UNIVERSIDADE FEREDAL DE UBERLÂNDIA

FACULDADE DE ENGENHARIA MECÂNICA

VINICIUS ROCHA NASCIMENTO

DESENVOLVIMENTO DE UM BROKER MQTT EM CLOUD E EDGE INTEGRADO COM APLICATIVO PARA AUTOMAÇÃO DE SISTEMAS A EVENTOS DISCRETOS

UBERLÂNDIA

2024

VINICIUS ROCHA NASCIMENTO

DESENVOLVIMENTO DE UM BROKER MQTT EM CLOUD E EDGE INTEGRADO COM APLICATIVO PARA AUTOMAÇÃO DE SISTEMAS A EVENTOS DISCRETOS

Projeto de Fim de Curso apresentado ao curso de graduação em Engenharia Mecatrônica da Faculdade de Engenharia Mecânica da Universidade Federal de Uberlândia, como parte dos requisitos para obtenção do título de BACHAREL em ENGENHARIA MECATRÔNICA.

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

UBERLÂNDIA

2024

VINICIUS ROCHA NASCIMENTO

DESENVOLVIMENTO DE UM BROKER MQTT EM CLOUD E EDGE INTEGRADO COM APLICATIVO PARA AUTOMAÇÃO DE SISTEMAS A EVENTOS DISCRETOS

Projeto de Fim de Curso apresentado ao curso de graduação em Engenharia Mecatrônica da Faculdade de Engenharia Mecânica da Universidade Federal de Uberlândia, como parte dos requisitos para obtenção do título de BACHAREL em ENGENHARIA MECATRÔNICA.

Aprovado em:_____de_____de _____

BANCA EXAMINADORA

Prof. Dr. Jose Jean-Paul Zanlucchi de Souza Tavares
Universidade Federal de Uberlândia

Profª. Dra. Gabriela Vieira Lima
Universidade Federal de Uberlândia

Prof. Dr. Werley Rocherter Borges Ferreira
Universidade Federal de Uberlândia

AGRADECIMENTOS

Primeiramente, agradeço à minha mãe Claudia Aparecida Rocha Nascimento e ao meu pai Luiz Henrique Nascimento Junior pelos cuidados, ensinamentos e por permitirem que eu tivesse acesso a todas as oportunidades que tive até hoje, tanto na vida pessoal, quanto na profissional. À minha família, especialmente irmãos, primos, avós e tios que me apoiam nas realizações dos meus sonhos quando nos encontramos em momentos mais do que especiais. Aos meus amigos, aos meus professores. À Universidade Federal de Uberlândia (UFU) e a Faculdade de Engenharia Mecânica (FEMEC) por providenciar um serviço público de qualidade para que eu me tornasse um Engenheiro Mecatrônico.

Queria agradecer a equipe de profissionais do laboratório, meu professor orientador José Jean-Paul Zanlucchi de Souza Tavares e ao monitor Alisson Carvalho Vasconcelos, se propuseram a ajudar pela empolgação de ver a máquina funcionando.

Queria agradecer acima de tudo ao meu filho de criação, foram os melhores 5 anos e 10 meses da minha vida e tudo porque você estava do meu lado, me incentivava a formar sem dizer nenhuma palavra, apenas a vontade de te ver usando um terno, dedico tudo isso a você, espero te ver em breve.

"How do I live without the ones I love?
Time still turns the pages of the book it's burned
Place and time always on my mind
I have so much to say but you're so far away"
(M. Shadows, 2010)

RESUMO

Este projeto tem o intuito de estudar o protocolo de comunicação MQTT, o uso da Internet das Coisas – IoT (*Internet of things)*. Uma estrutura de programação que está sendo muito desenvolvida nos dias de hoje por suas características. Está sendo chamada de 4ª revolução industrial por prometer mudar a estrutura dos processos industriais, colocar máquinas para comunicarem entre si sem precisar passar por um controlador logico programável (CLP) central, os atuadores e sensores deixariam de ser apenas escravos passivos e passarão a ser ativos, capazes de interpretar dados.

Abordaremos linguagens C++ para IDE do Arduino, aplicado a placas ESP32. Criarei um aplicativo em Java com o software Android Studio para ser a Interface Homem Máquina (IHM) do processo, permitindo que o usuário controle todo o sistema através do seu celular.

O sistema consiste em um protótipo de um separador de peças industrial, onde o usuário possui controle sobre quais peças serão dispensadas em cada saída, tudo isso de forma remota pelo aparelho celular.

Palavra-chave: MQTT, IoT, 4ª revolução industrial, CLP

ABSTRACT

This project aims to study the MQTT communication protocol, the use of the Internet of Things – IoT (Internet of things). A programming structure that is being developed a lot these days due to its characteristics. It is being called the 4th industrial revolution because it promises to change the structure of industrial processes, placing machines to communicate with each other without having to go through a central programmable logic controller (PLC), actuators and sensors would no longer be just passive slaves and would become active , capable of interpreting data. We will cover C++ languages for the Arduino IDE, applied to ESP32 boards. I will create a Java application with Android Studio software to be the Human Machine Interface (HMI) of the process, allowing the user to control the entire system through their cell phone.
The system consists of a prototype of an industrial parts separator, where the user has control over which parts will be dispensed at each exit, all remotely via a cell phone.

Palavra-chave: MQTT, IoT, 4ª revolução industrial, CLP

LISTA DE ILUSTRAÇÕES

FIGURAS

TABELAS

LISTA DE ABREVIATURAS E SIGLAS

MQTT - Message Queuing Telemetry Transport

M2M - Machine to Machine

IDE - Integrated Development Environment

IoT - Internet of Things

IIoT - Industrial Internet of Things

IP - Internet Protocol

TCP/IP - Transmission Control Protocol/Internet Protocol

GPIO - General Purpose Input/Output

CLP - Controlador Lógico Programável

WEB - World Wide Web

TLS - Transport Layer Security

QoS - Quality of Service

HTTP - Hypertext Transfer Protocol

IHM - Interface Homem-Máquina

SUMÁRIO

1 INTRODUÇÃO

A Indústria 4.0 representa uma nova era de automação e inteligência na manufatura, que se caracteriza pela utilização de tecnologias inteligentes e conectadas. O advento da Internet das Coisas (IoT) possibilitou a integração de máquinas, equipamentos e dispositivos diversos, permitindo a criação de ambientes altamente colaborativos e conectados.

Nesse contexto, o protocolo MQTT tem se destacado como uma das principais tecnologias utilizadas na Indústria 4.0. O MQTT é um protocolo de mensagens projetado para dispositivos com recursos limitados, permitindo a comunicação de dados em dispositivos IoT de forma eficiente e confiável.

Este protocolo por ser leve e robusto é perfeito para o chão de fábrica, um local que possui bastante interferência, além de ser possível passar um código completamente novo, setar a máquina para um novo processo, sem precisar fazer um novo código em ladder para o CLP.

O ESP32 foi essencial para o projeto, por ser uma placa programável já com módulo wifi e com saídas e entradas suficientes. Mas a ideia se estende para que cada sensor ou atuador tivesse sua própria placa como um ESP01.

O intuito do aplicativo mobile foi de facilitar e deixar mais prático para o usuário que irá operar a máquina, por ser facilmente instalado em celulares ou tablets o sistema de controle permite, através da comunicação em nuvem (broker), que seja atuado de qualquer local.

Todos os programas e códigos desenvolvidos são para operar um protótipo de separador de peças que há no Laboratório de Ensino de Mecatrônica 4 (LEM4), bancada usada para estudos de automação industrial, um ótimo exemplo de um processo real em uma fábrica controlada por CLP, consistindo em sensores indutivos, capacitivos e de fins de curso, além de atuadores eletropneumáticos controlados por válvulas de simples e dupla ação, um sistema completo.

1.1 Objetivos

1.1.1 Objetivo Principal

Este projeto tem como objetivo principal explorar a utilização do protocolo MQTT em um processo industrial controlado por meio de um aplicativo mobile desenvolvido em Java no software Android Studio através de um servidor Broker local criado em um Raspberry pi 4. A

aplicação desenvolvia terá como finalidade escolher como o sistema irá trabalhar, fazendo escolhas simples e segura remotamente através de um celular.

1.1.2 Objetivos Específicos

- Estudo do MQTT
- Utilização do Broker local
- Estudo de Java como ferramenta para desenvolvimento
- Explorar o uso do Arduino IDE para programação da ESP32
- Desenvolver uma aplicação Android
- Demonstrar o potencial da IIoT
- Explorar a capacidades e versatilidades da Industria 4.0

1.2 Justificativa do Projeto

A justificativa para este projeto é a necessidade de mudança que vemos nas indústrias de hoje, o sistema mestre-escravo é caro e lento comparado com a tecnologia que temos disponível. Tal campo da ciência tem que ser estudada para que possamos melhorar a qualidade, a velocidade e o custo da forma de produção, economizando milhões para o setor industrial, diminuindo o preço do produto para o consumidor

2 REVISÃO BIBLIOGRÁFICA

Neste tópico serão abordados os principais elementos presentes neste projeto como a plataforma ESP32, IIOT, protocolo MQTT, Android Studio, indústria 4.0.

2.1 ESP32

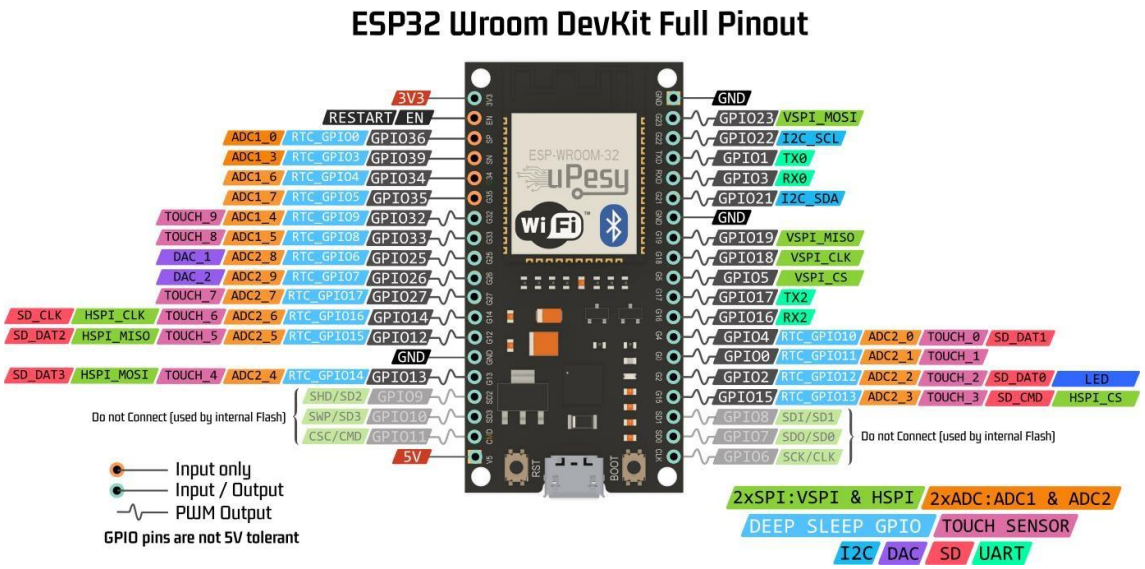
O ESP32 é uma placa microcontroladora desenvolvida pela Espressif Systems, da família do Arduino são famosos pela sua eficiência de processamento e baixo custo. Oferece suporte a diversas tecnologias de conexão, incluindo Wi-Fi, Bluetooth, ZigBee, tornando-o altamente versátil. Equipado com um processador dual-core de até 240 MHz, 520 KB de RAM e 4MB de memória flash interna, além de uma variedade de periféricos como UART, SPI, I2C e câmera,foi totalmente desenvolvido para projetos de IoT (ESP32).

Embora também seja aplicável em áreas como robótica, entretenimento, automação residencial, automotiva e agora industrial.

Sua flexibilidade de linguagens é um enorme diferencial na comunidade, permitindo aplicação em uma ampla gama de projetos além dos voltados para IoT. Graças à sua capacidade de programação em linguagens como C/C++ e sua compatibilidade com a Arduino IDE o ESP32 tornou-se uma escolha popular entre os desenvolvedores.

Na Figura 1 vemos os pinos presentes no ESP32, juntamente com as especificações das entradas e saídas.

Figura1: Pinagem ESP32



Fonte: https://www.upesy.com/blogs/tutorials/esp32-pinout-reference-gpio-pins-ultimate-guide

2.2 Protocolo MQTT

O protocolo de comunicação MQTT é uma rede baseada em TCP/IP, faz parte do protocolo de comunicação da camada de aplicação. Este protocolo utiliza o modelo de mensagem publicação/subscritor. Esse modelo implica que o editor e o assinante não trocam dados diretamente, mas realizam a interação de mensagens de forma indireta, por meio do intermediário chamado Broker, ele é responsavel por intermedias as mensagens entre as maquinas. (ENGPROCESS).

O protocolo oferece suporte à qualidade do serviço de publicação com base na identificação do QoS. Durante o processo de publicação, o cliente publicador envia uma mensagem ao Broker, e o processo de publicação e confirmação de envio está relacionado ao QoS.

Quando QoS=0, significa no máximo uma vez. Quando o Broker recebe uma mensagem PUBLISH do usuário, ele a encaminha para todos os assinantes inscritos neste tópico. Nesses

dois processos, o payload é enviado apenas uma vez e não tem a necessidade de recebimento de um ACK de confirmação.

Quando QoS=1, significa pelo menos uma vez. Após receber a mensagem PUBLISH do usuário, o Broker retorna uma mensagem de confirmação PUBACK. Em seguida, a mensagem PUBLISH pode ser enviada a todos os assinantes inscritos no tópico.

Quando QoS=2, significa exatamente uma vez, o Broker registra a mensagem PUBLISH recebida do usuário, mas só a envia a todos os assinantes após receber a mensagem PUBREL, o Broker envia a mensagem PUBLISH para todos os assinantes inscritos no tópico e envia uma mensagem PUBCOMP ao usuário.

Pelo lado do subscritor/assinante, inicialmente, o cliente estabelece uma conexão TCP com o servidor e envia uma mensagem CONNECT. Após receber a mensagem de confirmação CONNACK autorizada pelo Broker, o cliente envia uma mensagem SUBSCRIBE, especificando a lista de tópicos de interesse. Por fim, o servidor retorna uma mensagem de confirmação SUBACK, indicando que a assinatura foi bem-sucedida (Souza, R. et al.2022).

O MQTT usa o TLS como comunicação segura, para um Broker local, quando um usuário tenta conectar ele necessita de um usuário e senha pré-cadastrados no Broker quando iniciado, mas mesmo sendo uma comunicação segura o Broker ainda será capaz de saber a mensagem enviada (Santos, P. et al. 2020).

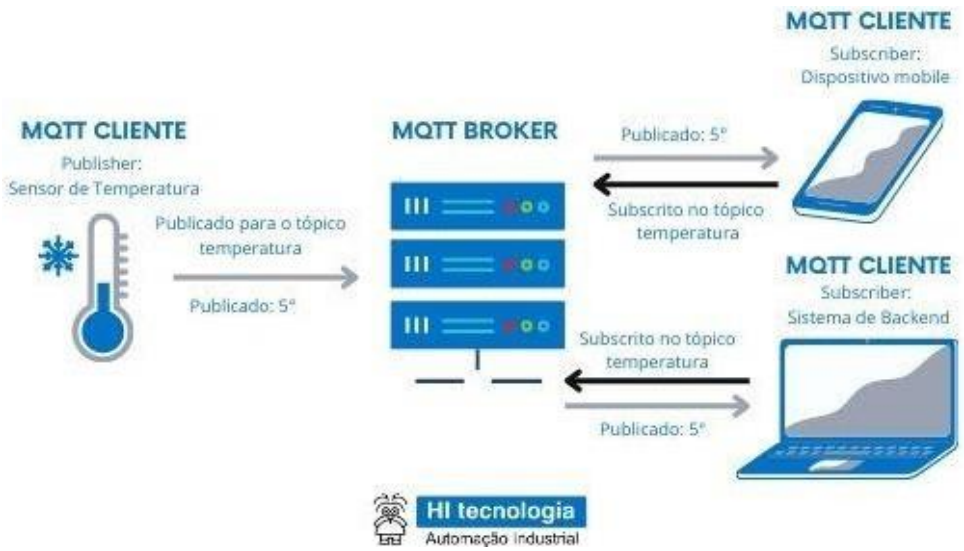
Por conta de todas essas qualidades este protocolo é perfeito para uma comunicação M2M, com dados leves, robusto, e em tempo real é perfeito para a comunicação entre máquinas no setor industrial.

