Roteiro Nº 10	**Node-RED e ESP32 - interações com broker MQTT**	
Curso Engenharia Elétrica	**Disciplina** ELT 1119 – Redes e Aplicações IoT	**Professor** Carlos Alberto Vasconcelos Bezerra
Nome do Estudante		**Data**

1. Objetivos da aula

- Compreender como utilizar o Node-RED para enviar comandos via MQTT
- Programar o ESP32 em MicroPython para receber comandos PWM
- Controlar a frequência de um inversor de frequência a partir de um sinal PWM filtrado
- Aplicar conceitos de automação, IoT e eletrônica de potência no controle de motores de indução

2. Materiais utilizados

- Placa ESP32
- Inversor de frequência com motor de indução trifásico
- Jumpers
- Protoboard
- Cabo USB para comunicação e alimentação
- Computador com:
  - Python instalado
  - Thonny IDE
  - Node.js
  - Node-Red
  - Acesso à internet

3. Resumo teórico

O Node-RED é uma ferramenta de programação visual baseada em fluxo, desenvolvida pela IBM, amplamente utilizada em projetos de automação e Internet das Coisas (IoT). Com sua interface intuitiva e modular, o Node-RED permite a integração de dispositivos físicos, serviços online e APIs por meio de protocolos como MQTT, HTTP, Modbus, entre outros.

Nesta aula, será utilizada a comunicação entre o Node-RED e o ESP32 por meio do protocolo MQTT, com o ESP32 programado em MicroPython. O ESP32 atuará como cliente MQTT, inscrito em um tópico específico para receber comandos de controle de largura de pulso (PWM). Esses comandos serão gerados no Node-RED e enviados ao ESP32, que converterá o valor recebido em um sinal PWM proporcional.

Este sinal PWM será filtrado por um circuito RC, convertendo-o em uma tensão contínua analógica. Essa tensão será utilizada como entrada de referência para um inversor de frequência, que por sua vez controlará a velocidade de um motor de indução trifásico, variando sua frequência de operação de forma proporcional ao sinal recebido.

Para isso, será necessário:

- Criar fluxos no Node-RED que permitam o envio de comandos MQTT ao ESP32;
- Programar o ESP32 para interpretar esses comandos e gerar um PWM proporcional;
- Filtrar o PWM com um filtro ativo para obtenção de sinal analógico suave;
- Aplicar a tensão resultante ao inversor de frequência, controlando a frequência (e, portanto, a velocidade) do motor de indução.

4. Desenvolvimento da aula

4.1 Montagem do Circuito

- Conecte o pino GPIO 5 do ESP32 ao filtro ativo para suavizar o sinal PWM.
- A saída do filtro deve ser conectada à entrada analógica de controle do inversor de frequência.
- O inversor deve estar ligado corretamente ao motor de indução trifásico, conforme recomendações do fabricante.

4.2 Programação do ESP32

- Programe o ESP32 com o código MicroPython fornecido (ver Anexo 1), que:
  - Conecta à rede Wi-Fi especificada.
  - Conecta-se a um broker MQTT (test.mosquitto.org).
  - Inscreve-se no tópico esp32/pwm.
  - Recebe valores de 0 a 120 e converte para duty cycle PWM de 0 a 100%.
  - Atualiza o sinal PWM na saída GPIO 5.

4.3 Verificação da Conexão MQTT

- Após carregar o código, conecte-se ao terminal do ESP32 via Thonny.
- Verifique no terminal:
  - A conexão Wi-Fi foi estabelecida com sucesso (IP exibido).
  - A conexão com o broker MQTT foi realizada.
  - O ESP32 está inscrito no tópico esp32/pwm.

