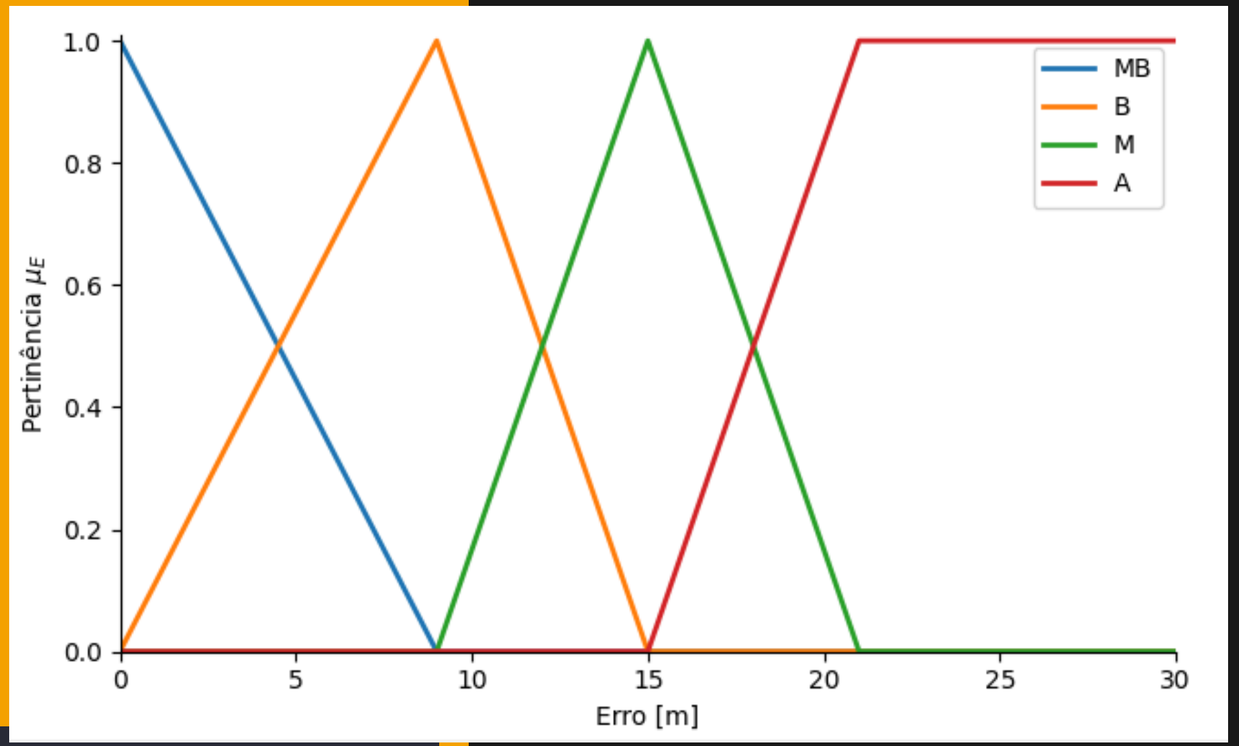
Projeto Fuzzy

Controle de Posição e Velocidade para Elevadores em Edifícios Residenciais

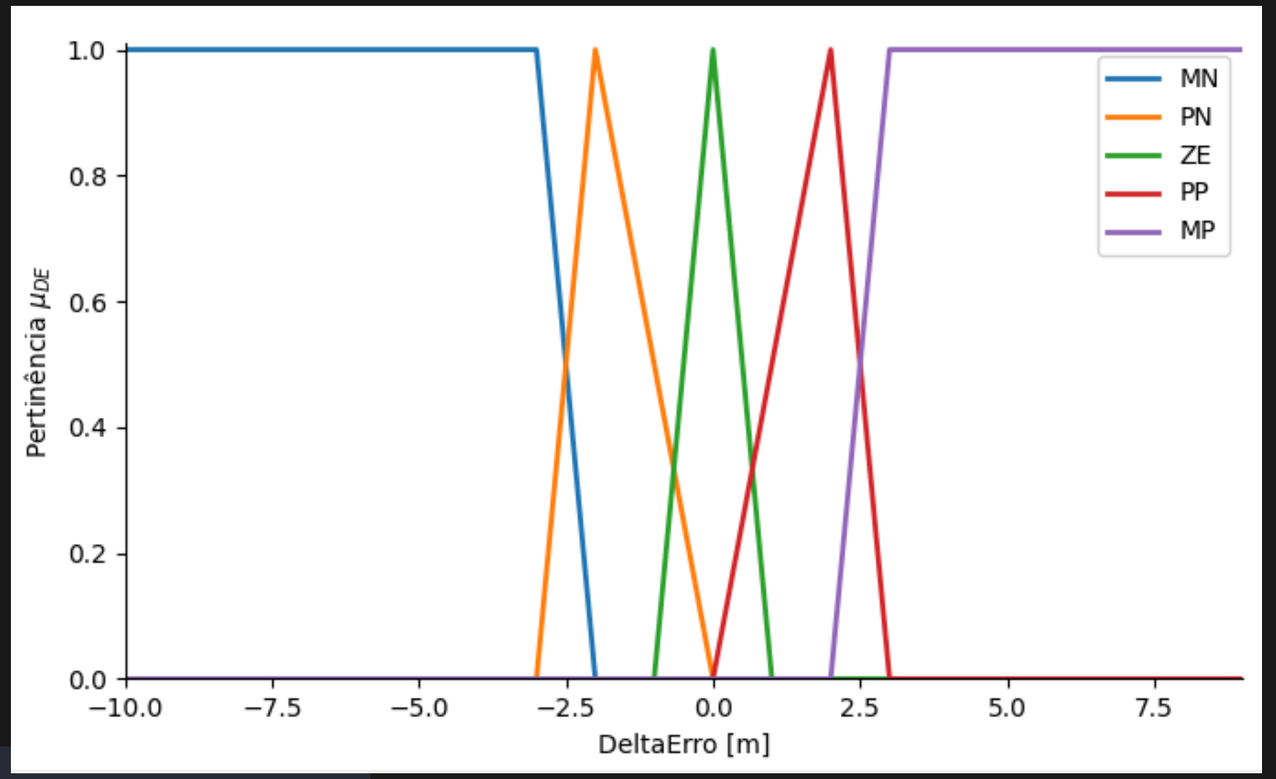
júlia de Freitas Carvalho

Funções de Pertinência



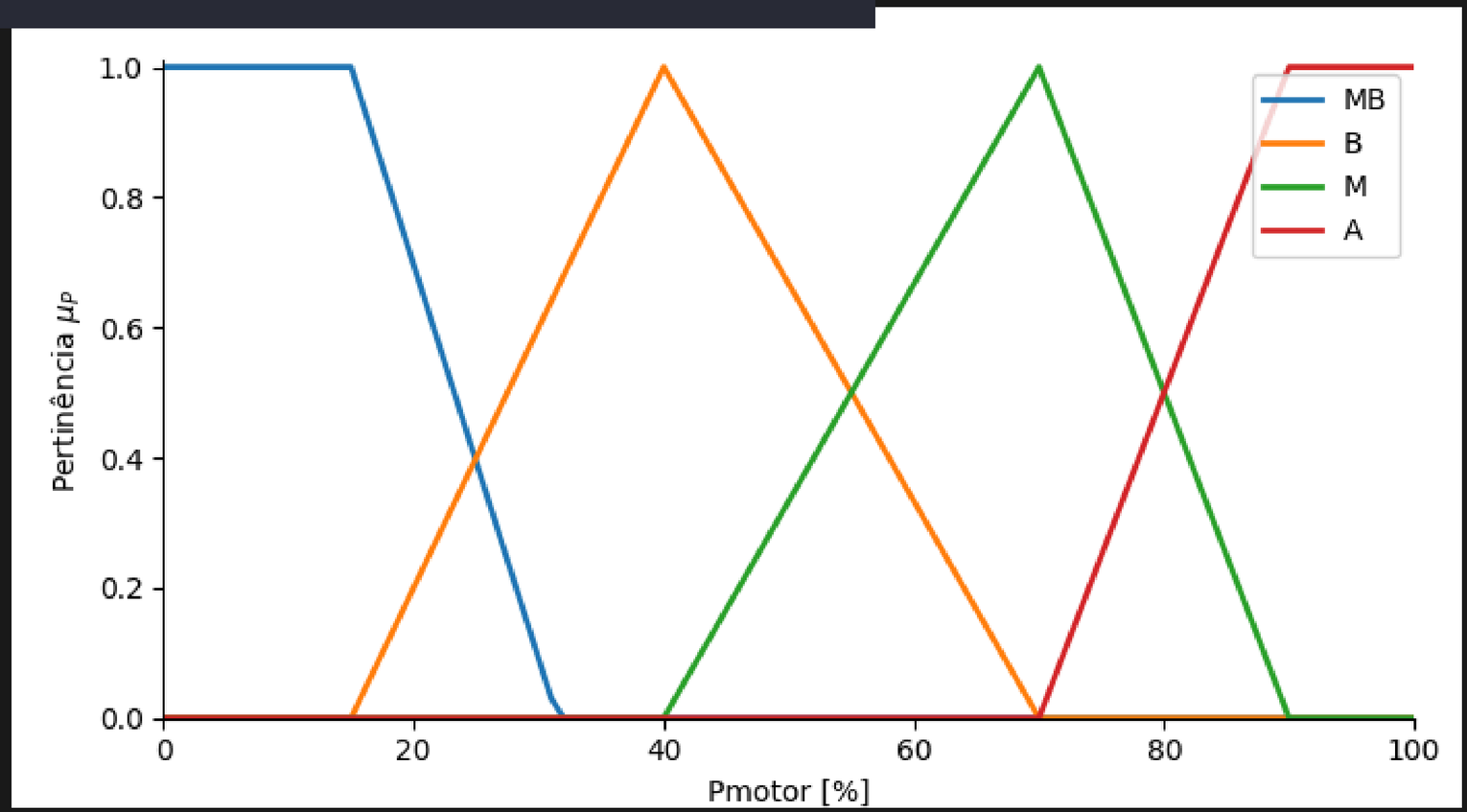

```python
# ---- Erro (posição de destino - posição atual)
Erro = ctrl.Antecedent(np.arange(0, 31, 1), 'Erro')
Erro.membershipA = 'E'
Erro.unit = 'm'
Erro['MB'] = fuzzy.trapmf(Erro.universe, [0, 0, 0, 9])
Erro['B']  = fuzzy.trimf(Erro.universe, [0, 9, 15])