Por exemplo, um sensor e um atuador, ligados fisicamente na alimentação, conectados ao Broker via Wifi, o sensor publicando as informações coletadas, logo a frente no processo, um atuador subscrevendo a informação do sensor no tópico atua de acordo com o desejado.

Na figura 2 mostra esquematicamente uma comunicação entre um sensor via MQTT que pode ser acessado via mobile ou computador.

O Broker do protocolo serve como um servidor armazenando tópicos, podemos exemplificar como sendo blocos de memória, cada tópico criado é um bloco de memória e nele é possível armazenar uma informação(payload), um usuário ou uma máquina pública uma mensagem em um tópico, outro usuário ou máquina acessa esse tópico fazendo um Subscribe, de acordo com a mensagem buscada a máquina ou o usuário fara a devida tratativa e ação. Na figura 3 temos o ESP que foi usado para leitura dos sensores de entrada.

Figura 2: Infraestrutura de Comunicação do MQTT



Fonte: https://www.hitecnologia.com.br/o-que-e-o-protocolo-mqtt/

2.3 IIot e a Industria 4.0

A Internet Industrial das Coisas (IIoT) se refere à aplicação da tecnologia da internet das coisas em ambientes industriais, onde máquinas, sensores e atuadores se conectarão na rede e comunicarão entre si. As redes IIoT são projetadas para suportar a comunicação entre máquinas e a transmissão regular de dados entre o sistema central e todos os dispositivos IoT integrados essa tecnologia é a essência da iniciativa relacionada à Indústria 4.0 (SAP).

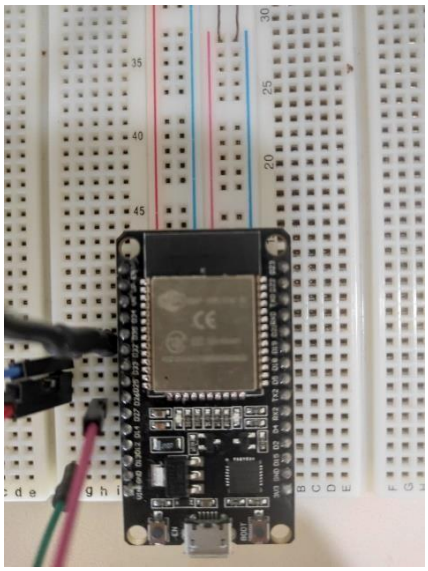
Algumas das tecnologias usadas em IIoT com o protocolo MQTT são (SAP):

- Sensores IIoT: Os sensores são comumente integrados a equipamentos industriais e máquinas. No entanto, mesmo máquinas analógicas e equipamentos de fabricação podem ser equipados com dispositivos de gateway IoT, como câmeras e medidores. Isso permite que os ativos da IIoT detectem condições em seu ambiente, como proximidade de outros objetos, pressão do ar ou umidade, além de monitorar a velocidade do motor, níveis de fluido e outras condições mecânicas.
- Poder da computação em nuvem e edge computing: Tanto a computação em nuvem quanto o edge computing têm melhorado significativamente a flexibilidade e a usabilidade da IIoT. Através da nuvem, as redes IIoT podem

aproveitar um alto poder de processamento e capacidade de armazenamento conforme necessário. Isso possibilita que os dispositivos dentro da rede coletem e transmitam conjuntos de dados maiores e mais complexos. Já o edge computing envolve trazer sistemas capazes de processar e analisar esses dados para mais perto da rede IIoT. Isso reduz a latência e os atrasos, permitindo o processamento em tempo real de dados sensíveis ao tempo.

- IA e Machine Learning: As tecnologias de inteligência artificial e machine learning possibilitam que as empresas processem dados da IIoT usando funções analíticas preditivas avançadas. Bancos de dados e algoritmos modernos ajudam as empresas a gerenciarem e dar sentido a uma variedade de conjuntos de dados complexos e não estruturados.
- Segurança para sistemas ciberfísicos: A mesma conectividade que impulsiona as redes IIoT também as expõe a riscos de segurança. Embora a maioria das empresas tenha protocolos de segurança em torno de seus sistemas e bancos de dados centrais, seus dispositivos IoT podem ser vulneráveis. É essencial implementar protocolos e tecnologias de segurança robustas.

Figura 3: ESP32



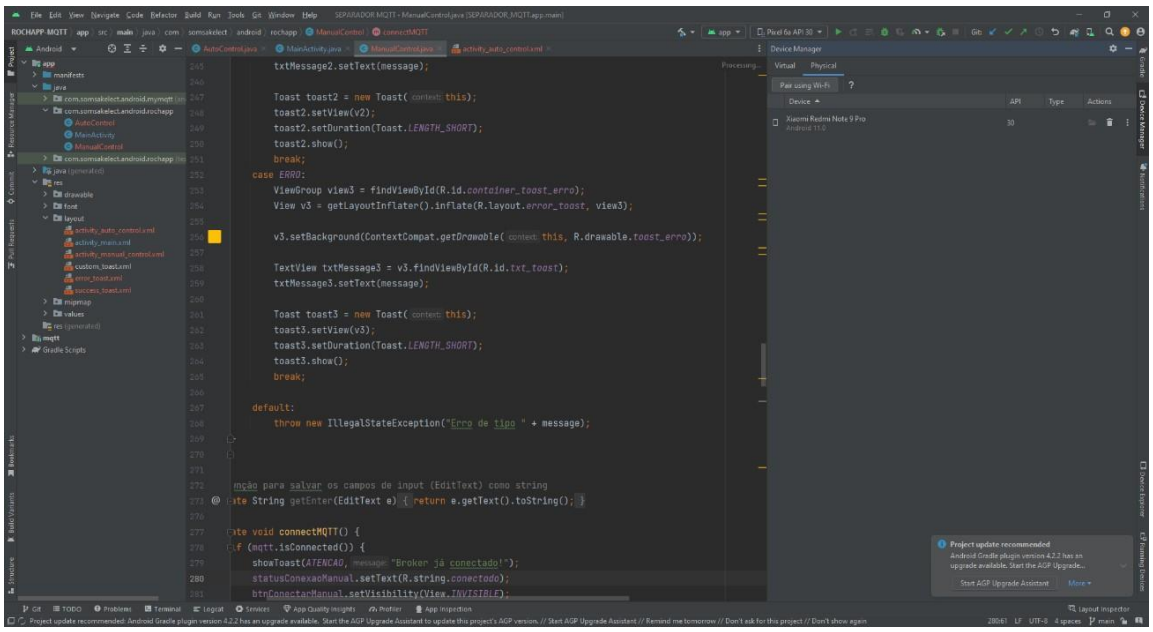
Fonte: Autoria própria

2.4 Android Studio e Java

Com intuito mais didático o aplicativo desenvolvido para controle foi feito para o sistema Android, a escolha teve pôr os aparelhos celulares de todos os integrantes do laboratório serem compatíveis, mas a ideia não se limita a tal sistema.

O software usado para programação do aplicativo foi o Android Studio, a linguagem nativa da plataforma para front e Backend usada foi o Java.

Figura 4: código Backend da tela manual do aplicativo de controle



Fonte: Autoria Própria

O interessante deste modelo é que pode ser criado a interface que desejar, pode ser o mais prático, intuitivo e ao mesmo tempo temático e visual para melhor compreensão do usuário.

3 METODOLOGIA

Neste momento será falado como procedeu o desenvolvimento e a implementação deste projeto. Será descrito os processos e materiais que serão usados para a aplicação e a comunicação com o protótipo da separadora de peças através do protocolo MQTT. Os apêndices A, B, C, D,E e F tem os detalhes dos códigos dos ESPs e do Android Studio.

3.1 Lista de Materiais Usados

a) 3 Placas ESP32;
b) 1 Raspberry pi 4;
c) Protoboard;
d) Jumpers;
e) 09 relés contatores cada um deles usando um contato normalmente fechado;

f) 2 módulos relés para Arduino usando o contato normalmente fechado;
g) 1 atuador de dupla ação e 2 atuadores de simples ação com retorno por mola;
h) 1 válvula eletropneumática 5/2 dupla solenoide e 2 válvulas eletropneumática 3/2 simples solenoide;
i) ar comprimido;
j) 04 sensores fim de curso;
k) 3 sensores ópticos, 1 sensor indutivo e 1 sensor capacitivo;
l) Um computador com Arduino IDE e Android Studio instalados;
m) Celular com sistema operacional Android;
n) Servidor Broker EMQX para testes;
o) Servidor Broker local criado no Raspeberry pi 4;

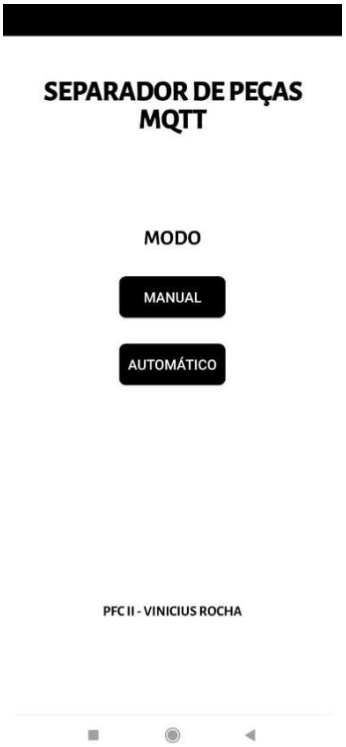
3.2 Aplicação implementada

O projeto em questão é uma aplicação do protocolo MQTT na IIoT. Foi usado microcontroladores e relés no lugar de um CLP para controlar um processo industrial, e a IHM de controle é um aplicativo mobile que pode ser instalada em qualquer aparelho celular, substituindo também um controlador fixo a máquina no chão de fábrica

3.2.1 Aplicativo

Seguindo uma ordem do processo, o primeiro passo fica por conta de o usuário escolher no aplicativo como ele quer controlar o sistema, se irá avançar e recuar os cilindros de forma "manual" ou se a maquina ira trabalhar de forma automática. Na figura 5 temos a tela inicial do aplicativo.

Figura 5: Tela inicial do aplicativo



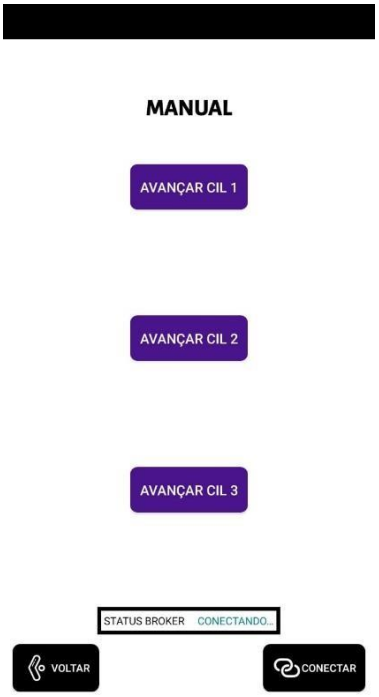
Fonte: Autoria própria

Caso escolher manual, o usuário é direcionado para a tela mostrada na figura 6, nela ao clicar nos botões em roxo com o texto “AVANÇAR CIL 1”, “AVANÇAR CIL 2” ou “AVANÇAR CIL 3”, irá fazer com que o aplicativo publique um comando no tópico “/valv/cilindroacao”, fazendo que o ESP responsável pelos atuadores acione o avanço das respectivas válvulas.

Além disso, ele irá mudar de cor e o texto passara a informar “RECUAR CIL 1” que se acionado novamente uma nova mensagem é publicada fazendo que o cilindro recue.

A figura 7 mostra o backend da tela manual, nela é mostrada as funções de avanço do cilindro 1 e 2, e recuo do cilindro 1, explicitado em vermelho o tópico(“/valv/cilindroacao”) e as mensagens(payload) que serão publicadas, ao acionar o botão de avanço do cilindro 1, a mensagem “2” é publicada através da chamada apontada (mqtt.publish(topic, message);), ao acionar o botão novamente é publicado “1” fazendo o cilindro recuar.

Figura 6: tela manual do aplicativo



Fonte: Autoria própria

Figura 7: Backend da tela manual do aplicativo

```
public void AvanCilind1() {
    String topic = "/valv/cilindroacao";
    String payload = "2"; // Avança o cilindro 1
    try {
        byte[] encodedPayload = payload.getBytes(StandardCharsets.UTF_8);
        MqttMessage message = new MqttMessage(encodedPayload);
        mqtt.publish(topic, message);
    } catch (MqttException e) {
        e.printStackTrace();
    }
}

public void RecuaCilind1() {
    String topic = "/valv/cilindroacao";
    String payload = "1"; // Recua o cilindro 1
    try {
        byte[] encodedPayload = payload.getBytes(StandardCharsets.UTF_8);
        MqttMessage message = new MqttMessage(encodedPayload);
        mqtt.publish(topic, message);
    } catch (MqttException e) {
        e.printStackTrace();
    }
}

public void AvanCilind2() {
    String topic = "/valv/cilindroacao";
    String payload = "4"; // Avança o cilindro 2
    try {
        byte[] encodedPayload = payload.getBytes(StandardCharsets.UTF_8);
        MqttMessage message = new MqttMessage(encodedPayload);
        mqtt.publish(topic, message);
    } catch (MqttException e) {
        e.printStackTrace();
    }
}
```

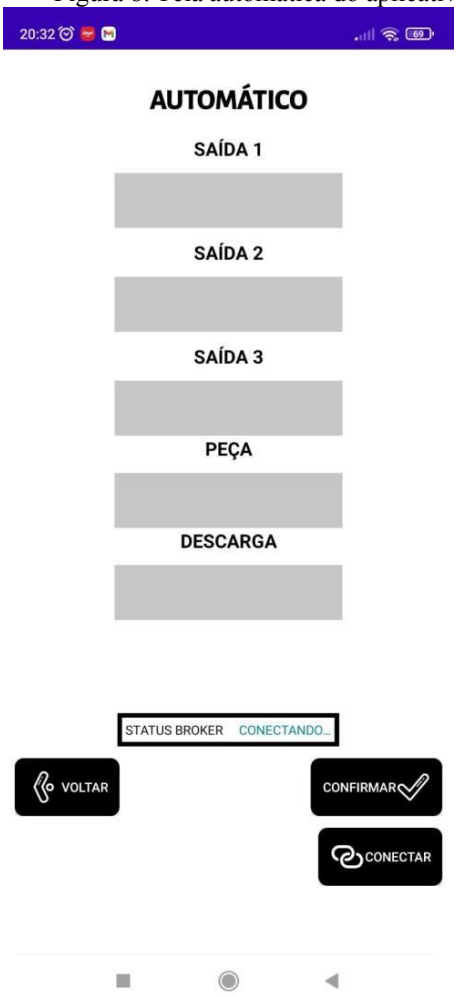
Fonte: Autoria própria

Caso o usuário escolher a opção automático ele será encaminhado para a tela da

imagem 8. Aqui o usuário irá setar a máquina para ela trabalhar em loop, deve escolher qual peça deve ir em cada uma das 3 saídas, não podendo escolher peças iguais para saídas diferentes, após a escolha o programa irá ficar esperando uma mensagem no tópico "/sensor/peça", sempre que algum usuário modifica o conteúdo do tópico o broker encaminha essa nova mensagem a todos os equipamentos inscritos nesse tópico, assim sempre que uma peça nova passar o tópico é atualizado informando ao aplicativo, que irá comparar a informação recebida com as escolhas do usuário em cada saída. De acordo com o resultado da comparação o aplicativo irá publicar no tópico “/valv/cilindroacao”, qual válvula deve ser acionada.

