Controle de Posição e Velocidade para Elevadores em Edifícios Residenciais

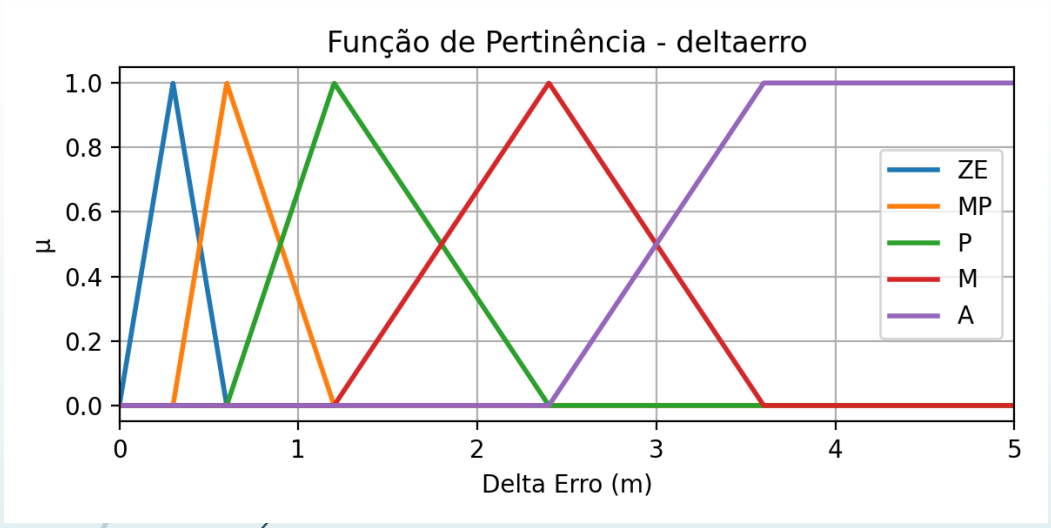
Gustavo Ribeiro de Oliveira

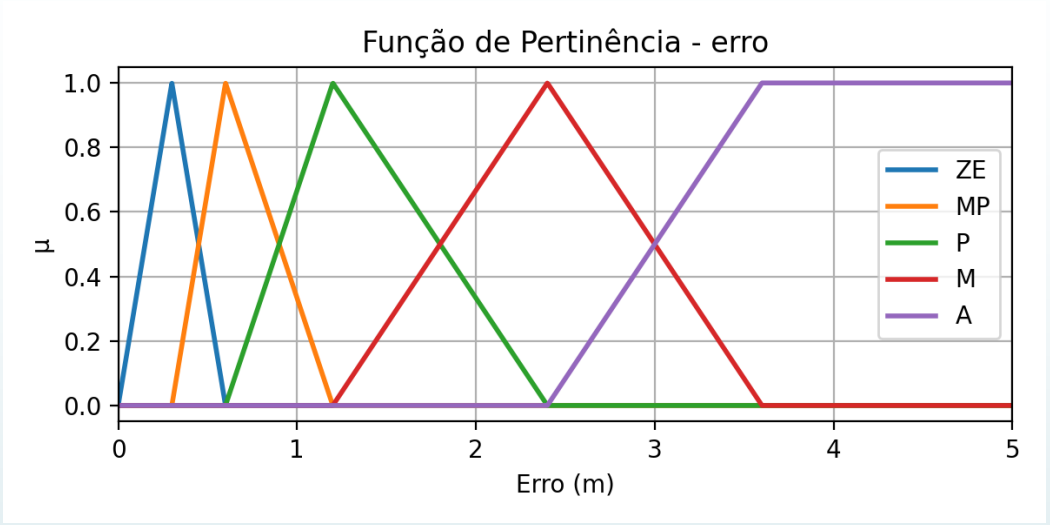
Tecnológias

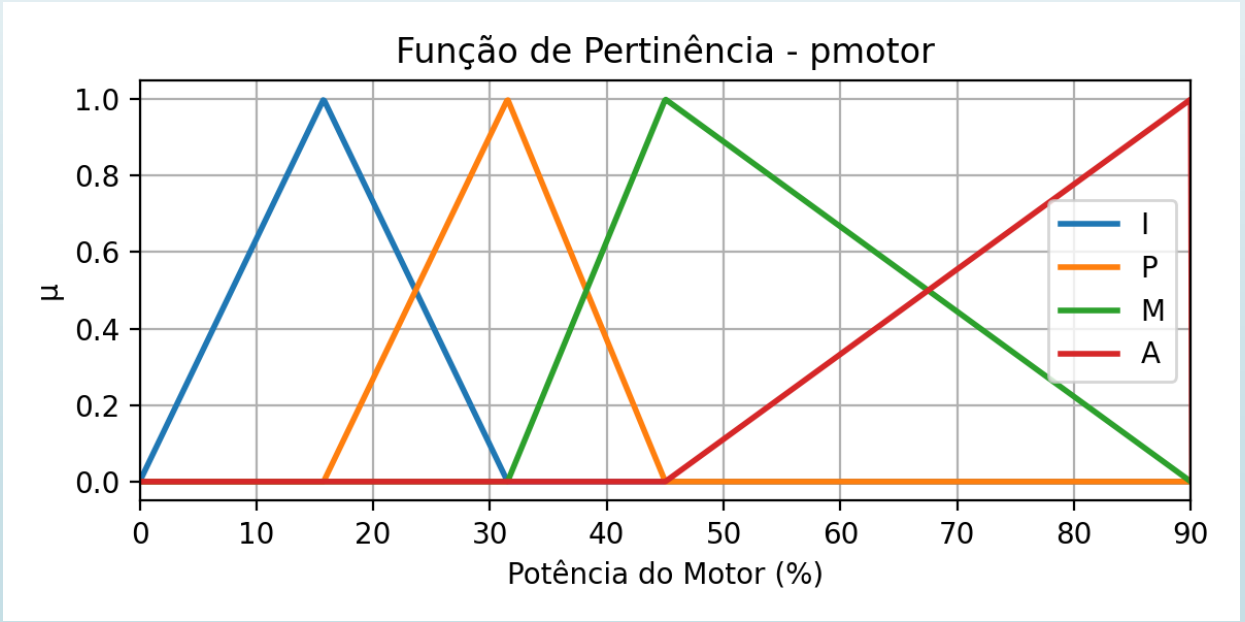
Universos

```
Erro['ZE'] = fuzzy.trimf(Erro.universe, [0, 0.3, 0.6])
Erro['MP'] = fuzzy.trimf(Erro.universe, [0.3, 0.6, 1.2])
Erro['P'] = fuzzy.trimf(Erro.universe, [0.6, 1.2, 2.4])
Erro['M'] = fuzzy.trimf(Erro.universe, [1.2, 2.4, 3.6])
Erro['A'] = fuzzy.trapmf(Erro.universe, [2.4, 3.6, 32, 32])

DeltaErro['ZE'] = fuzzy.trimf(Erro.universe, [0, 0.3, 0.6])
DeltaErro['MP'] = fuzzy.trimf(Erro.universe, [0.3, 0.6, 1.2])
DeltaErro['P'] = fuzzy.trimf(Erro.universe, [0.6, 1.2, 2.4])
DeltaErro['M'] = fuzzy.trimf(Erro.universe, [1.2, 2.4, 3.6])
DeltaErro['A'] = fuzzy.trapmf(Erro.universe, [2.4, 3.6, 32, 32])

PMotor['I'] = fuzzy.trimf(PMotor.universe, [0, 15.75, 31.5])
PMotor['P'] = fuzzy.trimf(PMotor.universe, [15.75, 31.5, 45])
PMotor['M'] = fuzzy.trimf(PMotor.universe, [31.5, 45, 90])
PMotor['A'] = fuzzy.trimf(PMotor.universe, [45, 90, 90])
```

