQTT.publish("/estado/atuadorB", "Atuador B Avançado");
        Serial.println("Mensagem publicada no t3pico: /estado/atuadorB Mensagem: Atuador B
Avançado");

    } else if (b0Value == LOW && b1Value == LOW) {
        MQTT.publish("/estado/atuadorB", "Sensor(es) do atuador B com mau contato!");
        Serial.println("Mensagem publicada no t3pico: /estado/atuadorB Mensagem: Sensor(es) do
Atuador B com mau contato!");

    } else {
        //Serial.println("\n ... \n");
    }
}

// FUNÇÃO SENSORES ATUADOR C
void publishSensorC0C1State(int c0Value, int c1Value) {
    if (c0Value == LOW && c1Value == HIGH) {

```

```
Serial.println("Mensagem publicada no tópico: /estado/atuadorC Mensagem: Atuador C
Recuado");
Serial.println("\n Atuador C Recuado \n");

} else if (c0Value == HIGH && c1Value == LOW) {
MQTT.publish("/estado/atuadorC", "Atuador C Avançado");
Serial.println("Mensagem publicada no tópico: /estado/atuadorC Mensagem: Atuador C
Avançado");

} else if (c0Value == LOW && c1Value == LOW) {
MQTT.publish("/estado/atuadorC", "Sensor(es) do atuador C com mau contato!");
Serial.println("Mensagem publicada no tópico: /estado/atuadorC Mensagem: Sensor(es) do
Atuador C com mau contato!");

} else {
//Serial.println("\n ... \n");
}
}
```

14 APÊNDICE H – CÓDIGO XML DA PRIMEIRA TELA

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/white"
    android:orientation="vertical"
    android:padding="10dp"
    tools:context=".MainActivity">

    <!-- Título MQTT SETUP -->
    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_weight="0.1" >

        <TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:fontFamily="@font/bungee_shade"
            android:gravity="center"
            android:text="@string/mqtt_setup"
            android:textColor="@color/roxo"
            android:textSize="30sp"
            android:textStyle="bold"/>

    </LinearLayout>

    <!-- BROKERS -->
    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"

```

```

android:layout_gravity="center"
android:orientation="vertical"
android:layout_weight="0.15">

```

```

<!-- Título BROKER -->

```

```

<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:fontFamily="@font/bungee"
    android:gravity="center"
    android:text="@string/broker"
    android:textColor="@color/roxo"
    android:textSize="20sp" />

```

```

<RadioGroup
    android:id="@+id/radioGroupBroker"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center">

```

```

<RadioButton
    android:id="@+id/localLEM_rb"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:fontFamily="sans-serif"
    android:text="@string/local_lem"
    android:textColor="@color/roxo"
    android:textSize="18sp"
    android:textStyle="bold" />

```

```

<RadioButton
    android:id="@+id/nuvemShiftr_rb"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"

```

```

        android:text="@string/cloud_shiftr_io"
        android:textColor="@color/roxo"
        android:textSize="18sp"
        android:textStyle="bold" />

```

```

<RadioButton
    android:id="@+id/nuvemHiveMQ_rb"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/cloud_hivemq"
    android:textColor="@color/roxo"
    android:textSize="18sp"
    android:textStyle="bold" />

```

```

</RadioGroup>

```

```

</LinearLayout>

```

```

<!-- ATUADORES -->

```

```

<LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:orientation="vertical"
    android:layout_weight="0.1" >

```

```

<!-- Título ATUADORES -->

```

```

<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:fontFamily="@font/bungee"
    android:gravity="center"
    android:text="@string/atuadores"
    android:textColor="@color/roxo"
    android:textSize="20sp" />

```

```

<RadioGroup
    android:id="@+id/radioGroupAtuad"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:orientation="horizontal">

    <RadioButton
        android:id="@+id/doesAtuad_rb"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginEnd="10dp"
        android:text="@string/does"
        android:textColor="@color/roxo"
        android:textSize="18sp"
        android:textStyle="bold" />

    <RadioButton
        android:id="@+id/tresAtuad_rb"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/tres"
        android:textColor="@color/roxo"
        android:textSize="18sp"
        android:textStyle="bold" />
</RadioGroup>

</LinearLayout>
<!-- MODOS -->
<LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"

```

```

android:orientation="vertical"
android:layout_weight="0.15" >

```

```

<!-- Título MODO -->

```

```

<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:fontFamily="@font/bungee"
    android:gravity="center"
    android:text="@string/modo"
    android:textColor="@color/roxo"
    android:textSize="20sp" />