Após o avanço do cilindro e a peça cair na descarga, ela aciona um sensor fim de curso que está ligado ao ESP responsável pelos sensores de saída, este irá publicar no tópico "/valv/recuo" uma mensagem dizendo em qual saída a peça se encontra, o aplicativo por sua vez, subscreve essa informação e publica no tópico “/valv/cilindroacao” o comando para recuar o cilindro que estava avançado.

Figura 8: Tela automática do aplicativo



Fonte: autoria própria

Na tela automática, mostrada na figura 8, possui as 3 saídas que o usuário deveescolher entre os 6 tipos de peças que irão ser selecionadas, um bloco com o título PEÇA que irá mostrar para o usuário qual peça está na linha, e o bloco DESCARGA que irá mostrar em qual saída a peça está.

Agora falando sobre o código, a imagem 9 mostra o comando Subscribe da biblioteca do MQTT para subscrever os tópicos dos sensores de entrada e dos sensores de saída para quando a peça sair da linha. Na figura 10 tem-se a parte do código que busca a mensagem no tópico "/sensor/peça" e compara essa mensagem com uma das saídas escolhidas pelo usuário, caso alguma delas sejam iguais o aplicativo irá publicar usando a chamada publishToTopic o valor que representa qual cilindro deve avançar.

Figura 9: Comando MQTT para subscrever os tópicos dos sensores.

```
// MQTT
mqtt = new MqttAndroidClient( context: this, MQTT_URL, MQTT_ID);
mqtt.setCallback(new MqttCallbackExtended() {
    @Override
    public void connectComplete(boolean reconnect, String serverURI) {
        statusConexaoAuto.setText(reconnect ? "Reconectando..." : "Conectado!");
        if (mqtt.isConnected()) {
            String tsx = "CONECTADO!";
            statusConexaoAuto.setText(tsx);
            btnConectarAuto.setVisibility(View.INVISIBLE);
            // Subscrever tópicos aqui
            subscribe( topic: "/sensor/peça"); // Recebe o status do ESP32 dos sensores de entrada
            subscribe( topic: "/valv/recuo");  // Recebe o status do ESP32 dos sensores de saida
        }
    }
```

Fonte: Autoria própria

Figura 10: Código que compara os inputs do usuário com a mensagem do tópico.

```
@Override
public void messageArrived(String topic, MqttMessage message) {
    if (topic.equals("/sensor/peca")) {
        String sensIN = message.toString();
        // Exibir a mensagem na caixa de texto com o ID "ind_peca"
        TextView peca = findViewById(R.id.ind_peca);
        if (peca != null) {
            peca.setText(sensIN);
        }

        // Comparar com opcaoSpinner1
        if (opcaoSpinner1 != null && opcaoSpinner1.equals(sensIN)) {
            // Publicar no tópico "/valv/cilindroacao" com payload "2"
            publishToTopic( topic: "/valv/cilindroacao",  payload: "2");
        }

        // Comparar com opcaoSpinner2
        if (opcaoSpinner2 != null && opcaoSpinner2.equals(sensIN)) {
            // Publicar no tópico "/valv/cilindroacao" com payload "4"
            publishToTopic( topic: "/valv/cilindroacao",  payload: "4");
        }

        // Comparar com opcaoSpinner3
        if (opcaoSpinner3 != null && opcaoSpinner3.equals(sensIN)) {
            // Publicar no tópico "/valv/cilindroacao" com payload "6"
            publishToTopic( topic: "/valv/cilindroacao",  payload: "6");
        }
    }
    if (topic.equals("/valv/recuo")) {
        //String sensorB;
        String sensOUT = message.toString();
        // Exibir a mensagem na caixa de texto com o ID "ind_saida"
        TextView saida = findViewById(R.id.ind_saida);
        if (saida != null) {
            saida.setText(sensOUT);
```

Fonte: Autoria própria

Na figura 11 é mostrado as peças com seus tamanhos e materiais e na figura 12 a bancada separadora de peças completa que o aplicativo em questao irá controlar. Nela podemos veras 4 rampas de descarga com seus sensores de fim de curso, os sensores de entrada, os atuadores, a esteira e o painel de comandos.

Figura 11: Peças

Fonte: Autoria própria

Figura 12: Bancada separadora de peças.

Fonte: Autoria própria

3.2.2 ESP32_SENSOR_IN

O primeiro ESP32 (Figura 3) foi usado para coletar os dados dos sensores de entrada, interpretar e publicar no tópico "/sensor/peça" qual o tipo de peça que entrou na linha, o controlador recebera sinal de 5 sensores, sendo 3 deles ópticos, 1 capacitivo e 1 indutivo.

São colocados 3 sensores óticos em alturas diferentes, em relação a esteira, caso todos eles deem sinal positivo a peça é grande, caso apenas o mais baixo de sinal logico positivo a peça por tanto é pequena. Continuando na esteira a peça passa pelo sensor indutivo, que avisara caso a peça seja feita de metal, e por último temos um sensor capacitivo para controle, quando a peça passa por ele o ESP faz a tratativa de acordo com os sinais recebidos, publica no tópico informando qual peça está na esteira e zera as variáveis dos sensores.

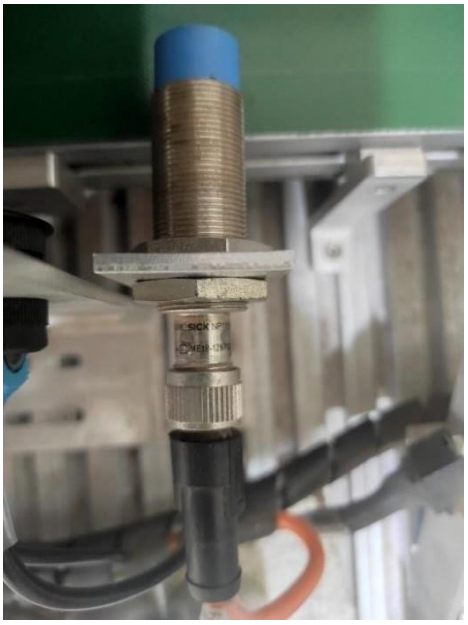
Figura 13: Sensores óticos posicionados com alturas diferentes.

Fonte: Autoria própria

Na figura 13 tem-se os 3 sensores responsáveis por informar o tamanho da peça, F2(lado esquerdo) posicionado para ler peças de tamanho mediano, F1(central) para peças pequenas e F3(lado direito) peças grandes.

A figura 14 mostra o sensor indutivo usado para captar as peças metálicas.

Figura 14: sensor indutivo.



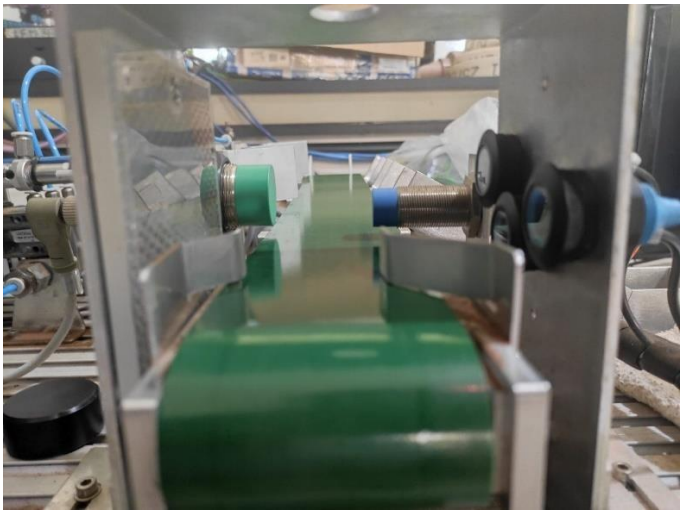
Fonte: Autoria própria.

Figura 15: Sensor capacitivo.

Fonte: Autoria própria.

A imagem 15 tem-se o sensor capacitivo responsável por avisar que a peça chegou no final dos sensores.

Figura 16: montagem dos sensores de entrada.



Fonte: Autoria própria

Na figura 16 tem-se a forma que estão distribuídos os sensores na linha. Mostrado a parte física será detalhado as partes mais importantes do código agora.

Inicialmente é definido variáveis que serão usados no corpo do programa (Fgura 17), além do broker local (mqttserver) e a porta (1883), cria um tópico("/sensor/peça") que será usado para armazenar a informação de qual peça entrou na linha

Figura 17: Variáveis do ESP32 do sensor de entrada.

COD_SENSORESINCERTO_DUAL_CORE.ino

```
30
31  /* DECLARATION OF MQTT SERVER VARIABLES */
32  const char* mqttserver = "10.14.48.104";
33  const uint16_t mqttport = 1883;
34  // const char* mqttUser = "rocha";
35  // const char* mqttpsswrd = "rocha";
36
37  /* DECLARATION OF TOPICS VARIABLES */
38  const char* topsenIN = "/sensor/peça";
39
40  /* DECLARATION OF A CLASS VARIABLE */
41  WiFiClient wifiClient;
42  PubSubClient MQTT(wifiClient);
43  TaskHandle_t Task1;
44
45  /* DEFINITION OF GPIO PINES VARIABLES */
46  #define sP 33
47  #define sM 25
48  #define sG 12
49  #define sMET 26
50  #define sCONT 27
51
52  /* DEFINITION OF A CONSTANT VARIABLES */
53  #define MSG_BUFFER_SIZE (24)
54  #define WAIT_CAP 1
55
56  /* DECLARATION OF GLOBAL VARIABLES */
57  bool clesP = 0;
58  bool clesM = 0;
59  bool clesG = 0;
60  bool clesMET = 0;
61  bool clesCONT = 0;
62  bool flag = false;
63  unsigned long timercap = 0;
64  char msg[MSG_BUFFER_SIZE];
65
66  /* DECLARATION OF FUNCTIONS WITH DEAFULT INPUT VALUES */
67  void conectarWiFi(int timeout = 30);
```

Output

Fonte: autoria própria

Na figura 18 tem-se o código responsável pela leitura dos sensores, tratativa e publicação. O comando usado da biblioteca PubSubClient.h foi o (MQTT.publish(topsenIN, msg, 2);) que temos o tópico, a mensagem (payload) e a QoS desejado para a publicação.

Figura 18: void loop do ESP32 do sensor de entrada.

```
void loop() {
  delay(10);

  if (!clesM) clesM = digitalRead(sM);
  if (!clesP) clesP = digitalRead(sP);
  if (!clesG) clesG = digitalRead(sG);
  if (!clesMET && !flag) clesMET = digitalRead(sMET);
  if (!clesCONT && !flag) clesCONT = digitalRead(sCONT);

  if (!flag) {
    timercap = millis();
    if (clesCONT) flag = true;
  } else if (int(millis() - timercap) >= WAIT_CAP*1000) flag = false;

  //printf("sensor pequeno: %d\tsensor medio: %d\tsensor grande: %d\tsensor capacitivo: %d\tsensor metalico: %d\n", clesP, clesM, clesG, clesCONT, clesMET);

  if(clesP && clesCONT){
    printf("sensor pequeno: %d\tsensor medio: %d\tsensor grande: %d\tsensor capacitivo: %d\tsensor metalico: %d\n", clesP, clesM, clesG, clesCONT, clesMET);
    if (!clesM && !clesG && !clesMET) snprintf (msg, MSG_BUFFER_SIZE, "Peça Pequena");
    if (clesM && !clesG && !clesMET) snprintf (msg, MSG_BUFFER_SIZE, "Peça Média");
    if (clesM && clesG && !clesMET) snprintf (msg, MSG_BUFFER_SIZE, "Peça Grande");
    if (!clesM && !clesG && clesMET) snprintf (msg, MSG_BUFFER_SIZE, "Peça Pequena Metallica");
    if (clesM && !clesG && clesMET) snprintf (msg, MSG_BUFFER_SIZE, "Peça Média Metallica");
    if (clesM && clesG && clesMET) snprintf (msg, MSG_BUFFER_SIZE, "Peça Grande Metallica");

    printf("%s\n", msg);

    MQTT.publish(topsenIN, msg, 2);
    clesP = 0;
    clesM = 0;
    clesG = 0;
    clesMET = 0;
    clesCONT = 0;
  }
}
```

Fonte: Autoria Própria.

3.2.3 ESP32_ATUADOR

O segundo ESP a atuar será o responsável por acionar as válvulas. Este microcontrolador irá acessar o tópico "/valv/cilindroacao" e de acordo com o payload ele irá acionar uma saída digital acionando um relé que comutará a válvula. Os comandos da tabela 1 são recebidos no tópico pelo aplicativo.

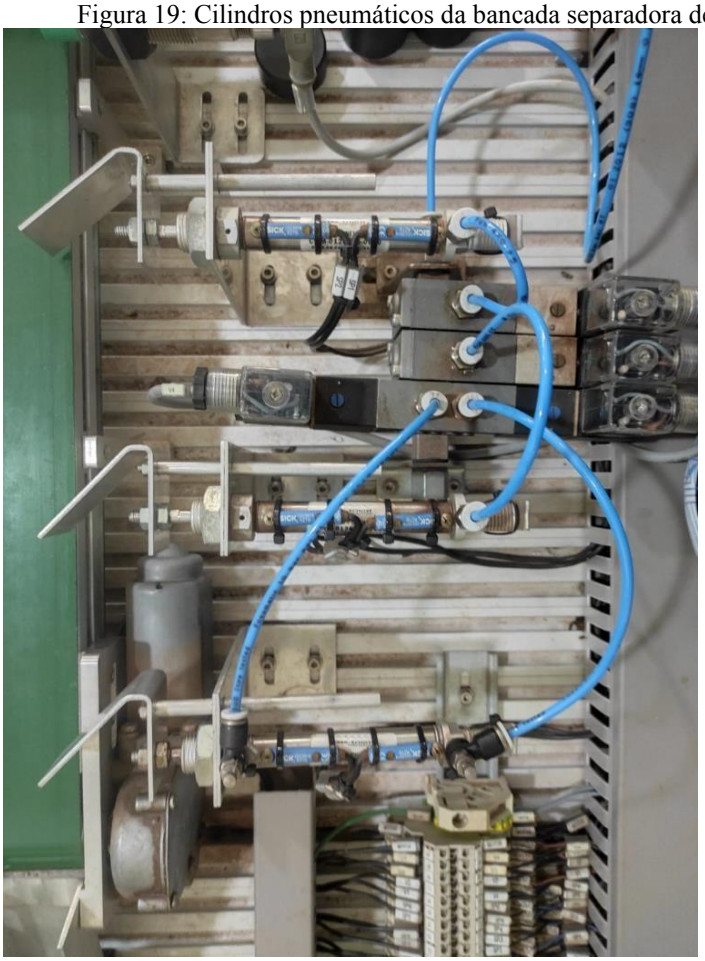
Tabela 1- Comandos de acionamento das válvulas.

Payload	Ação
1	Recuar Cilindro 1
2	Avançar Cilindro 1
3	Recuar Cilindro 2
4	Avançar Cilindro 2
5	Recuar Cilindro 3
6	Avançar Cilindro 3

Fonte: Elaborada pelo autor.