4.4 Configuração do Node-RED

- Abra o Node-RED no navegador: http://localhost:1880.
- Crie um novo fluxo (ver Anexo 2) com os seguintes nós:
  - Um nó inject com payload numérico (ex: 0, 60, 120), configurado para enviar valores representando a frequência desejada.
  - Um nó mqtt out:
    - Tópico: esp32/pwm
    - Broker: test.mosquitto.org ou IP do seu broker MQTT

4.5 Testes no Node-RED

- Clique no botão do nó inject para enviar o valor escolhido (ex: 60).
- No terminal do ESP32, verifique se o valor foi recebido e o duty atualizado.
- Meça a tensão na saída do filtro com um multímetro.
- Verifique no display do inversor se a frequência do motor varia proporcionalmente.
- Observe o comportamento do motor de indução.

5. Questões

Questão 1 – Conversão PWM
Durante a aula, foi utilizado um valor de 0 a 120 no Node-RED para controlar o PWM do ESP32. Sabendo que esse valor é convertido para um duty cycle proporcional entre 0 e 1023, calcule:
a) O duty correspondente ao valor 90.
b) A tensão esperada na saída do filtro ativo, sabendo que 1023 equivale a 3,3 V.
Apresente os cálculos.

Questão 2 – Papel do Broker MQTT
Explique com suas próprias palavras o papel do **broker MQTT** no funcionamento do sistema proposto. Por que ele é necessário na comunicação entre o Node-RED e o ESP32?

Questão 3 – Publicador e Assinante
No exemplo utilizado em aula:
a) Quem atua como **publicador** e quem atua como **assinante** no protocolo MQTT?
b) O que aconteceria se o ESP32 não estivesse inscrito no tópico esp32/pwm?

Questão 4 – Relação PWM x Frequência do Motor
Sabendo que o valor PWM enviado ao ESP32 influencia diretamente a frequência do inversor, que por sua vez controla a velocidade do motor de indução:
a) Qual é a frequência do motor esperada quando o valor PWM enviado for 60?
(Considere que 0 corresponde a 0 Hz e 120 corresponde a 60 Hz)
b) E qual seria a rotação (RPM) esperada para um motor de 2 polos?

Questão 5 – Segurança e Aplicações
a) Cite dois cuidados de segurança elétrica que devem ser observados ao trabalhar com inversores de frequência e motores.
b) Dê um exemplo de aplicação real em que o controle de velocidade via PWM e MQTT pode ser útil em um sistema industrial ou residencial.

6. Códigos utilizados (em anexo)

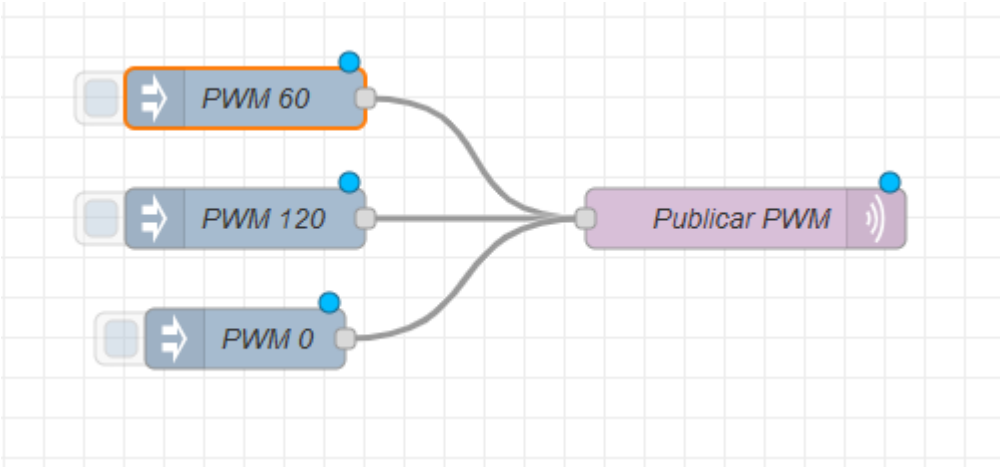
Anexo 1

```
import network
import time
from umqtt.simple import MQTTClient
from machine import Pin, PWM

# === CONFIGURAÇÕES DO USUÁRIO ===
SSID = 'SEU_WIFI'
SENHA = 'SUA_SENHA'
BROKER = 'test.mosquitto.org'
TOPICO = b'esp32/pwm'

# === CONFIGURAÇÃO DO PWM ===
pwm = PWM(Pin(5), freq=1000)
pwm.duty(0)

# === FUNÇÃO PARA CONVERTER VALOR PARA DUTY (0–1023) ===
def pwm_duty(valor):
    duty = int((valor / 120) * 1023)
    return min(max(duty, 0), 1023)

# === CONEXÃO WI-FI ===
wifi = network.WLAN(network.STA_IF)
wifi.active(True)
wifi.connect(SSID, SENHA)

while not wifi.isconnected():
    print("Conectando ao Wi-Fi...")
    time.sleep(1)
```