```

```

<RadioGroup
    android:id="@+id/radiGroupModo"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:orientation="vertical">

```

```

<RadioButton
    android:id="@+id/controle_rb"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/controle"
    android:textColor="@color/roxo"
    android:textSize="18sp"
    android:textStyle="bold" />

```

```

<RadioButton
    android:id="@+id/comando_rb"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/comando"

```

```

        android:textColor="@color/roxo"
        android:textSize="18sp"
        android:textStyle="bold" />

```

```

<RadioButton
    android:id="@+id/dashboard_rb"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/dashboard"
    android:textColor="@color/roxo"
    android:textSize="18sp"
    android:textStyle="bold" />

```

```

</RadioGroup>

```

```

</LinearLayout>

```

```

<!-- BOTÕES -->

```

```

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:orientation="horizontal"
    android:layout_weight="0.5" >

```

```

<androidx.appcompat.widget.AppCompatButton
    android:id="@+id/exit_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="@drawable/background_button"
    android:drawableStart="@drawable/exit_icon"
    android:text="@string/sair"
    android:padding="10dp"
    android:textColor="@color/white" />

```

```

<Space

```

```
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        tools:ignore="NestedWeights" />

<androidx.appcompat.widget.AppCompatButton
    android:id="@+id/confirm_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="@drawable/background_button"
    android:drawableEnd="@drawable/next_icon"
    android:text="@string/avancar"
    android:textColor="@color/white"
    android:padding="10dp" />
</LinearLayout>
</LinearLayout>
```

15 APÊNDICE I – CÓDIGO JAVA DA PRIMEIRA TELA

```

package com.somsakelect.android.mqttcontrolapp;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.ViewGroup;
import android.view.WindowManager;
import android.widget.RadioButton;
import android.widget.TextView;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
    private RadioButton localLEM_rb, nuvemShiftr_rb, nuvemHiveMQ_rb, doisAtuad_rb,
    tresAtuad_rb, controle_rb, comando_rb, dashboard_rb;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        try {
            Thread.sleep(1000); // Delay de um segundo para o Splash manter ativo na tela
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }

        setTheme(R.style.Theme_MyMQTT); // App inicia com o tema Splash. Aqui altera para
o tema principal
        setContentView(R.layout.activity_main);
    }
}

```

```
getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,WindowManager.LayoutParams.FLAG_FULLSCREEN); // Iniciar no modo tela cheia
```

```

localLEM_rb = findViewById(R.id.localLEM_rb);
nuvemShiftr_rb = findViewById(R.id.nuvemShiftr_rb);
nuvemHiveMQ_rb = findViewById(R.id.nuvemHiveMQ_rb);
doisAtuad_rb = findViewById(R.id.doisAtuad_rb);
tresAtuad_rb = findViewById(R.id.tresAtuad_rb);
controle_rb = findViewById(R.id.controle_rb);
comando_rb = findViewById(R.id.comando_rb);
dashboard_rb = findViewById(R.id.dashboard_rb);

// Botão Sair
findViewById(R.id.exit_button).setOnClickListener(v -> finishAffinity());

//Botão Confirmar
findViewById(R.id.confirm_button).setOnClickListener(v -> {
    if (localLEM_rb.isChecked() && doisAtuad_rb.isChecked() &&
controle_rb.isChecked()) {
        // Crie um Intent para iniciar DoisAtuad
        Intent localLEM_DoisAtuad = new Intent(MainActivity.this, DoisAtuad.class);
        localLEM_DoisAtuad.putExtra("MQTT_BROKER", "10.32.80.12");
        localLEM_DoisAtuad.putExtra("MQTT_USUARIO", "localufulem");
        localLEM_DoisAtuad.putExtra("MQTT_SENHA", "r4spb3rryp!@L3M");
        // Inicie a DoisAtuad com o Intent
        startActivity(localLEM_DoisAtuad);
    } else if (localLEM_rb.isChecked() && doisAtuad_rb.isChecked() &&
comando_rb.isChecked()) {
        // Crie um Intent para iniciar ComandoDoisAtuad
        Intent localLEM_ComandoDoisAtuad = new Intent(MainActivity.this,
ComandoDoisAtuad.class);
        localLEM_ComandoDoisAtuad.putExtra("MQTT_BROKER", "10.32.80.12");
        localLEM_ComandoDoisAtuad.putExtra("MQTT_USUARIO", "localufulem");
    }
});