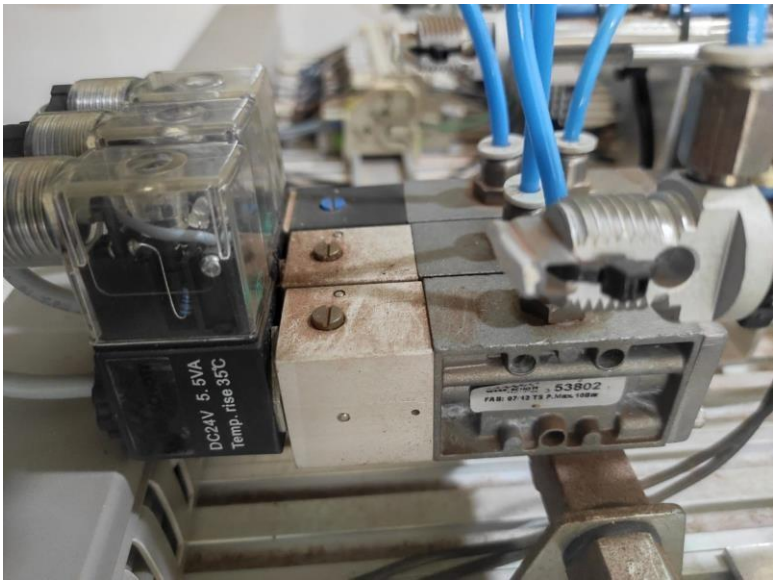
Nas imagens 19 e 20 tem-se os atuadores pneumáticos e as válvulas usadas no projeto, note que são duas válvulas de simples solenoide com cilindro de simples ação com retorno por mola e uma de dupla solenoide com cilindro de dupla ação.

Figura 19: Cilindros pneumáticos da bancada separadora de peças.



Fonte: Autoria própria.

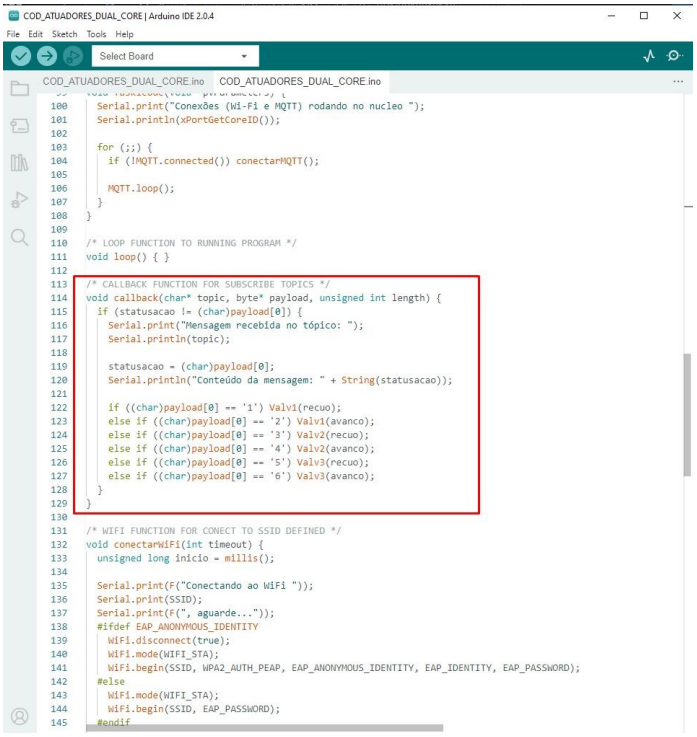
Figura 20: Válvulas da bancada separadora de peças.



Fonte: Autoria própria.

Na imagem 21 é mostrado um pedaço do código do ESP, tem-se a função callback com os valores correspondentes de cada ação das válvulas. Para cada payload é chamado uma função com acionamento e desacionamento das portas digitais.

Figura 21: Função callback do ESP32 ligado as válvulas eletropneumáticas.



Fonte: autoria própria

3.2.4 ESP32_SENSOR_OUT

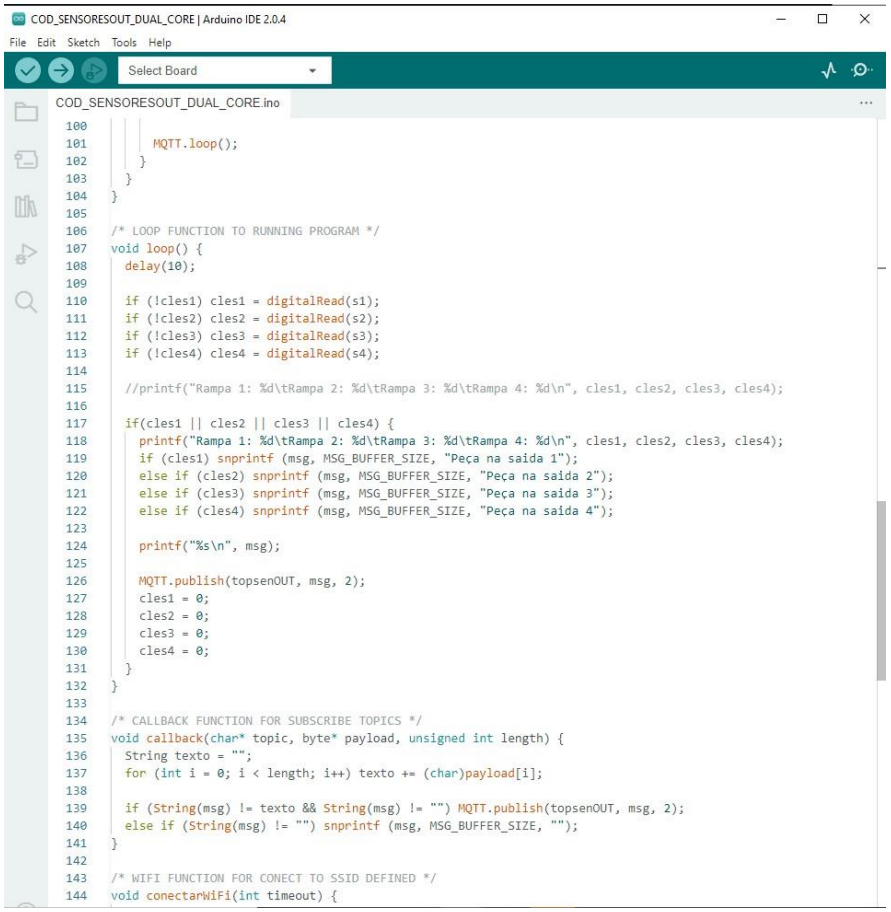
O terceiro ESP a atuar será o responsável por saber em qual saída a peça está, de acordo com o sensor de fim de curso acionado na rampa ele irá informar ao broker no tópico "/valv/recuo" qual das 4 saídas a peça se encontra. Ao publicar a informação o ESP responsável pelos atuadores recua, imediatamente, o cilindro que estava avançado. Na figura 22 é mostrado as saídas de descargas das peças e os sensores de fim de curso da bancada.

Figura 22: Rampas de descarga da bancada separadora de peças.

Fonte: autoria própria.

Na figura 23 tem-se o loop, um pedaço do código onde mostra como que irá fazer a tratativa de qual fim de curso foi acionado e publicar a mensagem. Esta que será recebida pelo aplicativo mostrando para o usuário em qual saída a peça está e que por sua vez publicara no tópico dos atuadores para recuar o cilindro.

Figura 23: Código usado no ESP32 para leitura dos sensores de saída.



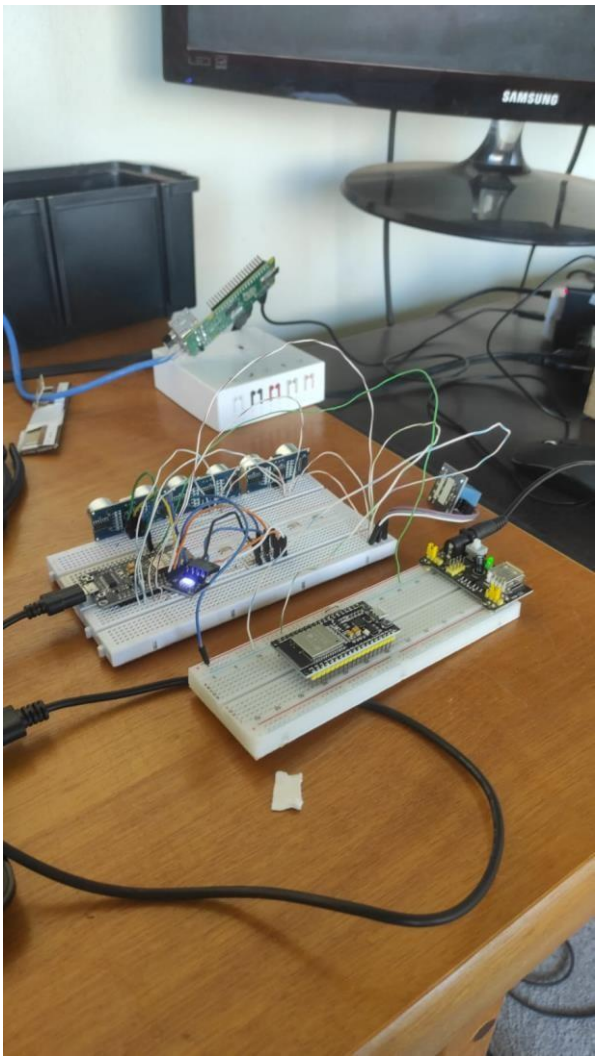
Fonte: autoria própria.

4 RESULTADOS E DISCUSSÃO

Neste item será apresentado os resultados dos testes feitos separadamente de cada processo, e o teste em conjunto, será falado sobre os protótipos feitos com sensores ultrassônicos ebrokers de sites, aplicativo de dashboard para teste de subscrição e publicação do protocolo MQTT.

Inicialmente buscando conhecimento sobre o protocolo em questão foi encontrado muitas informações teóricas, mas poucos exemplos práticos e com explicação, isso forçou a tentativa e erro. Com a ajuda de dois estudantes que também estavam pesquisando sobre o tema, conseguimos desenvolver os primeiros códigos para teste de comunicação via MQTT com o ESP, foi desenvolvido um programa de leitura de distancias com sensores ultrassônico que acionava um led, a distância limite era controlada via MQTT DASHBOARD, caso o sensor lesse uma distância menor que a escolhida como limite um led era acionado.

Figura 24: Protótipo limitador de distância.



Fonte: autoria própria.

Na imagem 24 tem-se 2 ESPs, um deles montado com 3 sensores ultrassónicos na protoboard, um led e um relé como atuadores, e o Raspberry pi 3 sendo o broker local.

Os resultados foram promissores, as leituras eram publicadas pelo ESP e o dashboard fazia o subscibe e mostrava na tela para o usuário, e de acordo com o limite escolhido pelo usuário e publicado no tópico do limite o led mudava da cor azul para vermelha, basicamente um sensor de ré veicular.

4.1 Montagem da bancada

Queria primeiramente agradecer a Faculdade Federal de Uberlândia por disponibilizar a bancada separadora de peças usada nas aulas de automação industrial, a bancada simula vários processos de uma indústria e é usado para disciplinas com CLP e

linguagem ladder, uma linguagem de baixo nível, neste projeto ela será controlada por ESPs. A bancada e os relés foram alimentadas com 24V, e na comutação dos contatos dos relés usado 5V dos próprios controladores.

Um detalhe muito importante que vale ressaltar é que todos os equipamentos devem estarem aterrados no terra comum.

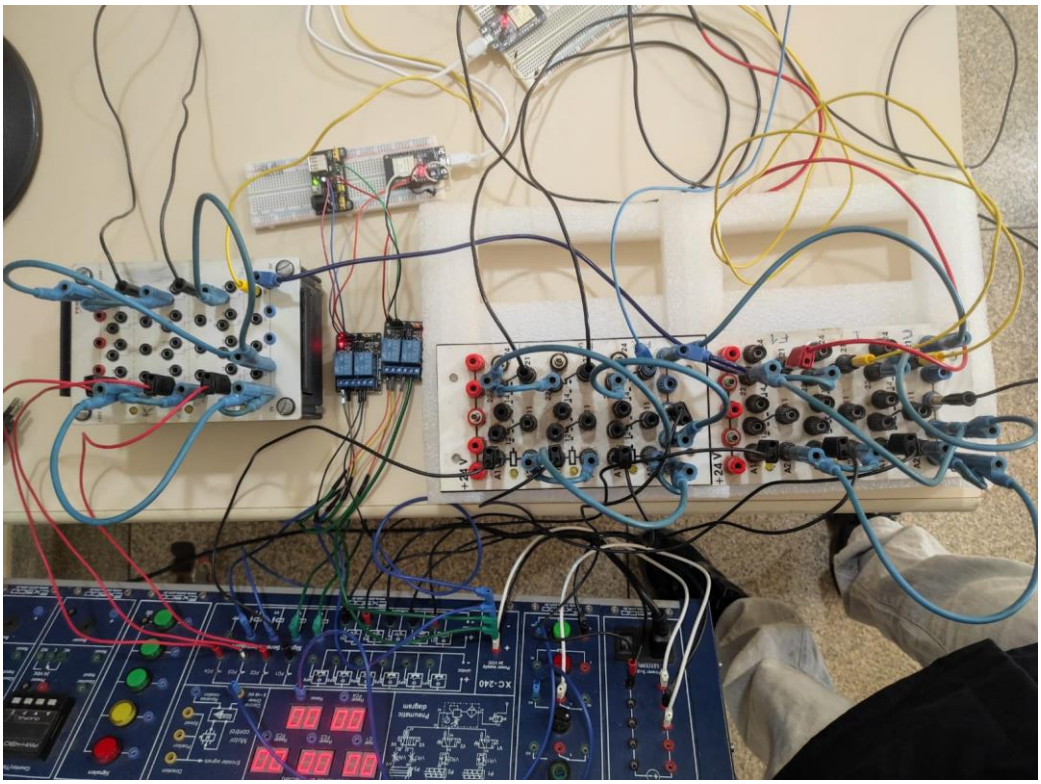
Na figura 25 tem-se a imagem do painel de controle com as entradas e saidas de sinal a 24Vdc da bancada, onde será feito as ligação das portas dos sensores aos relés.

Figura 25: Painel de controle da bancada.

Fonte: autoria própria.

Da bancada será ligado jumpers com ponteiras adaptas para poder conecta-la aos relés. Na imagem 26 é mostrado a montagem da bancada nos relés, foi feito uma montagem pullup, quando tiver tensão nas portas dos ESPs o sinal lógico lido será 0 e quando não tiver o sinal será 1. Para isso foi conectado 5Vdc vindo dos controladores nos conectores normalmente fechados dos relés, assim quando o relé receber um sinal vindo da bancada comutará as chaves e cortara a energia nas portas fazendo o ESP ler um sinal.

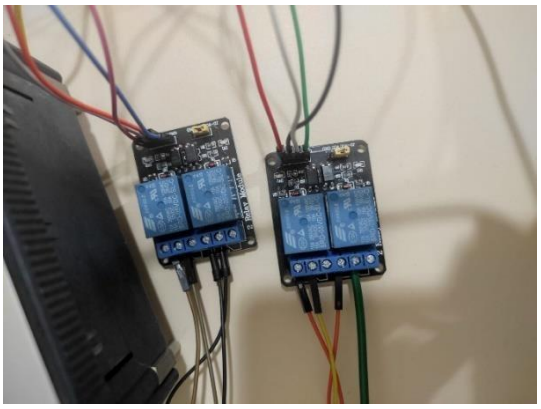
Figura 26: Relés de controle.



Fonte: autoria própria.