Erro['M']  = fuzzy.trimf(Erro.universe, [9, 15, 21])
Erro['A']  = fuzzy.trapmf(Erro.universe, [15, 21, 32, 32])
```

Funções de Pertinência




```
# ---- DeltaErro (erro atual - erro anterior)
DeltaErro = ctrl.Antecedent(np.arange(-31, 31, 1), 'DeltaErro')
DeltaErro.membershipA = 'DE'
DeltaErro.unit = 'm'
DeltaErro['MN'] = fuzzy.trapmf(DeltaErro.universe, [-25, -25, -3, -2])
DeltaErro['PN'] = fuzzy.trimf(DeltaErro.universe, [-3, -2, 0])
DeltaErro['ZE'] = fuzzy.trimf(DeltaErro.universe, [-0.5, 0, 0.5])
DeltaErro['PP'] = fuzzy.trimf(DeltaErro.universe, [0, 2, 3])
DeltaErro['MP'] = fuzzy.trapmf(DeltaErro.universe, [2,3, 25, 25])
```

```python
# ---- Pmotor (potência do motor - saída)
Pmotor = ctrl.Consequent(np.arange(0, 101, 1), 'Pmotor')
Pmotor.membershipA = 'P'
Pmotor.unit = '%'
Pmotor['MB']  = fuzzy.trapmf(Pmotor.universe, [0, 0, 15, 31.5]) #Inicialização
Pmotor['B']  = fuzzy.trimf(Pmotor.universe, [15, 40, 70])
Pmotor['M']  = fuzzy.trimf(Pmotor.universe, [40, 70, 90])
Pmotor['A']  = fuzzy.trapmf(Pmotor.universe, [70, 90, 100, 100])
```