```

```

        localLEM_ComandoDoisAtuad.putExtra("MQTT_SENHA", "r4spb3rryp!@L3M");
        // Inicie a ComandoDoisAtuad com o Intent
        startActivity(localLEM_ComandoDoisAtuad);
    } else if (localLEM_rb.isChecked() && doisAtuad_rb.isChecked() &&
dashboard_rb.isChecked()) {
        // Crie um Intent para iniciar DashDoisAtuad
        Intent localLEM_DashDoisAtuad = new Intent(MainActivity.this,
DashDoisAtuad.class);
        localLEM_DashDoisAtuad.putExtra("MQTT_BROKER", "10.32.80.12");
        localLEM_DashDoisAtuad.putExtra("MQTT_USUARIO", "localufulem");
        localLEM_DashDoisAtuad.putExtra("MQTT_SENHA", "r4spb3rryp!@L3M");
        // Inicie a DashDoisAtuad com o Intent
        startActivity(localLEM_DashDoisAtuad);
    } else if (nuvemShiftr_rb.isChecked() && doisAtuad_rb.isChecked() &&
controle_rb.isChecked()) {
        // Crie um Intent para iniciar DoisAtuad
        Intent nuvemShiftr_DoisAtuad = new Intent(MainActivity.this, DoisAtuad.class);
        nuvemShiftr_DoisAtuad.putExtra("MQTT_BROKER", "ufumapl.cloud.shiftr.io");
        nuvemShiftr_DoisAtuad.putExtra("MQTT_USUARIO", "ufumapl");
        nuvemShiftr_DoisAtuad.putExtra("MQTT_SENHA", "p0dy0EfwVhUSzXlC");
        // Inicie a DoisAtuad com o Intent
        startActivity(nuvemShiftr_DoisAtuad);
    } else if (nuvemShiftr_rb.isChecked() && doisAtuad_rb.isChecked() &&
comando_rb.isChecked()) {
        // Crie um Intent para iniciar ComandoDoisAtuad
        Intent nuvemShiftr_ComandoDoisAtuad = new Intent(MainActivity.this,
ComandoDoisAtuad.class);
        nuvemShiftr_ComandoDoisAtuad.putExtra("MQTT_BROKER",
"ufumapl.cloud.shiftr.io");
        nuvemShiftr_ComandoDoisAtuad.putExtra("MQTT_USUARIO", "ufumapl");
        nuvemShiftr_ComandoDoisAtuad.putExtra("MQTT_SENHA",
"p0dy0EfwVhUSzXlC");
        // Inicie a ComandoDoisAtuad com o Intent
        startActivity(nuvemShiftr_ComandoDoisAtuad);
    }
}

```

```

        } else if (nuvemShiftr_rb.isChecked() && doisAtuad_rb.isChecked() &&
dashboard_rb.isChecked()) {
            // Crie um Intent para iniciar DashDoisAtuad
            Intent    nuvemShiftr_DashDoisAtuad    =    new    Intent(MainActivity.this,
DashDoisAtuad.class);
            nuvemShiftr_DashDoisAtuad.putExtra("MQTT_BROKER",
"ufumapl.cloud.shiftr.io");
            nuvemShiftr_DashDoisAtuad.putExtra("MQTT_USUARIO", "ufumapl");
            nuvemShiftr_DashDoisAtuad.putExtra("MQTT_SENHA",
"p0dy0EfwVhUSzXlC");
            // Inicie a DashDoisAtuad com o Intent
            startActivity(nuvemShiftr_DashDoisAtuad);
        } else if (nuvemHiveMQ_rb.isChecked() && doisAtuad_rb.isChecked() &&
controle_rb.isChecked()) {
            // Crie um Intent para iniciar DoisAtuad
            Intent nuvemHiveMQ_DoisAtuad = new Intent(MainActivity.this, DoisAtuad.class);
            nuvemHiveMQ_DoisAtuad.putExtra("MQTT_BROKER", "broker.hivemq.com");
            nuvemHiveMQ_DoisAtuad.putExtra("MQTT_USUARIO", "");
            nuvemHiveMQ_DoisAtuad.putExtra("MQTT_SENHA", "");
            // Inicie a DoisAtuad com o Intent
            startActivity(nuvemHiveMQ_DoisAtuad);
        } else if (nuvemHiveMQ_rb.isChecked() && doisAtuad_rb.isChecked() &&
comando_rb.isChecked()) {
            // Crie um Intent para iniciar ComandoDoisAtuad
            Intent    nuvemHiveMQ_ComandoDoisAtuad    =    new    Intent(MainActivity.this,
ComandoDoisAtuad.class);
            nuvemHiveMQ_ComandoDoisAtuad.putExtra("MQTT_BROKER",
"broker.hivemq.com");
            nuvemHiveMQ_ComandoDoisAtuad.putExtra("MQTT_USUARIO", "");
            nuvemHiveMQ_ComandoDoisAtuad.putExtra("MQTT_SENHA", "");
            // Inicie a ComandoDoisAtuad com o Intent
            startActivity(nuvemHiveMQ_ComandoDoisAtuad);
        } else if (nuvemHiveMQ_rb.isChecked() && doisAtuad_rb.isChecked() &&
dashboard_rb.isChecked()) {