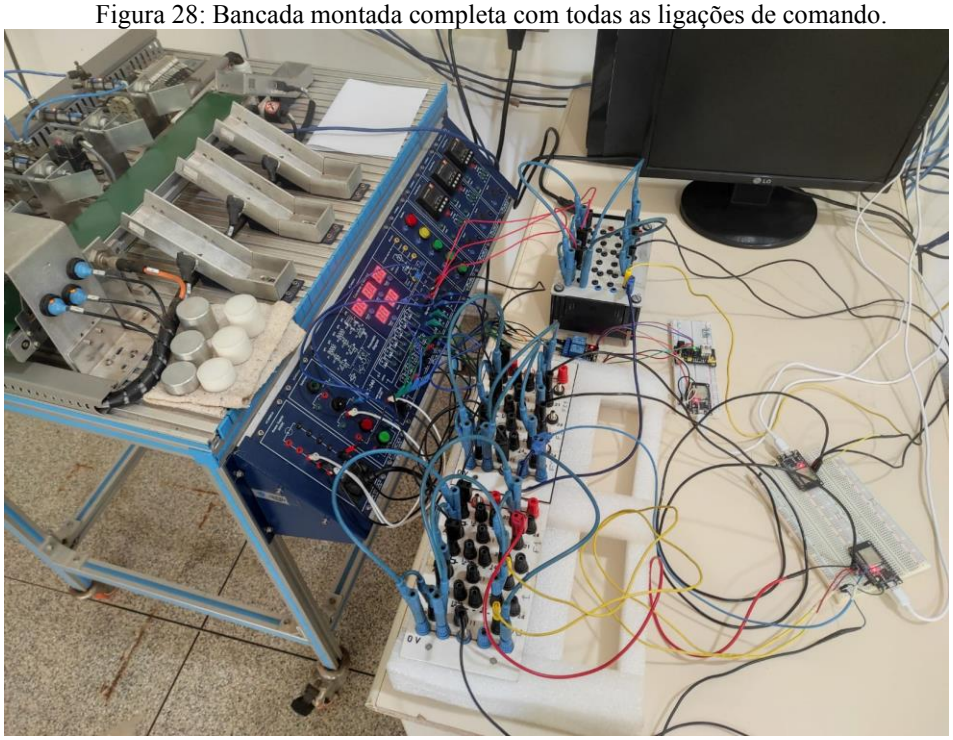
Na imagem 27 tem-se os módulos relés, muito usados em projetos com Arduino, também alimentado com com 5Vdc dos controladores, quando recebem um sinal comutam a chave que está ligada à bancada, nas entradas de acionamento das válvulas dos cilindros. Na figura 28 tem-se a bancada montada completa.

Figura 27: Módulos relé 5V.



Fonte: autoria própria.

Figura 28: Bancada montada completa com todas as ligações de comando.



Fonte: autoria própria.

4.2 Resultados obtidos

Os testes iniciais foram nos sensores de entrada, inicialmente foi feito a leitura dos sensores separados e publicados seus resultados no serial da IDE do Arduino, com uma montagem sem ser do tipo pullup as leituras sofriam bastante interferência, com esse tipo de montagem o sistema passou a responder melhor. Problemas com comunicação e perda de conexão com os brokers dos sites tambem foram enfrentados, devido ao nível de proteção da rede da faculdade, essas desconexões faziam que o sistema não publicasse a mensagem no tópico ou publicava após um grande delay, esse problema foi resolvido quando usado um broker local gerado no Raspberry Pi 4 e a implementação do código usando os 2 núcleos do ESP, foi separada a ação de ler os sensores em um núcleo e a de publicar no outro núcleo.

O segundo processo a ser testado foram os atuadores, o teste consistia em acionar as 3 válvulas via aplicativo pela tela manual, os testes foram conclusivos pois a publicação do aplicativo no broker, a subscrição do ESP no mesmo, e a atuação do controlador com os relés 5V comutando e acionando as válvulas 24V, foram perfeitos, foi alcançado a resposta em tempo real desejada, nesse momento foi possível fazer a lógica na tela automática no aplicativo para um teste futuro. Neste controlador foi implementado o código com 2 núcleos também com o mesmo intuito do sensor de entrada, um núcleo fazer a subscrição no tópico e o outro núcleo controlar as saídas.

O terceiro teste, e o mais simples de todos, foi o ESP com os sensores de fim de curso,

neste não seria necessário o código com 2 núcleos, mas foi implementado para diminuir falhas por perda de conexão com o broker. O controlador tinha a função de ler o sensor e publicar em um tópico qual sensor foi acionado, identificando a saída que a peça está.

O quarto teste foi a tela automática do aplicativo, o usuário escolhia qual peça irá em cada saída, era mandado uma peça na esteira e esperava-se que o cilindro correto fosse acionado, novamente o problema de comunicação e delay com os brokers dos sites não permitia o funcionamento correto, logo que foi resolvido este problema com o broker local foi testado o sistema completo. Neste momento além de acionar o cilindro correto, ele deverá retornar quando a peça acionar o fim de curso. Os testes foram um sucesso como podem ser visto nos vídeos que se encontram no GitHub (https://github.com/VRochoso/TCC_MQTT_para_sistemas_discretos/tree/main)

Link para o arquivo compactado com o video do funcionamento da tela manual do aplicativo:

- https://github.com/VRochoso/TCC_MQTT_para_sistemas_discretos/blob/main/manual.rar

Link para o arquivo o video de funcionamento da tela automática do aplicativo:

- https://github.com/VRochoso/TCC_MQTT_para_sistemas_discretos/blob/main/automatico_compactado.mp4

5 CONCLUSÃO

Tanto quanto a engenharia de produção a engenharia mecatrônica está atrelada a indústria, a automação industrial e a indústria 4.0 é uma realidade, com o avanço do estudo nessa área poderemos diminuir o custo de produção, poderemos deixar os processos industriais cada vez mais rápidos e independentes, melhorar a qualidade e a quantidade na fabricação dos produtos.

Com este projeto podemos concluir que já é possível mudar a forma de controle das fabricas, o protocolo MQTT se mostrou muito eficiente em todos os quesitos, velocidade de comunicação, segurança, versatilidade, sendo possível ser implementado de diversas formas, a lógica da comparação de peças foi implementada em C++ para o ESP atuador, e implementado em Java no aplicativo e ambas as linguagens funcionaram perfeitamente, e sendo possível ser implementado também em Python direto no Raspberry.

O nível de processamento dos controladores se mostrou capazes de trabalhar em nível industrial, podendo sim ser uma substituição ao CLP, mesmo com interferências e ruídos com o nível da tecnologia de rede (5G, MESH) que temos hoje é possível ter o sinal necessário para os sensores e atuadores trabalharem no chão de fábrica, a diminuição de custos com cabos para transporte de dados é uma realidade nessa aplicação, sendo necessário cabeamento apenas para alimentação, e com tensão baixa de 12V.

6. BIBLIOGRAFIA

AMAZON WEB SERVICES. MQTT: O que é MQTT? [S.l.], [202-]. Disponível em <https://aws.amazon.com/pt/what-is/mqtt/>. Acesso em: 09 abr. 2024.

ENGPROCESS. Disponível em < https://engprocess.com.br/mqtt-broker/>. Acessado em: 09 abr. 2024.

BROKER, #58: Introdução ao MQTT e instalação de um broker MQTT. Disponível em <https://www.youtube.com/watch?v=71NwPbcTyV0&ab_channel=LNPBR>. Acessado em: 09 abr. 2024.

ALURA. NodeMcu e Esp8266: medindo e publicando dados com MQTT. Disponível em <https://cursos.alura.com.br/course/iot-com-nodemcu/task/27300>. Acessado em: 09 abr. 2024.

UDEMY. Curso de ESP8266. Disponível em <https://www.udemy.com/course/curso-de-esp8266/learn/lecture/8256910#overview> Acessado em: 09 abr. 2024.

ELETROGATE. Automação residencial com broker MQTT local com Raspberry Pi. [S.l.], [202-]. Disponível em <https://blog.eletrogate.com/automacao-residencial-com-broker-mqtt-local-com-raspberry-pi/>. Acesso em: 09 abr. 2024.

EMQX, Disponivel em <https://www.emqx.com/en/blog/what-is-the-mqtt-protocol>. Acessado em: 09 abr. 2024.

ESP32, Disponível em < https://pdf1.alldatasheet.com/datasheet-pdf/view/1148023/ESPRESSIF/ESP32.html>. Acessado em: 09 abr. 2024.

SOUZA, R. et al. Internet of Things (IoT) and its Applications in the Industry. In: INTERNATIONAL CONFERENCE ON TECHNOLOGICAL INNOVATION FOR INDUSTRIAL DEVELOPMENT, 3., 2022, São Paulo. Anais... São Paulo: Springer, 2022. p. 012019. DOI: 10.1088/1742-6596/2023/1/012019. Disponível em <https://iopscience.iop.org/article/10.1088/1742-6596/2023/1/012019/pdf>. Acessado em: 09 abr. 2024.

SANTOS, P. et al. Emerging Trends in IoT Security: A Review. In: INTERNATIONAL CONFERENCE ON TECHNOLOGICAL INNOVATION FOR INDUSTRIAL DEVELOPMENT, 2., 2020, São Paulo. Anais... São Paulo: Springer, 2020. p. 012044. DOI: 10.1088/1742-6596/2020/1/012044. Disponível em <https://iopscience.iop.org/article/10.1088/1742-6596/2020/1/012044/pdf>. Acessado em: 09 abr. 2024.

SAP. Disponível em < https://www.sap.com/brazil/products/scm/industry-4-0/what-is-iiot.html>. Acessado em: 09 abr. 2024.

HIVEMQ, Disponível em <https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt- quality-of-service-levels/>. Acessado em: 28 abr. 2024.

SHADOW, M. Nightmare. The Pass and Phantom Studios: Warner Bros, 2010. Faixa 6. CD.

APÊNDICE A – Código da ESP32_SENSOR_IN

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

/** DEFINITION OF WIFI CONECT
 *  TIPO  | WiFi
 *  LOCAL | LOCAL
 *  UFU1  | UFU-INSTITUCIONAL
 *  UFU2  | eduroam
*/
//#define LOCAL
#define UFU1
//#define UFU2

/* DEFINITION OF WIFI ACCESS POINT VARIABLES */
#if defined(LOCAL)
  #define SSID "rocha"
  #define EAP_PASSWORD "12345678"
#elif defined(UFU1)
  #define SSID "UFU-Institucional"
  #define EAP_ANONYMOUS_IDENTITY "anonymous@ufu.br" // anonymous@example.com,
or you can use also nickname@example.com
  #define EAP_IDENTITY "rochoso@ufu.br"         // nickname@example.com, at
some organizations should work nickname only without realm, but it is not
recommended
  #define EAP_PASSWORD "xxxxxx"
#elif defined(UFU2)
  #define SSID "eduroam"
  #define EAP_ANONYMOUS_IDENTITY "anonymous@ufu.br" // anonymous@example.com,
or you can use also nickname@example.com
  #define EAP_IDENTITY "rochoso@ufu.br"         // nickname@example.com, at
some organizations should work nickname only without realm, but it is not
recommended
  #define EAP_PASSWORD " xxxxxx"
#endif

/* DECLARATION OF MQTT SERVER VARIABLES */
const char* mqttserver = "10.14.48.104";
const uint16_t mqttport = 1883;
const char* mqttUser = "rocha";
const char* mqttpsswrd = "rocha";

/* DECLARATION OF TOPICS VARIABLES */
const char* topsenIN = "/sensor/peça";

/* DECLARATION OF A CLASS VARIABLE */
WiFiClient wifiClient;
```

```
PubSubClient MQTT(wifiClient);
TaskHandle_t Task1;

/* DEFINITION OF GPIO PINES VARIABLES */
#define sP 33
#define sM 25
#define sG 12
#define sMET 26
#define sCONT 27

/* DEFINITION OF A CONSTANT VARIABLES */
#define MSG_BUFFER_SIZE (24)
#define WAIT_CAP 1

/* DECLARATION OF GLOBAL VARIABLES */
bool clesP = 0;
bool clesM = 0;
bool clesG = 0;
bool clesMET = 0;
bool clesCONT = 0;
bool flag = false;
unsigned long timercap = 0;
char msg[MSG_BUFFER_SIZE];

/* DECLARATION OF FUNCTIONS WITH DEAFULT INPUT VALUES */
void conectarWiFi(int timeout = 30);
void IP_print(IPAddress localip = WiFi.localIP());

/* SETUP FUNCTION FOR SETUP THE PROGRAM INITIALIZATION */
void setup() {
  Serial.begin(115200);
  conectarWiFi();

  // Setup of Server and Subscribe
  MQTT.setServer(mqttserver, mqttport);
  MQTT.setCallback(callback);

  pinMode(sM, INPUT_PULLUP);
  pinMode(sP, INPUT_PULLUP);
  pinMode(sG, INPUT_PULLUP);
  pinMode(sMET, INPUT_PULLUP);
  pinMode(sCONT, INPUT_PULLUP);

  // Setup a Task using the free core of ESP32
  int CORE_ID = int(!xPortGetCoreID());
  xTaskCreatePinnedToCore(
    Task1code, // Task function
    "Task1",   // Name of the Task to atach
    100000,     // Size to alocated to Task (Word or Byte)
```

```
    NULL,      // Task argument to be passed (void*)
    0,         // Priority of the Task [0 a 25]
    &Task1,    // Task ID
    CORE_ID); // Task core ID (0 ou 1)
  delay(500);
}

/* TASK FUNCTION USING FREE CORE OF ESP32 */
void Task1code(void* pvParameters) {
  Serial.print("Conexões MQTT rodando no nucleo ");
  Serial.println(xPortGetCoreID());

  for (;;) {
    if (msg != "") {
      if (!MQTT.connected()) conectarMQTT();

      MQTT.loop();
    }
  }
}

/* LOOP FUNCTION TO RUNNING PROGRAM */
void loop() {
  delay(10);

  if (!clesM) clesM = digitalRead(sM);
  if (!clesP) clesP = digitalRead(sP);
  if (!clesG) clesG = digitalRead(sG);
  if (!clesMET && !flag) clesMET = digitalRead(sMET);
  if (!clesCONT && !flag) clesCONT = digitalRead(sCONT);

  if (!flag) {
    timercap = millis();
    if (clesCONT) flag = true;
  } else if (int(millis() - timercap) >= WAIT_CAP*1000) flag = false;

  //printf("sensor pequeno: %d\tsensor medio: %d\tsensor grande: %d\tsensor
capacitivo: %d\tsensor metalico: %d\n", clesP, clesM, clesG, clesCONT,
clesMET);

  if(clesP && clesCONT){
    printf("sensor pequeno: %d\tsensor medio: %d\tsensor grande: %d\tsensor
capacitivo: %d\tsensor metalico: %d\n", clesP, clesM, clesG, clesCONT,
clesMET);
    if (!clesM && !clesG && !clesMET) snprintf (msg, MSG_BUFFER_SIZE, "Peça
Pequena");
    if (clesM && !clesG && !clesMET) snprintf (msg, MSG_BUFFER_SIZE, "Peça
Média");
```

```
    if (clesM && clesG && !clesMET) snprintf (msg, MSG_BUFFER_SIZE, "Peça
Grande");
    if (!clesM && !clesG && clesMET) snprintf (msg, MSG_BUFFER_SIZE, "Peça
Pequena Metallica");
    if (clesM && !clesG && clesMET) snprintf (msg, MSG_BUFFER_SIZE, "Peça
Média Metallica");
    if (clesM && clesG && clesMET) snprintf (msg, MSG_BUFFER_SIZE, "Peça
Grande Metallica");

    printf("%s\n", msg);

    MQTT.publish(topsenIN, msg, 2);
    clesP = 0;
    clesM = 0;
    clesG = 0;
    clesMET = 0;
    clesCONT = 0;
  }
}

/* CALLBACK FUNCTION FOR SUBSCRIBE TOPICS */
void callback(char* topic, byte* payload, unsigned int length) {
  String texto = "";
  for (int i = 0; i < length; i++) texto += (char)payload[i];

  if (String(msg) != texto && String(msg) != "") MQTT.publish(topsenIN, msg,
2);
  else if (String(msg) != "") snprintf (msg, MSG_BUFFER_SIZE, "");
}