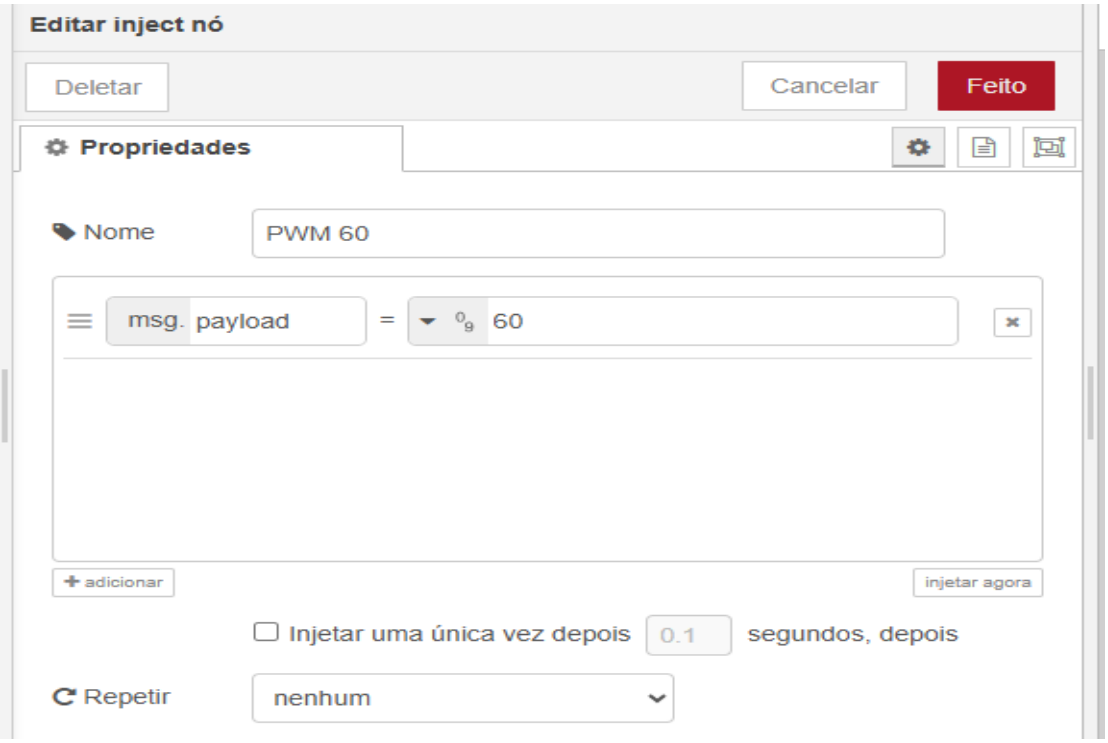
```
print("Conectado ao Wi-Fi:", wifi.ifconfig())

# === CALLBACK DO MQTT ===
def tratar_mensagem(topico, msg):
    try:
        valor_pwm = int(msg)
        duty = pwm_duty(valor_pwm)
        pwm.duty(duty)
        print(f"Recebido: {valor_pwm} → Duty: {duty}")
    except Exception as e:
        print("Erro ao interpretar valor:", e)

# === CONEXÃO COM BROKER MQTT ===
cliente = MQTTClient("esp32", BROKER, port=1883)
cliente.set_callback(tratar_mensagem)
cliente.connect()
cliente.subscribe(TOPICO)
print(f"Conectado ao broker MQTT '{BROKER}', aguardando dados no tópico '{TOPICO.decode()}'")

# === LOOP PRINCIPAL ===
try:
    while True:
        cliente.check_msg()
        time.sleep(0.1)
except KeyboardInterrupt:
    cliente.disconnect()
    print("Desconectado do broker")
```