```

```

        // Crie um Intent para iniciar DashDoisAtuad
        Intent    nuvemHiveMQ_DashDoisAtuad    =    new    Intent(MainActivity.this,
DashDoisAtuad.class);
        nuvemHiveMQ_DashDoisAtuad.putExtra("MQTT_BROKER",
"broker.hivemq.com");
        nuvemHiveMQ_DashDoisAtuad.putExtra("MQTT_USUARIO", "");
        nuvemHiveMQ_DashDoisAtuad.putExtra("MQTT_SENHA", "");
        // Inicie a DashDoisAtuad com o Intent
        startActivity(nuvemHiveMQ_DashDoisAtuad);
    }   else   if   (localLEM_rb.isChecked()   &&   tresAtuad_rb.isChecked()   &&
controle_rb.isChecked()) {
        // Crie um Intent para iniciar TresAtuad
        Intent localLEM_TresAtuad = new Intent(MainActivity.this, TresAtuad.class);
        localLEM_TresAtuad.putExtra("MQTT_BROKER", "10.32.80.12");
        localLEM_TresAtuad.putExtra("MQTT_USUARIO", "localufulem");
        localLEM_TresAtuad.putExtra("MQTT_SENHA", "r4spb3rryp!@L3M");
        // Inicie a TresAtuad com o Intent
        startActivity(localLEM_TresAtuad);
    }   else   if   (localLEM_rb.isChecked()   &&   tresAtuad_rb.isChecked()   &&
comando_rb.isChecked()) {
        // Crie um Intent para iniciar ComandoTresAtuad
        Intent    localLEM_ComandoTresAtuad    =    new    Intent(MainActivity.this,
ComandoTresAtuad.class);
        localLEM_ComandoTresAtuad.putExtra("MQTT_BROKER", "10.32.80.12");
        localLEM_ComandoTresAtuad.putExtra("MQTT_USUARIO", "localufulem");
        localLEM_ComandoTresAtuad.putExtra("MQTT_SENHA", "r4spb3rryp!@L3M");
        // Inicie a ComandoTresAtuad com o Intent
        startActivity(localLEM_ComandoTresAtuad);
    }   else   if   (localLEM_rb.isChecked()   &&   tresAtuad_rb.isChecked()   &&
dashboard_rb.isChecked()) {
        // Crie um Intent para iniciar DashTresAtuad
        Intent    localLEM_DashTresAtuad    =    new    Intent(MainActivity.this,
DashTresAtuad.class);
        localLEM_DashTresAtuad.putExtra("MQTT_BROKER", "10.32.80.12");

```

```

localLEM_DashTresAtuad.putExtra("MQTT_USUARIO", "localufulem");
localLEM_DashTresAtuad.putExtra("MQTT_SENHA", "r4spb3rryp!@L3M");
// Inicie a DashTresAtuad com o Intent
startActivity(localLEM_DashTresAtuad);
} else if (nuvemShiftr_rb.isChecked() && tresAtuad_rb.isChecked() &&
controle_rb.isChecked()) {
    // Crie um Intent para iniciar TresAtuad
    Intent nuvemShiftr_TresAtuad = new Intent(MainActivity.this, TresAtuad.class);
    nuvemShiftr_TresAtuad.putExtra("MQTT_BROKER", "ufumapl.cloud.shiftr.io");
    nuvemShiftr_TresAtuad.putExtra("MQTT_USUARIO", "ufumapl");
    nuvemShiftr_TresAtuad.putExtra("MQTT_SENHA", "p0dy0EfwVhUSzXlC");
    // Inicie a TresAtuad com o Intent
    startActivity(nuvemShiftr_TresAtuad);
} else if (nuvemShiftr_rb.isChecked() && tresAtuad_rb.isChecked() &&
comando_rb.isChecked()) {
    // Crie um Intent para iniciar ComandoTresAtuad
    Intent nuvemShiftr_ComandoTresAtuad = new Intent(MainActivity.this,
ComandoTresAtuad.class);
    nuvemShiftr_ComandoTresAtuad.putExtra("MQTT_BROKER",
"ufumapl.cloud.shiftr.io");
    nuvemShiftr_ComandoTresAtuad.putExtra("MQTT_USUARIO", "ufumapl");
    nuvemShiftr_ComandoTresAtuad.putExtra("MQTT_SENHA",
"p0dy0EfwVhUSzXlC");
    // Inicie a ComandoTresAtuad com o Intent
    startActivity(nuvemShiftr_ComandoTresAtuad);
} else if (nuvemShiftr_rb.isChecked() && tresAtuad_rb.isChecked() &&
dashboard_rb.isChecked()) {
    // Crie um Intent para iniciar DashTresAtuad
    Intent nuvemShiftr_DashTresAtuad = new Intent(MainActivity.this,
DashTresAtuad.class);
    nuvemShiftr_DashTresAtuad.putExtra("MQTT_BROKER",
"ufumapl.cloud.shiftr.io");
    nuvemShiftr_DashTresAtuad.putExtra("MQTT_USUARIO", "ufumapl");