/* WIFI FUNCTION FOR CONECT TO SSID DEFINED */
void conectarWiFi(int timeout) {
  unsigned long inicio = millis();

  Serial.print(F("Conectando ao WiFi "));
  Serial.print(SSID);
  Serial.print(F(", aguarde..."));
  #ifdef EAP_ANONYMOUS_IDENTITY
    WiFi.disconnect(true);
    WiFi.mode(WIFI_STA);
    WiFi.begin(SSID, WPA2_AUTH_PEAP, EAP_ANONYMOUS_IDENTITY, EAP_IDENTITY,
EAP_PASSWORD);
  #else
    WiFi.mode(WIFI_STA);
    WiFi.begin(SSID, EAP_PASSWORD);
  #endif

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
```

```
    Serial.print(F("."));
    int timer = (millis() - inicio)/1000;
    if ((WiFi.status() == WL_CONNECT_FAILED || WiFi.status() ==
WL_CONNECTION_LOST) && timer <= timeout) {
      #ifdef EAP_ANONYMOUS_IDENTITY
        WiFi.disconnect(true);
        WiFi.mode(WIFI_STA);
        WiFi.begin(SSID, WPA2_AUTH_PEAP, EAP_ANONYMOUS_IDENTITY, EAP_IDENTITY,
EAP_PASSWORD);
      #else
        WiFi.mode(WIFI_STA);
        WiFi.begin(SSID, EAP_PASSWORD);
      #endif
    } else if (timer > timeout) ESP.restart();
  }

  IP_print();
}

/* IP PRINT FUNCTION */
void IP_print(IPAddress localip) {
  byte mac[6];
  String IP = "";
  for (int i=0; i<4; i++)
    IP += i ? "." + String(localip[i]) : String(localip[i]);
  int IP_len = IP.length();

  Serial.println(F("\n\n|*********************************************|"));
  Serial.println(F("|\t\tWiFi is connected!\t\t |"));

  Serial.print(F("|"));
  for (int i = 0; i < int((37 - IP_len)/2); i++) Serial.print(F(" "));
  Serial.print(F("IP Address: "));
  Serial.print(WiFi.localIP()); //print LAN IP
  for (int i = 0; i < int((36 - IP_len)/2); i++) Serial.print(F(" "));
  Serial.print(F("|\n|\t MAC Address - "));

  WiFi.macAddress(mac);
  String texto = String(mac[0], HEX) + ":" + String(mac[1], HEX) + ":" +
String(mac[2], HEX) + ":" + String(mac[3], HEX) + ":" + String(mac[4], HEX) +
":" + String(mac[5], HEX);
  texto.toUpperCase();

  Serial.print(texto);
  Serial.println(F("\t
|\n|*********************************************|\n"));
}

/* MQTT FUNCTION FOR CONECT TO SERVER */
```

```
void conectarMQTT() {
  while (!MQTT.connected()) {
    if (MQTT.connect("ESP32-PEÇA")){
      MQTT.subscribe(topsenIN);
    }
  }
}
```

APÊNDICE B – Código da ESP32_ATUADOR

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

/* DEFINITION OF ACTION VARIABLES */
#define avanco 0
#define recuo 1

/** DEFINITION OF WIFI CONECT
 *  TIPO  | WiFi
 *  LOCAL | LOCAL
 *  UFU1  | UFU-INSTITUCIONAL
 *  UFU2  | eduroam
*/
// #define LOCAL
#define UFU1
//#define UFU2

/* DEFINITION OF WIFI ACCESS POINT VARIABLES */
#if defined(LOCAL)
  #define SSID "rocha"
  #define EAP_PASSWORD "12345678"
#elif defined(UFU1)
  #define SSID "UFU-Institucional"
  #define EAP_ANONYMOUS_IDENTITY "anonymous@ufu.br" // anonymous@example.com,
or you can use also nickname@example.com
  #define EAP_IDENTITY "rochoso@ufu.br"         // nickname@example.com, at
some organizations should work nickname only without realm, but it is not
recommended
  #define EAP_PASSWORD "xxxxxx"
#elif defined(UFU2)
  #define SSID "eduroam"
  #define EAP_ANONYMOUS_IDENTITY "anonymous@ufu.br" // anonymous@example.com,
or you can use also nickname@example.com
  #define EAP_IDENTITY "rochoso@ufu.br"         // nickname@example.com, at
some organizations should work nickname only without realm, but it is not
recommended
  #define EAP_PASSWORD "xxxxxx"
#endif
```

```
/* DECLARATION OF MQTT SERVER VARIABLES */
const char* mqttserver = "10.14.48.104";
const uint16_t mqttport = 1883;
const char* mqttUser = "rocha";
const char* mqttpsswrd = "rocha";

/* DECLARATION OF TOPICS VARIABLES */
const char* topvalv = "/valv/cilindroacao";


/* DECLARATION OF A CLASS VARIABLE */
WiFiClient espClient;
PubSubClient MQTT(espClient);
TaskHandle_t Task1;

/* DEFINITION OF GPIO PINES VARIABLES */
#define valvS1 26
#define valvS2 25
#define valvS3a 33
#define valvS3r 32

/* DECLARATION OF GLOBAL VARIABLES */
char statusacao = {0};
bool statusvalv1 = 0;
bool statusvalv2 = 0;
bool statusvalv3 = 0;


/* DECLARATION OF FUNCTIONS WITH DEAFULT INPUT VALUES */
void conectarWiFi(int timeout = 30);
void IP_print(IPAddress localip = WiFi.localIP());

/* SETUP FUNCTION FOR SETUP THE PROGRAM INITIALIZATION */
void setup() {
  Serial.begin(115200);
  conectarWiFi();

  // Setup of Server and Subscribe
  MQTT.setServer(mqttserver, mqttport);
  MQTT.setCallback(callback);

  pinMode(valvS1, OUTPUT);
  pinMode(valvS2, OUTPUT);
  pinMode(valvS3a, OUTPUT);
  pinMode(valvS3r, OUTPUT);

  Valv1(recuo);
  Valv2(recuo);
```

```
  Valv3(recuo);

  // Setup a Task using the free core of ESP32
  int CORE_ID = int(!xPortGetCoreID());
  xTaskCreatePinnedToCore(
    Task1code, // Task function
    "Task1",   // Name of the Task to atach
    10000,     // Size to alocated to Task (Word or Byte)
    NULL,      // Task argument to be passed (void*)
    0,         // Priority of the Task [0 a 25]
    &Task1,    // Task ID
    CORE_ID); // Task core ID (0 ou 1)
  delay(500);
}

/* TASK FUNCTION USING FREE CORE OF ESP32 */
void Task1code(void* pvParameters) {
  Serial.print("Conexões (Wi-Fi e MQTT) rodando no nucleo ");
  Serial.println(xPortGetCoreID());

  for (;;) {
    if (!MQTT.connected()) conectarMQTT();

    MQTT.loop();
  }
}

/* LOOP FUNCTION TO RUNNING PROGRAM */
void loop() { }

/* CALLBACK FUNCTION FOR SUBSCRIBE TOPICS */
void callback(char* topic, byte* payload, unsigned int length) {
  if (statusacao != (char)payload[0]) {
    Serial.print("Mensagem recebida no tópico: ");
    Serial.println(topic);

    statusacao = (char)payload[0];
    Serial.println("Conteúdo da mensagem: " + String(statusacao));

    if ((char)payload[0] == '1') Valv1(recuo);
    else if ((char)payload[0] == '2') Valv1(avanco);
    else if ((char)payload[0] == '3') Valv2(recuo);
    else if ((char)payload[0] == '4') Valv2(avanco);
    else if ((char)payload[0] == '5') Valv3(recuo);
    else if ((char)payload[0] == '6') Valv3(avanco);
  }
}

/* WIFI FUNCTION FOR CONECT TO SSID DEFINED */
```

```
void conectarWiFi(int timeout) {
  unsigned long inicio = millis();

  Serial.print(F("Conectando ao WiFi "));
  Serial.print(SSID);
  Serial.print(F(", aguarde..."));
  #ifdef EAP_ANONYMOUS_IDENTITY
    WiFi.disconnect(true);
    WiFi.mode(WIFI_STA);
    WiFi.begin(SSID, WPA2_AUTH_PEAP, EAP_ANONYMOUS_IDENTITY, EAP_IDENTITY,
EAP_PASSWORD);
  #else
    WiFi.mode(WIFI_STA);
    WiFi.begin(SSID, EAP_PASSWORD);
  #endif

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(F("."));
    int timer = (millis() - inicio)/1000;
    if ((WiFi.status() == WL_CONNECT_FAILED || WiFi.status() ==
WL_CONNECTION_LOST) && timer <= timeout) {
      #ifdef EAP_ANONYMOUS_IDENTITY
        WiFi.disconnect(true);
        WiFi.mode(WIFI_STA);
        WiFi.begin(SSID, WPA2_AUTH_PEAP, EAP_ANONYMOUS_IDENTITY, EAP_IDENTITY,
EAP_PASSWORD);
      #else
        WiFi.mode(WIFI_STA);
        WiFi.begin(SSID, EAP_PASSWORD);
      #endif
    } else if (timer > timeout) ESP.restart();
  }

  IP_print();
}

/* IP PRINT FUNCTION */
void IP_print(IPAddress localip) {
  byte mac[6];
  String IP = "";
  for (int i=0; i<4; i++)
    IP += i ? "." + String(localip[i]) : String(localip[i]);
  int IP_len = IP.length();

  Serial.println(F("\n\n|*********************************************|"));
  Serial.println(F("|\t\tWiFi is connected!\t\t |"));

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

```
  for (int i = 0; i < int((37 - IP_len)/2); i++) Serial.print(F(" "));
  Serial.print(F("IP Address: "));
  Serial.print(WiFi.localIP()); //print LAN IP
  for (int i = 0; i < int((36 - IP_len)/2); i++) Serial.print(F(" "));
  Serial.print(F("|\n|\t MAC Address - "));

  WiFi.macAddress(mac);
  String texto = String(mac[0], HEX) + ":" + String(mac[1], HEX) + ":" +
String(mac[2], HEX) + ":" + String(mac[3], HEX) + ":" + String(mac[4], HEX) +
":" + String(mac[5], HEX);
  texto.toUpperCase();

  Serial.print(texto);
  Serial.println(F("\t
|\n|***********************************************|\n"));
}

/* MQTT FUNCTION FOR CONECT TO SERVER */
void conectarMQTT() {
  while (!MQTT.connected()) {
    if (MQTT.connect("ESP32Client")){
      //publish();

      MQTT.subscribe(topvalv);
    }
  }
}

/* VALVE 1 FUNCTION FOR DRIVE THE VALVE */
void Valv1(bool status) {
  digitalWrite(valvS1, status);

  statusvalv1 = !status;
}

/* VALVE 2 FUNCTION FOR DRIVE THE VALVE */
void Valv2(bool status) {
  digitalWrite(valvS2, status);

  statusvalv2 = !status;
}

/* VALVE 3 FUNCTION FOR DRIVE THE VALVE */
void Valv3(bool status) {
  if (status) {
    digitalWrite(valvS3a, status);
    delay(10);
    digitalWrite(valvS3r, !status);
  } else {
```

```
    digitalWrite(valvS3r, !status);
    delay(10);
    digitalWrite(valvS3a, status);
  }

  statusvalv3 = !status;
}

/* PUBLISHER FUNCTION FOR RETURN VALVE STATUS */
void publish() {
  if (statusvalv1) MQTT.publish(topvalv1, "Avançado", 2);
  else if (!statusvalv1) MQTT.publish(topvalv1, "Recuado", 2);

  if (statusvalv2) MQTT.publish(topvalv2, "Avançado", 2);
  else if (!statusvalv2) MQTT.publish(topvalv2, "Recuado", 2);