Universos


Função de Pertinência - deltaerro
μ
Delta Erro (m)
ZE
MP
P
M
A


Função de Pertinência - erro
μ
Erro (m)
ZE
MP
P
M
A


Função de Pertinência - pmotor
μ
Potência do Motor (%)
I
P
M
A

Regras

```
R1 = ctrl.Rule(Erro['ZE'] & DeltaErro['ZE'], PMotor['I'])
R2 = ctrl.Rule(Erro['MP'] & DeltaErro['ZE'], PMotor['P'])
R3 = ctrl.Rule(Erro['P'] & DeltaErro['ZE'], PMotor['P'])
R4 = ctrl.Rule(Erro['M'] & DeltaErro['ZE'], PMotor['P'])
R5 = ctrl.Rule(Erro['A'] & DeltaErro['ZE'], PMotor['M'])

R6 = ctrl.Rule(Erro['ZE'] & DeltaErro['MP'], PMotor['P'])
R7 = ctrl.Rule(Erro['MP'] & DeltaErro['MP'], PMotor['P'])
R8 = ctrl.Rule(Erro['P'] & DeltaErro['MP'], PMotor['P'])
R9 = ctrl.Rule(Erro['M'] & DeltaErro['MP'], PMotor['M'])
R10 = ctrl.Rule(Erro['A'] & DeltaErro['MP'], PMotor['M'])

R11 = ctrl.Rule(Erro['ZE'] & DeltaErro['P'], PMotor['P'])
R12 = ctrl.Rule(Erro['MP'] & DeltaErro['P'], PMotor['P'])
R13 = ctrl.Rule(Erro['P'] & DeltaErro['P'], PMotor['M'])
```

```
R14 = ctrl.Rule(Erro['M'] & DeltaErro['P'], PMotor['M'])
R15 = ctrl.Rule(Erro['A'] & DeltaErro['P'], PMotor['A'])

R16 = ctrl.Rule(Erro['ZE'] & DeltaErro['M'], PMotor['P'])
R17 = ctrl.Rule(Erro['MP'] & DeltaErro['M'], PMotor['M'])
R18 = ctrl.Rule(Erro['P'] & DeltaErro['M'], PMotor['M'])
R19 = ctrl.Rule(Erro['M'] & DeltaErro['M'], PMotor['A'])
R20 = ctrl.Rule(Erro['A'] & DeltaErro['M'], PMotor['A'])

R21 = ctrl.Rule(Erro['ZE'] & DeltaErro['A'], PMotor['M'])
R22 = ctrl.Rule(Erro['MP'] & DeltaErro['A'], PMotor['M'])
R23 = ctrl.Rule(Erro['P'] & DeltaErro['A'], PMotor['A'])
R24 = ctrl.Rule(Erro['M'] & DeltaErro['A'], PMotor['A'])
R25 = ctrl.Rule(Erro['A'] & DeltaErro['A'], PMotor['A'])
```

E	ZE	MP	P	M	A
ZE	I	P	P	P	M
MP	P	P	P	M	M
P	P	P	M	M	A
M	P	M	M	A	A
A	M	M	A	A	A

Controle fuzzy

```
while True:
    if parar_controle:
        em_execucao = False
        return

    erro = abs(posicao_atual - setpoint)
    delta_erro = abs(erro - erro_anterior)

    if erro <= 0.03:
        em_execucao = False
        return

    controle.input['Erro'] = erro
    controle.input['DeltaErro'] = delta_erro
    controle.compute()
    potencia = controle.output['PMotor'] / 100

    posicao_atual = abs(k1 * posicao_atual * 0.9995 + potencia * 0.212312)
    client.publish(topic_posicao, round(posicao_atual, 3))
    time.sleep(Ts)

    erro_anterior = erro
    k1 = 1 if setpoint > posicao_atual else -1
```

Inicialização do elevador

```
for i in range(10):
    if parar_controle:
        em_execucao = False
        return
    potencia_inicial = 0.0315 * (i + 1)
    posicao_atual = abs(k1 * posicao_atual * 0.999 + potencia_inicial * 0.251287)
    client.publish(topic_posicao, round(posicao_atual, 3))
    time.sleep(Ts)
```

Configuração MQTT

```
broker = 'broker.hivemq.com'
topic_setpoint = 'elevador/destino'
topic_posicao = 'elevador/posicao'
topic_resetar = 'elevador/resetar'
```

```
# Callback do MQTT
def on_message(client, userdata, msg):
    global setpoint, posicao_atual, em_execucao, parar_controle
    try:
        if msg.topic == topic_setpoint:
            novo = float(msg.payload.decode())
            setpoint = novo
            threading.Thread(target=controle_loop).start()
        elif msg.topic == topic_resetar:
            print("Reset solicitado!")
            parar_controle = True
            setpoint = 4
            posicao_atual = 4
            client.publish(topic_posicao, round(posicao_atual, 3))
    except Exception as e:
        print("Erro ao processar mensagem:", e)


client = mqtt.Client()
client.connect(broker, 1883, 60)
client.subscribe(topic_setpoint)
client.subscribe(topic_resetar)
client.on_message = on_message

print("Aguardando comandos MQTT...")
client.loop_forever()
```

Configuração MQTT

```
const client = mqtt.connect("wss://broker.hivemq.com:8884/mqtt");

function enviarDestino(valor) {
    const posAtual = grafico.data.datasets[0].data.at(-1);
    if (Math.abs(posAtual - valor) < 1) {
        exibirErro("O elevador já está no andar selecionado.");
        return;
    }

    destinoSelecionado.valor = valor;
    client.publish("elevador/destino", valor.toString());

    tempo = 0;
    grafico.data.labels = [];
    grafico.data.datasets[0].data = [];
    grafico.data.datasets[1].data = [];
    grafico.update();
    metricas.innerHTML = '';

    // Marca botão pressionado
    if (botaoAtual) botaoAtual.classList.remove("pressionado");
    botaoAtual = document.querySelector(`button[data-valor="${valor}"]`);
    if (botaoAtual) botaoAtual.classList.add("pressionado");
}
```