Anexo 2

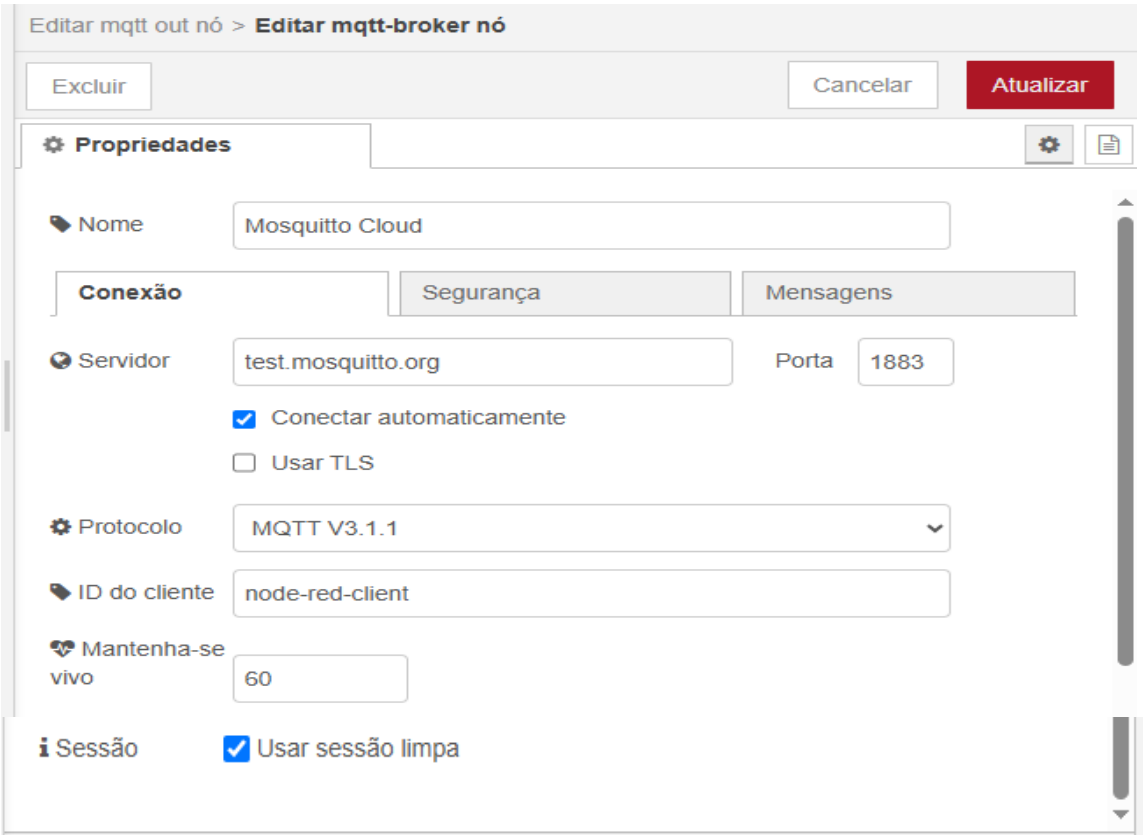
Fluxo



Nó Inject



Nó mqtt out
Preencher o nó e depois clicar em editar



Anexo3

Fluxo Node-RED (JSON para Importação)
Esse fluxo permite testar o controle PWM sem dashboard, usando botões de injeção.
Como importar:

1. Copie o JSON abaixo.
2. No Node-RED, clique no menu (☰) → *Import* → *Clipboard* → cole e clique em *Import*.

```
[
  {
    "id": "inject_pwm_60",
    "type": "inject",
    "z": "fluxo_pwm",
    "name": "PWM 60",
    "props": [
      {
        "p": "payload"
      }
    ],
    "repeat": "",
    "crontab": "",
    "once": false,
    "onceDelay": 0.1,
    "topic": "",
```

```
  "payload": "60",
  "payloadType": "num",
  "x": 140,
  "y": 100,
  "wires": [["mqtt_out"]]
 },
 {
  "id": "inject_pwm_120",
  "type": "inject",
  "z": "fluxo_pwm",
  "name": "PWM 120",
  "props": [
   {
    "p": "payload"
   }
  ],
  "repeat": "",
  "crontab": "",
  "once": false,
  "onceDelay": 0.1,
  "topic": "",
  "payload": "120",
  "payloadType": "num",
  "x": 140,
  "y": 160,
  "wires": [["mqtt_out"]]
 },
 {
  "id": "inject_pwm_0",
  "type": "inject",
  "z": "fluxo_pwm",
  "name": "PWM 0",
  "props": [
   {
    "p": "payload"
   }
  ],
  "repeat": "",
  "crontab": "",
  "once": false,
  "onceDelay": 0.1,
  "topic": "",
  "payload": "0",
  "payloadType": "num",
  "x": 140,
  "y": 220,
  "wires": [["mqtt_out"]]
 },
 {
  "id": "mqtt_out",
```

```
    "type": "mqtt out",
    "z": "fluxo_pwm",
    "name": "Publicar PWM",
    "topic": "esp32/pwm",
    "qos": "",
    "retain": "",
    "broker": "broker_mqtt_cloud",
    "x": 390,
    "y": 160,
    "wires": []
  },
  {
    "id": "broker_mqtt_cloud",
    "type": "mqtt-broker",
    "name": "Mosquitto Cloud",
    "broker": "test.mosquitto.org",
    "port": "1883",
    "clientid": "node-red-client",
    "usetls": false,
    "compatmode": false,
    "keepalive": "60",
    "cleansession": true,
    "birthTopic": "",
    "birthQos": "0",
    "birthPayload": "",
    "closeTopic": "",
    "closePayload": "",
    "willTopic": "",
    "willQos": "0",
    "willPayload": ""
  }
]
```


Anexo 4

Instalação do Node-RED

Antes de instalar o Node-RED, é necessário ter o **Node.js** instalado no sistema.
Node.js é o ambiente de execução JavaScript necessário para rodar o Node-RED.

Instalação no Windows, Linux ou macOS (via terminal)

1. Instale o Node.js

- Acesse: https://nodejs.org
- Baixe e instale a versão LTS (Long Term Support)

2. Instale o Node-RED com o npm
Após instalar o Node.js, abra o terminal (ou PowerShell no Windows) e digite:

npm install -g --unsafe-perm node-red

3. Como iniciar o Node-RED
Após a instalação, execute:

node-red
Isso iniciará o servidor e exibirá no terminal o endereço local, normalmente:

http://127.0.0.1:1880

4.Abra esse link no navegador para acessar o editor visual.

Bibliografia

ASCHER, D.; LUTZ, M. Aprendendo Python. Porto Alegre: Bookman, 2007.