  if (statusvalv3) MQTT.publish(topvalv3, "Avançado", 2);
  else if (!statusvalv3) MQTT.publish(topvalv3, "Recuado", 2);
}
```

APÊNDICE C – Código da ESP32_SENSOR_OUT

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

/** DEFINITION OF WIFI CONECT
 *  TIPO  | WiFi
 *  LOCAL | LOCAL
 *  UFU1  | UFU-INSTITUCIONAL
 *  UFU2  | eduroam
*/
//#define LOCAL
#define UFU1
//#define UFU2

/* DEFINITION OF WIFI ACCESS POINT VARIABLES */
#if defined(LOCAL)
  #define SSID "rocha"
  #define EAP_PASSWORD "12345678"
#elif defined(UFU1)
  #define SSID "UFU-Institucional"
  #define EAP_ANONYMOUS_IDENTITY "anonymous@ufu.br" // anonymous@example.com,
or you can use also nickname@example.com
  #define EAP_IDENTITY "rochoso@ufu.br"         // nickname@example.com, at
some organizations should work nickname only without realm, but it is not
recommended
  #define EAP_PASSWORD " xxxxxx "
#elif defined(UFU2)
```

```
  #define SSID "eduroam"
  #define EAP_ANONYMOUS_IDENTITY "anonymous@ufu.br" // anonymous@example.com,
or you can use also nickname@example.com
  #define EAP_IDENTITY "rochoso@ufu.br"         // nickname@example.com, at
some organizations should work nickname only without realm, but it is not
recommended
  #define EAP_PASSWORD " xxxxxx "
#endif

/* DECLARATION OF MQTT SERVER VARIABLES */
const char* mqttserver = "10.14.48.104";
const uint16_t mqttport = 1883;
const char* mqttUser = "rocha";
const char* mqttpsswrd = "rocha";

/* DECLARATION OF TOPICS VARIABLES */
const char* topsenOUT = "/valv/recuo";

/* DECLARATION OF A CLASS VARIABLE */
WiFiClient wifiClient;
PubSubClient MQTT(wifiClient);
TaskHandle_t Task1;

/* DEFINITION OF GPIO PINES VARIABLES */
#define s1 32
#define s2 33
#define s3 25
#define s4 26

/* DEFINITION OF A CONSTANT VARIABLES */
#define MSG_BUFFER_SIZE (17)

/* DECLARATION OF GLOBAL VARIABLES */
bool cles1 = 0;
bool cles2 = 0;
bool cles3 = 0;
bool cles4 = 0;
char msg[MSG_BUFFER_SIZE];

/* DECLARATION OF FUNCTIONS WITH DEAFULT INPUT VALUES */
void conectarWiFi(int timeout = 30);
void IP_print(IPAddress localip = WiFi.localIP());

/* SETUP FUNCTION FOR SETUP THE PROGRAM INITIALIZATION */
void setup() {
  Serial.begin(115200);
  conectarWiFi();

  // Setup of Server and Subscribe
```

```
  MQTT.setServer(mqttserver, mqttport);
  MQTT.setCallback(callback);

  pinMode(s2, INPUT_PULLUP);
  pinMode(s1, INPUT_PULLUP);
  pinMode(s3, INPUT_PULLUP);
  pinMode(s4, INPUT_PULLUP);

  // Setup a Task using the free core of ESP32
  int CORE_ID = int(!xPortGetCoreID());
  xTaskCreatePinnedToCore(
    Task1code, // Task function
    "Task1",   // Name of the Task to atach
    100000,     // Size to alocated to Task (Word or Byte)
    NULL,      // Task argument to be passed (void*)
    0,         // Priority of the Task [0 a 25]
    &Task1,    // Task ID
    CORE_ID); // Task core ID (0 ou 1)
  delay(500);
}

/* TASK FUNCTION USING FREE CORE OF ESP32 */
void Task1code(void* pvParameters) {
  Serial.print("Conexões MQTT rodando no nucleo ");
  Serial.println(xPortGetCoreID());

  for (;;) {
    if (msg != "") {
      if (!MQTT.connected()) conectarMQTT();

      MQTT.loop();
    }
  }
}

/* LOOP FUNCTION TO RUNNING PROGRAM */
void loop() {
  delay(10);

  if (!cles1) cles1 = digitalRead(s1);
  if (!cles2) cles2 = digitalRead(s2);
  if (!cles3) cles3 = digitalRead(s3);
  if (!cles4) cles4 = digitalRead(s4);

  //printf("Rampa 1: %d\tRampa 2: %d\tRampa 3: %d\tRampa 4: %d\n", cles1,
cles2, cles3, cles4);

  if(cles1 || cles2 || cles3 || cles4) {
```

```
    printf("Rampa 1: %d\tRampa 2: %d\tRampa 3: %d\tRampa 4: %d\n", cles1,
cles2, cles3, cles4);
    if (cles1) snprintf (msg, MSG_BUFFER_SIZE, "Peça na saida 1");
    else if (cles2) snprintf (msg, MSG_BUFFER_SIZE, "Peça na saida 2");
    else if (cles3) snprintf (msg, MSG_BUFFER_SIZE, "Peça na saida 3");
    else if (cles4) snprintf (msg, MSG_BUFFER_SIZE, "Peça na saida 4");

    printf("%s\n", msg);

    MQTT.publish(topsenOUT, msg, 2);
    cles1 = 0;
    cles2 = 0;
    cles3 = 0;
    cles4 = 0;
  }
}

/* CALLBACK FUNCTION FOR SUBSCRIBE TOPICS */
void callback(char* topic, byte* payload, unsigned int length) {
  String texto = "";
  for (int i = 0; i < length; i++) texto += (char)payload[i];

  if (String(msg) != texto && String(msg) != "") MQTT.publish(topsenOUT, msg,
2);
  else if (String(msg) != "") snprintf (msg, MSG_BUFFER_SIZE, "");
}

/* WIFI FUNCTION FOR CONECT TO SSID DEFINED */
void conectarWiFi(int timeout) {
  unsigned long inicio = millis();

  Serial.print(F("Conectando ao WiFi "));
  Serial.print(SSID);
  Serial.print(F(", aguarde..."));
  #ifdef EAP_ANONYMOUS_IDENTITY
    WiFi.disconnect(true);
    WiFi.mode(WIFI_STA);
    WiFi.begin(SSID, WPA2_AUTH_PEAP, EAP_ANONYMOUS_IDENTITY, EAP_IDENTITY,
EAP_PASSWORD);
  #else
    WiFi.mode(WIFI_STA);
    WiFi.begin(SSID, EAP_PASSWORD);
  #endif

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(F("."));
    int timer = (millis() - inicio)/1000;
```

```
    if ((WiFi.status() == WL_CONNECT_FAILED || WiFi.status() ==
WL_CONNECTION_LOST) && timer <= timeout) {
      #ifdef EAP_ANONYMOUS_IDENTITY
        WiFi.disconnect(true);
        WiFi.mode(WIFI_STA);
        WiFi.begin(SSID, WPA2_AUTH_PEAP, EAP_ANONYMOUS_IDENTITY, EAP_IDENTITY,
EAP_PASSWORD);
      #else
        WiFi.mode(WIFI_STA);
        WiFi.begin(SSID, EAP_PASSWORD);
      #endif
    } else if (timer > timeout) ESP.restart();
  }

  IP_print();
}

/* IP PRINT FUNCTION */
void IP_print(IPAddress localip) {
  byte mac[6];
  String IP = "";
  for (int i=0; i<4; i++)
    IP += i ? "." + String(localip[i]) : String(localip[i]);
  int IP_len = IP.length();

  Serial.println(F("\n\n|*********************************************|"));
  Serial.println(F("|\t\tWiFi is connected!\t\t |"));

  Serial.print(F("|"));
  for (int i = 0; i < int((37 - IP_len)/2); i++) Serial.print(F(" "));
  Serial.print(F("IP Address: "));
  Serial.print(WiFi.localIP()); //print LAN IP
  for (int i = 0; i < int((36 - IP_len)/2); i++) Serial.print(F(" "));
  Serial.print(F("|\n|\t MAC Address - "));

  WiFi.macAddress(mac);
  String texto = String(mac[0], HEX) + ":" + String(mac[1], HEX) + ":" +
String(mac[2], HEX) + ":" + String(mac[3], HEX) + ":" + String(mac[4], HEX) +
":" + String(mac[5], HEX);
  texto.toUpperCase();

  Serial.print(texto);
  Serial.println(F("\t
|\n|*********************************************|\n"));
}

/* MQTT FUNCTION FOR CONECT TO SERVER */
void conectarMQTT() {
  while (!MQTT.connected()) {
```

```
    if (MQTT.connect("ESP32-OUT")){
      MQTT.subscribe(topsenOUT);
    }
  }
}
```

APÊNDICE D – Código backend da tela inicial do aplicativo

```
package com.somsakelect.android.rochapp;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.view.WindowManager;

public class MainActivity extends AppCompatActivity {

    @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

        // Botão Manual
        findViewById(R.id.manual_btn).setOnClickListener(v -> {
            // Crie um Intent para iniciar ManualControl
            Intent ManualControl = new Intent(MainActivity.this,
ManualControl.class);
            // Inicie a DoisAtuad com o Intent
            startActivity(ManualControl);
        });

        //Botão Automático
        findViewById(R.id.automatico_btn).setOnClickListener(v -> {
            // Crie um Intent para iniciar AutoControl
            Intent AutoControl = new Intent(MainActivity.this,
AutoControl.class);
            // Inicie a ControlAuto com o Intent
            startActivity(AutoControl);
        });
    }

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

```
    }
}
```

APÊNDICE E – Código backend da tela manual do aplicativo

```
package com.somsakelect.android.rochapp;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.content.ContextCompat;

import android.content.Intent;
import android.content.res.ColorStateList;
import android.graphics.Color;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.view.ViewGroup;
import android.view.WindowManager;
import android.widget.Button;
import android.widget.EditText;
import android.widget.LinearLayout;
import android.widget.TextView;
import android.widget.Toast;

import com.somsakelect.android.mqtt.MqttAndroidClient;

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;

import java.nio.charset.StandardCharsets;
import java.util.Arrays;

public class ManualControl extends AppCompatActivity {
    private static final int ATENCAO = 1;
    private static final int SUCESSO = 2;
    private static final int ERRO = 3;
    private MqttAndroidClient mqtt;

    private TextView statusConexaoManual;
    private LinearLayout btnConectarManual;
    private Button btnCilindro1, btnCilindro2, btnCilindro3;
    private static final String TAG = "ManualControl";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setTheme(R.style.Theme_MyMQTT); // App inicia com o tema Splash.
Aqui altera para o tema principal
        setContentView(R.layout.activity_manual_control);

getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON); //
Manter a tela ligada para evitar desconexão com o broker

        // Recupere o valor da string (HOST) do Intent
```

```
        final String MQTT_HOST = "10.14.48.104";
        final int MQTT_PORT = 1883;
        final String MQTT_URL = "tcp://" + MQTT_HOST + ":" + MQTT_PORT;
        final String MQTT_ID = MqttClient.generateClientId();

        statusConexaoManual = findViewById(R.id.tv_statusConexaoManual);
        btnConectarManual = findViewById(R.id.ll_btnConectar);

        //MQTT
        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);
                statusConexaoManual.setText(reconnect ? "Reconectando..." :
"Conectado!");
                if (mqtt.isConnected()) {
                    String tsx = "CONECTADO!";
                    statusConexaoManual.setText(tsx);
                    btnConectarManual.setVisibility(View.INVISIBLE);
                    // Subscrever tópicos aqui
                    // subscribe("/status/espSensores"); // Recebe o status
do ESP32 dos sensores de fim de curso
                }
            }

            @Override
            public void connectionLost(Throwable cause) {
                if (cause != null) {
                    Log.e(TAG, "Conexão MQTT perdida..." +
cause.getMessage());
                    String st = "Conexão perdida! " + cause.getMessage();
                    statusConexaoManual.setText(st);
                    btnConectarManual.setVisibility(View.VISIBLE);
                }
            }

            @Override
            public void messageArrived(String topic, MqttMessage message) {
                // Tratamento das mensagens nos tópicos subscritos
            }

            @Override
            public void deliveryComplete(IMqttDeliveryToken token) {
                Log.w(TAG, "Publish success...");
            }
        });

        // BOTÃO CILINDRO 1
        btnCilindro1 = findViewById(R.id.avanCilin1_btn);
        btnCilindro1.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                // Verifica o texto atual do botão
                String textoAtual1 = btnCilindro1.getText().toString();
                // Verifica conexão com broker
                if (!mqtt.isConnected()) {
                    showToast(ERRO,"Conecte-se ao broker e tente
novamente!");
                    return;
```

```
                }
                // Realiza o Publish e altera texto e cor do botão com base
no atual
                if (textoAtual1.equals("AVANÇAR CIL 1")) {
                    AvanCilind1();
                    showToast(SUCESSO,"Avançando cilindro 1...");
                    btnCilindro1.setText("RECUAR CIL 1");

btnCilindro1.setBackgroundTintList(ColorStateList.valueOf(Color.parseColor(
"#a8323e")));
                } else if (textoAtual1.equals("RECUAR CIL 1")) {
                    RecuaCilind1();
                    showToast(SUCESSO,"Recuando cilindro 1...");
                    btnCilindro1.setText("AVANÇAR CIL 1");

btnCilindro1.setBackgroundTintList(ColorStateList.valueOf(Color.parseColor(
"#4A148C")));
                }
            }
        });

        // BOTÃO CILINDRO 2
        btnCilindro2 = findViewById(R.id.avanCilin2_btn);
        btnCilindro2.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                // Verifica o texto atual do botão
                String textoAtual2 = btnCilindro2.getText().toString();
                // Verifica conexão com broker
                if (!mqtt.isConnected()) {
                    showToast(ERRO,"Conecte-se ao broker e tente
novamente!");
                    return;
                }
                // Realiza o Publish e altera texto e cor do botão com base
no atual
                if (textoAtual2.equals("AVANÇAR CIL 2")) {
                    AvanCilind2();
                    showToast(SUCESSO,"Avançando cilindro 2...");
                    btnCilindro2.setText("RECUAR CIL 2");

btnCilindro2.setBackgroundTintList(ColorStateList.valueOf(Color.parseColor(
"#a8323e")));
                } else if (textoAtual2.equals("RECUAR CIL 2")) {
                    RecuaCilind2();
                    showToast(SUCESSO,"Recuando cilindro 2...");
                    btnCilindro2.setText("AVANÇAR CIL 2");

btnCilindro2.setBackgroundTintList(ColorStateList.valueOf(Color.parseColor(
"#4A148C")));
                }
            }
        });

        // BOTÃO CILINDRO 3
        btnCilindro3 = findViewById(R.id.avanCilin3_btn);
        btnCilindro3.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                // Verifica o texto atual do botão
                String textoAtual3 = btnCilindro3.getText().toString();
```

```
                // Verifica conexão com broker
                if (!mqtt.isConnected()) {
                    showToast(ERRO,"Conecte-se ao broker e tente
novamente!");
                    return;
                }
                // Realiza o Publish e altera texto e cor do botão com base
no atual
                if (textoAtual3.equals("AVANÇAR CIL 3")) {
                    AvanCilind3();
                    showToast(SUCESSO,"Avançando cilindro 3...");
                    btnCilindro3.setText("RECUAR CIL 3");

btnCilindro3.setBackgroundTintList(ColorStateList.valueOf(Color.parseColor(
"#a8323e")));
                } else if (textoAtual3.equals("RECUAR CIL 3")) {
                    RecuaCilind3();
                    showToast(SUCESSO,"Recuando cilindro 3...");
                    btnCilindro3.setText("AVANÇAR CIL 3");

btnCilindro3.setBackgroundTintList(ColorStateList.valueOf(Color.parseColor(
"#4A148C")));
                }
            }
        });

        //Botão Conectar
        findViewById(R.id.conectar_btn).setOnClickListener(v ->
connectMQTT());

        //Botão Voltar
        findViewById(R.id.voltar_btn).setOnClickListener(v -> {
            // Crie um Intent para iniciar MainActivity
            Intent tela_inicial = new Intent(ManualControl.this,
MainActivity.class);
            disconnectMQTT();
            // Inicie a TresAtuad com o Intent
            startActivity(tela_inicial);
        });

        //Try connect
        connectMQTT();
    }
    // Função para desabilitar "Voltar" via botão do hardware
    @Override
    public void onBackPressed() {
        // Não faz nada ao clicar no botão voltar do hardware
    }
    // Funções para ativar modo tela cheia ao iniciar o aplicativo
    @Override
    public void onWindowFocusChanged(boolean hasFocus) {
        super.onWindowFocusChanged(hasFocus);
        if (hasFocus) {
            hideSystemUI();
        }
    }

    private void hideSystemUI() {
        View decorView = getWindow().getDecorView();
        decorView.setSystemUiVisibility(
                View.SYSTEM_UI_FLAG_IMMERSIVE
```

```
                        | View.SYSTEM_UI_FLAG_LAYOUT_STABLE
                        | View.SYSTEM_UI_FLAG_LAYOUT_HIDE_NAVIGATION
                        | View.SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN
                        | View.SYSTEM_UI_FLAG_HIDE_NAVIGATION
                        | View.SYSTEM_UI_FLAG_FULLSCREEN);
    }

    public void showToast(int type, String message) {

        switch (type) {
            case ATENCAO:
                ViewGroup view = findViewById(R.id.container_toast);
                View v = getLayoutInflater().inflate(R.layout.custom_toast,
view);

                v.setBackground(ContextCompat.getDrawable(this,
R.drawable.toast_atencao));

                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();
                break;
            case SUCESSO:
                ViewGroup view2 =
findViewById(R.id.container_toast_sucesso);
                View v2 =
getLayoutInflater().inflate(R.layout.success_toast, view2);

                v2.setBackground(ContextCompat.getDrawable(this,
R.drawable.toast_sucesso));

                TextView txtMessage2 = v2.findViewById(R.id.txt_toast);
                txtMessage2.setText(message);

                Toast toast2 = new Toast(this);
                toast2.setView(v2);
                toast2.setDuration(Toast.LENGTH_SHORT);
                toast2.show();
                break;
            case ERRO:
                ViewGroup view3 = findViewById(R.id.container_toast_erro);
                View v3 = getLayoutInflater().inflate(R.layout.error_toast,
view3);

                v3.setBackground(ContextCompat.getDrawable(this,
R.drawable.toast_erro));

                TextView txtMessage3 = v3.findViewById(R.id.txt_toast);
                txtMessage3.setText(message);

                Toast toast3 = new Toast(this);
                toast3.setView(v3);
                toast3.setDuration(Toast.LENGTH_SHORT);
                toast3.show();
                break;

            default:
```

```
                throw new IllegalStateException("Erro de tipo " + message);
            }
        }

        // Função para salvar os campos de input (EditText) como string
        private String getEnter(EditText e) {
            return e.getText().toString();
        }

        private void connectMQTT() {
            if (mqtt.isConnected()) {
                showToast(ATENCAO,"Broker já conectado!");
                statusConexaoManual.setText(R.string.conectado);
                btnConectarManual.setVisibility(View.INVISIBLE);
                return;
            }
            Log.w(TAG, "Conectando ao broker MQTT...");
            statusConexaoManual.setText(R.string.conectando);

            // Recupere o valor da string (USERNAME e PASSWORD) do Intent
            //final String MQTT_USERNAME = "rocha";
           // final String MQTT_PASSWORD = "rocha";

            //Set option
            MqttConnectOptions options = new MqttConnectOptions();
            //options.setUserName(MQTT_USERNAME);
            //options.setPassword(MQTT_PASSWORD.toCharArray());
            options.setAutomaticReconnect(true);
            options.setCleanSession(true);
            try {
                IMqttToken token = mqtt.connect(options);
                token.setActionCallback(new IMqttActionListener() {
                    @Override
                    public void onSuccess(IMqttToken asyncActionToken) {
                        Log.w(TAG, "Connect success!");
                        //Subscrever tópicos aqui
                        //subscribe("/status/espSensores"); // Recebe o status
do ESP32 dos sensores de fim de curso
                    }