```

```

        nuvemShiftr_DashTresAtuad.putExtra("MQTT_SENHA",
"p0dy0EfwVhUSzXlC");

        // Inicie a DashTresAtuad com o Intent
        startActivity(nuvemShiftr_DashTresAtuad);

    } else if (nuvemHiveMQ_rb.isChecked() && tresAtuad_rb.isChecked() &&
controle_rb.isChecked()) {
        // Crie um Intent para iniciar TresAtuad
        Intent nuvemHiveMQ_TresAtuad = new Intent(MainActivity.this, TresAtuad.class);
        nuvemHiveMQ_TresAtuad.putExtra("MQTT_BROKER", "broker.hivemq.com");
        nuvemHiveMQ_TresAtuad.putExtra("MQTT_USUARIO", "");
        nuvemHiveMQ_TresAtuad.putExtra("MQTT_SENHA", "");
        // Inicie a TresAtuad com o Intent
        startActivity(nuvemHiveMQ_TresAtuad);

    } else if (nuvemHiveMQ_rb.isChecked() && tresAtuad_rb.isChecked() &&
comando_rb.isChecked()) {
        // Crie um Intent para iniciar ComandoTresAtuad
        Intent nuvemHiveMQ_ComandoTresAtuad = new Intent(MainActivity.this,
ComandoTresAtuad.class);
        nuvemHiveMQ_ComandoTresAtuad.putExtra("MQTT_BROKER",
"broker.hivemq.com");
        nuvemHiveMQ_ComandoTresAtuad.putExtra("MQTT_USUARIO", "");
        nuvemHiveMQ_ComandoTresAtuad.putExtra("MQTT_SENHA", "");
        // Inicie a ComandoTresAtuad com o Intent
        startActivity(nuvemHiveMQ_ComandoTresAtuad);

    } else if (nuvemHiveMQ_rb.isChecked() && tresAtuad_rb.isChecked() &&
dashboard_rb.isChecked()) {
        // Crie um Intent para iniciar DashTresAtuad
        Intent nuvemHiveMQ_DashTresAtuad = new Intent(MainActivity.this,
DashTresAtuad.class);
        nuvemHiveMQ_DashTresAtuad.putExtra("MQTT_BROKER",
"broker.hivemq.com");
        nuvemHiveMQ_DashTresAtuad.putExtra("MQTT_USUARIO", "");
        nuvemHiveMQ_DashTresAtuad.putExtra("MQTT_SENHA", "");
        // Inicie a DashTresAtuad com o Intent

```

```

        startActivity(nuvemHiveMQ_DashTresAtuad);
    } else {
        // Informe ao usuário que ele deve selecionar uma opção de cada grupo
        showToast("Por favor, selecione uma opção de cada grupo!");
    }
});
}

// Função para habilitar o Toast personalizado
public void showToast(String message) {
    ViewGroup view = findViewById(R.id.container_toast);
    View v = getLayoutInflater().inflate(R.layout.custom_toast, view);

    TextView txtMessage = v.findViewById(R.id.txt_toast);
    txtMessage.setText(message);

    Toast toast = new Toast(this);
    toast.setView(v);
    toast.setDuration(Toast.LENGTH_SHORT);
    toast.show();
}

// Função para desabilitar "Voltar" via botão do hardware
@Override
public void onBackPressed() {
    // Não faz nada ao clicar no botão voltar do hardware
}
}

```