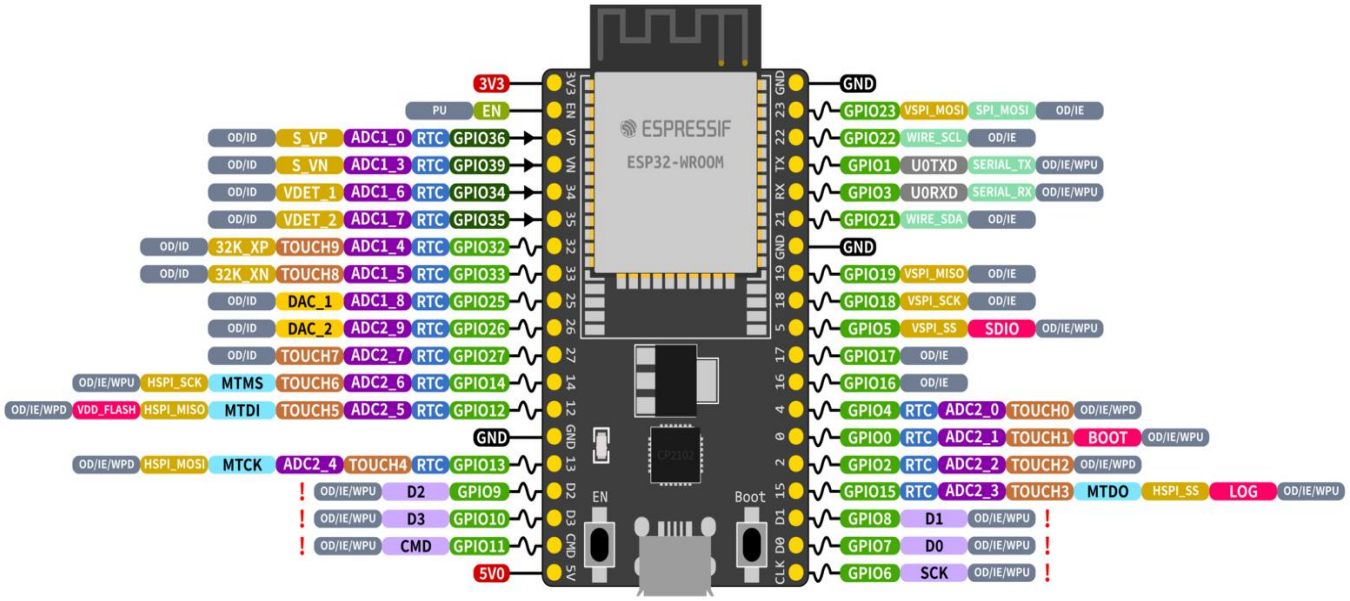
MENEZES, N. N. C. Introdução à programação com Python. São Paulo: Ed. Novatec, 2014.

OLIVEIRA, S. Internet das coisas com ESP8266, Arduino e Raspberry Pi. São Paulo: Ed. Novatec, 2017.

OLIVEIRA, C. L. V. IoT com Micropython e NodeMCU. São Paulo: Ed. Novatec, 2022.

ESP32-DevKitC

ESPRESSIF



ESP32 Specs

32-bit Xtensa® dual-core @240MHz
Wi-Fi IEEE 802.11 b/g/n 2.4GHz
BLuetooth 4.2 BR/EDR and BLE
520 KB SRAM (16 KB for cache)
448 KB ROM
34 GPIOs, 4x SPI, 3x UART, 2x I2C,
2x I2S, RMT, LED PWM, 1 host SD/eMMC/SDIO,
1 slave SDIO/SPI, TWAI®, 12-bit ADC, Ethernet

PWM Capable Pin
GPIOX GPIO Input Only
GPIOX GPIO Input and Output
DAC_X Digital-to-Analog Converter
DEBUG JTAG for Debugging
FLASH External Flash Memory (SPI)
ADCX_CH Analog-to-Digital Converter
TOUCHX Touch Sensor Input Channel
OTHER Other Related Functions
SERIAL Serial for Debug/Programming
ARDUINO Arduino Related Functions
STRAP Strapping Pin Functions

RTC RTC Power Domain (VDD3P3_RTC)
GND Ground
PWD Power Rails (3V3 and 5V)
! Pin Shared with the Flash Memory
Can't be used as regular GPIO

GPIO STATE
WPU: *Weak Pull-up (Internal)*
WPD: *Weak Pull-down (Internal)*
PU: *Pull-up (External)*
IE: *Input Enable (After Reset)*
ID: *Input Disabled (After Reset)*
OE: *Output Enable (After Reset)*
OD: *Output Disabled (After Reset)*