                    @Override
                    public void onFailure(IMqttToken asyncActionToken,
Throwable exception) {
                        Log.e(TAG, "Error..." + exception.getMessage());
                        String tsx = "Falha na conexão... " +
exception.getMessage();
                        statusConexaoManual.setText(tsx);
                        btnConectarManual.setVisibility(View.VISIBLE);
                    }
                });
            } catch (MqttException e) {
                e.printStackTrace();

                String tsx = "Connect MqttException: " + e.getMessage();
                statusConexaoManual.setText(tsx);
                btnConectarManual.setVisibility(View.VISIBLE);
            }
        }

        private void disconnectMQTT() {
            Log.d(TAG, "Disconnecting MQTT server...");
```

```
        try {
            IMqttToken token = mqtt.disconnect();
            token.setActionCallback(new IMqttActionListener() {
                @Override
                public void onSuccess(IMqttToken asyncActionToken) {
                    Log.w(TAG, "Disconnect success...");
                }

                @Override
                public void onFailure(IMqttToken asyncActionToken,
Throwable exception) {
                    Log.e(TAG, "Disconnect failed...");
                }
            });
        } catch (MqttException e) {
            e.printStackTrace();
            Log.e(TAG, "Error..." + e.getMessage());
        }
    }

    private void subscribe(@NonNull String topic) {
        //Connect
        if (!mqtt.isConnected()) {
            return;
        }

        try {
            //Set
            IMqttToken token = mqtt.subscribe(topic, 0);
            //Check result
            token.setActionCallback(new IMqttActionListener() {
                @Override
                public void onSuccess(IMqttToken asyncActionToken) {
                    Log.w(TAG, "Subscribed..."
                            +
Arrays.toString(asyncActionToken.getTopics()));
                }

                @Override
                public void onFailure(IMqttToken asyncActionToken,
Throwable exception) {
                    Log.e(TAG, "Subscribe failed..."
                            +
Arrays.toString(asyncActionToken.getTopics()));
                }
            });
        } catch (MqttException e) {
            e.printStackTrace();
        }
    }

    public void AvanCilind1() {
        String topic = "/valv/cilindroacao";
        String payload = "2"; // Avança o cilindro 1
        try {
            byte[] encodedPayload =
payload.getBytes(StandardCharsets.UTF_8);
            MqttMessage message = new MqttMessage(encodedPayload);
            mqtt.publish(topic, message);
        } catch (MqttException e) {
            e.printStackTrace();
```

```
        }
    }

    public void RecuaCilind1() {
        String topic  =  "/valv/cilindroacao";
        String payload = "1"; // Recua o cilindro 1
        try {
            byte[] encodedPayload =
payload.getBytes(StandardCharsets.UTF_8);
            MqttMessage message = new MqttMessage(encodedPayload);
            mqtt.publish(topic, message);
        } catch (MqttException e) {
            e.printStackTrace();
        }
    }

    public void AvanCilind2() {
        String topic = "/valv/cilindroacao";
        String payload = "4"; // Avança o cilindro 2
        try {
            byte[] encodedPayload =
payload.getBytes(StandardCharsets.UTF_8);
            MqttMessage message = new MqttMessage(encodedPayload);
            mqtt.publish(topic, message);
        } catch (MqttException e) {
            e.printStackTrace();
        }
    }

    public void RecuaCilind2() {
        String topic  =  "/valv/cilindroacao";
        String payload = "3"; // Recua o cilindro 2
        try {
            byte[] encodedPayload =
payload.getBytes(StandardCharsets.UTF_8);
            MqttMessage message = new MqttMessage(encodedPayload);
            mqtt.publish(topic, message);
        } catch (MqttException e) {
            e.printStackTrace();
        }
    }

    public void AvanCilind3() {
        String topic = "/valv/cilindroacao";
        String payload = "6"; // Avança o cilindro 3
        try {
            byte[] encodedPayload =
payload.getBytes(StandardCharsets.UTF_8);
            MqttMessage message = new MqttMessage(encodedPayload);
            mqtt.publish(topic, message);
        } catch (MqttException e) {
            e.printStackTrace();
        }
    }

    public void RecuaCilind3() {
        String topic  =  "/valv/cilindroacao";
        String payload = "5"; // Recua o cilindro 3
        try {
            byte[] encodedPayload =
payload.getBytes(StandardCharsets.UTF_8);
```

```
            MqttMessage message = new MqttMessage(encodedPayload);
            mqtt.publish(topic, message);
        } catch (MqttException e) {
            e.printStackTrace();
        }
    }
}
```

APÊNDICE F – Código backend da tela automática do aplicativo

```
package com.somsakelect.android.rochapp;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.WindowManager;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Spinner;
import android.widget.TextView;
import android.widget.Toast;

import com.somsakelect.android.mqtt.MqttAndroidClient;

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;

import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class AutoControl extends AppCompatActivity {

    private Spinner spinner1, spinner2, spinner3;
    private String opcaoSpinner1, opcaoSpinner2, opcaoSpinner3;
    private String saida1 = "Peça na saida 1";
    private String saida2 = "Peça na saida 2";
    private String saida3 = "Peça na saida 3";
    private List<String> opcoes;

    private MqttAndroidClient mqtt;
    private static final String MQTT_HOST = "10.14.48.104";
    private static final int MQTT_PORT = 1883;
    private static final String MQTT_URL = "tcp://" + MQTT_HOST + ":" +
MQTT_PORT;
    private static final String MQTT_ID = MqttClient.generateClientId();
    private static final String TAG = "AutoControl";

    private TextView statusConexaoAuto;
    private View btnConectarAuto;

    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setTheme(R.style.Theme_MyMQTT);
        setContentView(R.layout.activity_auto_control);

getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);

        statusConexaoAuto = findViewById(R.id.tv_statusConexaoAuto);
        btnConectarAuto = findViewById(R.id.ll_btnConectarAuto);

        opcoes = new ArrayList<>(Arrays.asList("", "Peça Pequena", "Peça
Pequena Metallica", "Peça Média", "Peça Média Metallica", "Peça Grande",
"Peça Grande Metallica"));

        spinner1 = findViewById(R.id.OpcoesSaida1);
        spinner2 = findViewById(R.id.OpcoesSaida2);
        spinner3 = findViewById(R.id.OpcoesSaida3);

        ArrayAdapter<String> adapter = new ArrayAdapter<>(this,
android.R.layout.simple_spinner_item, opcoes);

adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_it
em);

        spinner1.setAdapter(adapter);
        spinner2.setAdapter(adapter);
        spinner3.setAdapter(adapter);

        spinner1.setOnItemSelectedListener(new
AdapterView.OnItemSelectedListener() {
            @Override
            public void onItemSelected(AdapterView<?> parent, View view,
int position, long id) {
                opcaoSpinner1 = opcoes.get(position);
                verificarOpcoesIguais();
            }

            @Override
            public void onNothingSelected(AdapterView<?> parent) {
            }
        });

        spinner2.setOnItemSelectedListener(new
AdapterView.OnItemSelectedListener() {
            @Override
            public void onItemSelected(AdapterView<?> parent, View view,
int position, long id) {
                opcaoSpinner2 = opcoes.get(position);
                verificarOpcoesIguais();
            }

            @Override
            public void onNothingSelected(AdapterView<?> parent) {
            }
        });

        spinner3.setOnItemSelectedListener(new
AdapterView.OnItemSelectedListener() {
            @Override
            public void onItemSelected(AdapterView<?> parent, View view,
int position, long id) {
```

```
                opcaoSpinner3 = opcoes.get(position);
                verificarOpcoesIguais();
            }

            @Override
            public void onNothingSelected(AdapterView<?> parent) {
            }
        });

        // MQTT
        mqtt = new MqttAndroidClient(this, MQTT_URL, MQTT_ID);
        mqtt.setCallback(new MqttCallbackExtended() {
            @Override
            public void connectComplete(boolean reconnect, String
serverURI) {
                statusConexaoAuto.setText(reconnect ? "Reconectando..." :
"Conectado!");
                if (mqtt.isConnected()) {
                    String tsx = "CONECTADO!";
                    statusConexaoAuto.setText(tsx);
                    btnConectarAuto.setVisibility(View.INVISIBLE);
                    // Subscrever tópicos aqui
                    subscribe("/sensor/peça"); // Recebe o status do ESP32
dos sensores de entrada
                    subscribe("/valv/recuo"); // Recebe o status do ESP32
dos sensores de saida
                }
            }

            @Override
            public void connectionLost(Throwable cause) {
                if (cause != null) {
                    String st = "Conexão perdida! " + cause.getMessage();
                    statusConexaoAuto.setText(st);
                    btnConectarAuto.setVisibility(View.VISIBLE);
                }
            }

            @Override
            public void messageArrived(String topic, MqttMessage message) {
                if (topic.equals("/sensor/peça")) {
                    String sensIN = message.toString();
                    // Exibir a mensagem na caixa de texto com o ID
"ind_peca"
                    TextView peca = findViewById(R.id.ind_peca);
                    if (peca != null) {
                        peca.setText(sensIN);
                    }

                    // Comparar com opcaoSpinner1
                    if (opcaoSpinner1 != null &&
opcaoSpinner1.equals(sensIN)) {
                        // Publicar no tópico "/valv/cilindroacao" com
payload "2"
                        publishToTopic("/valv/cilindroacao", "2");
                    }

                    // Comparar com opcaoSpinner2
                    if (opcaoSpinner2 != null &&
opcaoSpinner2.equals(sensIN)) {
                        // Publicar no tópico "/valv/cilindroacao" com
```

```
payload "4"
                        publishToTopic("/valv/cilindroacao", "4");
                    }

                    // Comparar com opcaoSpinner3
                    if (opcaoSpinner3 != null &&
opcaoSpinner3.equals(sensIN)) {
                        // Publicar no tópico "/valv/cilindroacao" com
payload "6"
                        publishToTopic("/valv/cilindroacao", "6");
                    }
                }
                if (topic.equals("/valv/recuo")) {
                    //String sensorB;
                    String sensOUT = message.toString();
                    // Exibir a mensagem na caixa de texto com o ID
"ind_saida"
                    TextView saida = findViewById(R.id.ind_saida);
                    if (saida != null) {
                        saida.setText(sensOUT);
                    }

                    // Comparar com a saida1
                    if (sensOUT != null && saida1.equals(sensOUT)) {
                        // Publicar no tópico "/valv/cilindroacao" com
payload "1"
                        publishToTopic("/valv/cilindroacao", "1");
                    }
                    // Comparar com a saida2
                    if (sensOUT != null && saida2.equals(sensOUT)) {
                        // Publicar no tópico "/valv/cilindroacao" com
payload "3"
                        publishToTopic("/valv/cilindroacao", "3");
                    }
                    // Comparar com a saida3
                    if (sensOUT != null && saida3.equals(sensOUT)) {
                        // Publicar no tópico "/valv/cilindroacao" com
payload "5"
                        publishToTopic("/valv/cilindroacao", "5");
                    }
                }
            }

            @Override
            public void deliveryComplete(IMqttDeliveryToken token) {
            }
        });

        // Botão Conectar
        findViewById(R.id.conectarAuto_btn).setOnClickListener(v ->
connectMQTT());

        // Botão Voltar
        findViewById(R.id.voltarAuto_btn).setOnClickListener(v -> {
            Intent tela_inicial = new Intent(AutoControl.this,
MainActivity.class);
            disconnectMQTT();
            startActivity(tela_inicial);
        });

        // Tentar conectar
```

```
        connectMQTT();
    }

    private void verificarOpcoesIguais() {
        if (opcaoSpinner1 != null && opcaoSpinner2 != null && opcaoSpinner3
!= null) {
            if (opcaoSpinner1.equals(opcaoSpinner2) ||
opcaoSpinner1.equals(opcaoSpinner3) || opcaoSpinner2.equals(opcaoSpinner3))
{
                Toast.makeText(getApplicationContext(), "Por favor,
selecione opções diferentes para cada spinner", Toast.LENGTH_SHORT).show();
            }
        }
    }

    private void connectMQTT() {
        if (mqtt.isConnected()) {
            showToast("Broker já conectado!");
            statusConexaoAuto.setText(R.string.conectado);
            btnConectarAuto.setVisibility(View.INVISIBLE);
            return;
        }

        statusConexaoAuto.setText(R.string.conectando);

        // Recupere o valor da string (USERNAME e PASSWORD) do Intent
        //final String MQTT_USERNAME = "rocha";
        //final String MQTT_PASSWORD = "rocha";

        // Set options
        MqttConnectOptions options = new MqttConnectOptions();
        //options.setUserName(MQTT_USERNAME);
        //options.setPassword(MQTT_PASSWORD.toCharArray());
        options.setAutomaticReconnect(true);
        options.setCleanSession(true);
        try {
            IMqttToken token = mqtt.connect(options);
            token.setActionCallback(new IMqttActionListener() {
                @Override
                public void onSuccess(IMqttToken asyncActionToken) {
                    // Subscrever tópicos aqui
                    subscribe("/sensor/peça"); // Recebe o status do ESP32
dos sensores de fim de curso
                    subscribe("/valv/recuo");
                }

                @Override
                public void onFailure(IMqttToken asyncActionToken,
Throwable exception) {
                    String tsx = "Falha na conexão... " +
exception.getMessage();
                    statusConexaoAuto.setText(tsx);
                    btnConectarAuto.setVisibility(View.VISIBLE);
                }
            });
        } catch (MqttException e) {
            e.printStackTrace();
            String tsx = "Connect MqttException: " + e.getMessage();
            statusConexaoAuto.setText(tsx);
            btnConectarAuto.setVisibility(View.VISIBLE);
        }
```

```
    }

    private void disconnectMQTT() {
        try {
            IMqttToken token = mqtt.disconnect();
            token.setActionCallback(new IMqttActionListener() {
                @Override
                public void onSuccess(IMqttToken asyncActionToken) {
                }

                @Override
                public void onFailure(IMqttToken asyncActionToken,
Throwable exception) {
                }
            });
        } catch (MqttException e) {
            e.printStackTrace();
        }
    }

    private void subscribe(String topic) {
        if (!mqtt.isConnected()) {
            return;
        }

        try {
            IMqttToken token = mqtt.subscribe(topic, 0);
            token.setActionCallback(new IMqttActionListener() {
                @Override
                public void onSuccess(IMqttToken asyncActionToken) {
                    showToast("Subscribed..."
                            +
Arrays.toString(asyncActionToken.getTopics()));
                }

                @Override
                public void onFailure(IMqttToken asyncActionToken,
Throwable exception) {
                    showToast("Subscribe failed..."
                            +
Arrays.toString(asyncActionToken.getTopics()));
                }
            });
        } catch (MqttException e) {
            e.printStackTrace();
        }
    }

    private void publishToTopic(String topic, String payload) {
        if (!mqtt.isConnected()) {
            return;
        }

        try {
            MqttMessage message = new
MqttMessage(payload.getBytes(StandardCharsets.UTF_8));
            IMqttToken token = mqtt.publish(topic, message);
            token.setActionCallback(new IMqttActionListener() {
                @Override
                public void onSuccess(IMqttToken asyncActionToken) {
                    showToast("Publicado no tópico: " + topic);
```

```
                }

                @Override
                public void onFailure(IMqttToken asyncActionToken,
Throwable exception) {
                    showToast("Falha ao publicar no tópico: " + topic);
                }
            });
        } catch (MqttException e) {
            e.printStackTrace();
        }
    }

    private void showToast(String message) {
        Toast.makeText(this, message, Toast.LENGTH_SHORT).show();
    }
}
```