Base de regras

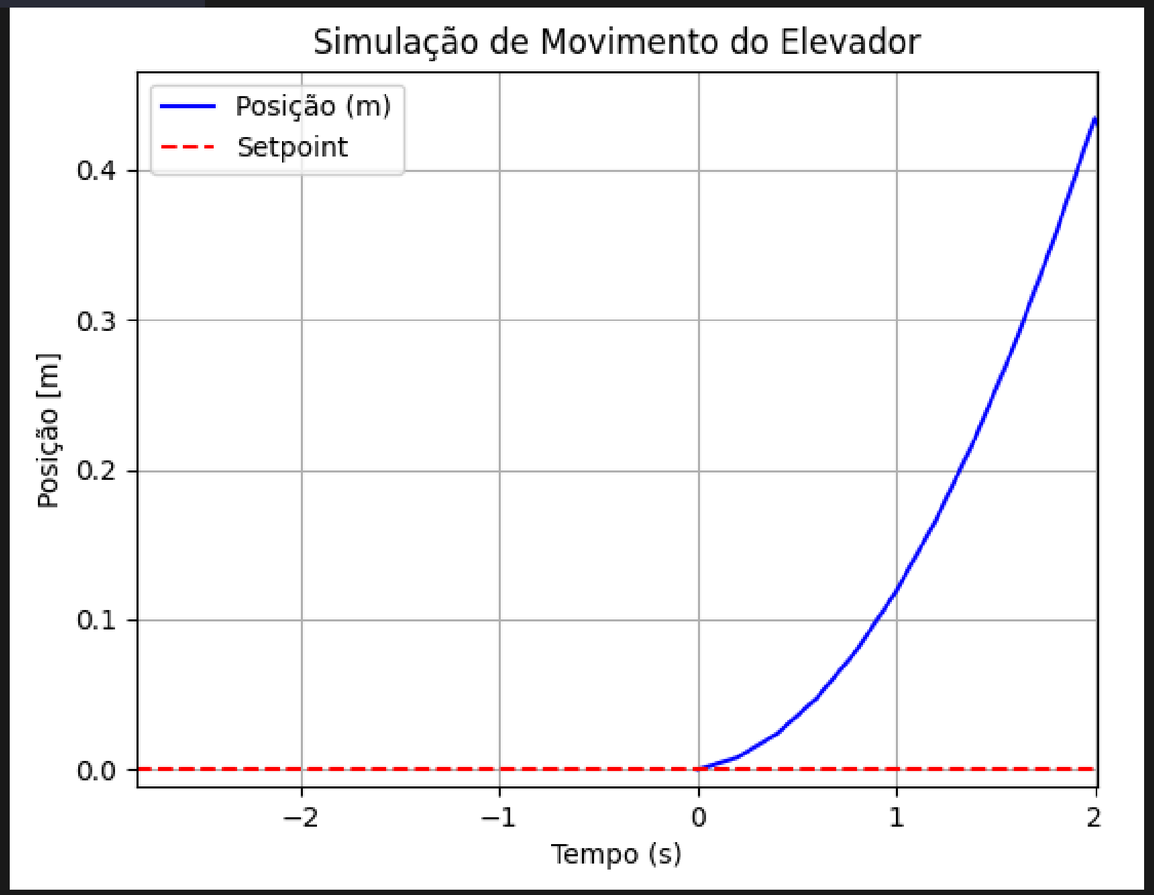
```
# ========== Base de Regras ================
regras = [
    ctrl.Rule(Erro['A'] & DeltaErro['MN'], Pmotor['A']),
    ctrl.Rule(Erro['A'] & DeltaErro['PN'], Pmotor['A']),
    ctrl.Rule(Erro['A'] & DeltaErro['ZE'], Pmotor['M']),
    ctrl.Rule(Erro['A'] & DeltaErro['PP'], Pmotor['M']),
    ctrl.Rule(Erro['A'] & DeltaErro['MP'], Pmotor['B']),

    ctrl.Rule(Erro['M'] & DeltaErro['MN'], Pmotor['A']),
    ctrl.Rule(Erro['M'] & DeltaErro['PN'], Pmotor['A']),
    ctrl.Rule(Erro['M'] & DeltaErro['ZE'], Pmotor['M']),
    ctrl.Rule(Erro['M'] & DeltaErro['PP'], Pmotor['M']),
    ctrl.Rule(Erro['M'] & DeltaErro['MP'], Pmotor['B']),

    ctrl.Rule(Erro['B'] & DeltaErro['MN'], Pmotor['A']),
    ctrl.Rule(Erro['B'] & DeltaErro['PN'], Pmotor['A']),
    ctrl.Rule(Erro['B'] & DeltaErro['ZE'], Pmotor['M']),
    ctrl.Rule(Erro['B'] & DeltaErro['PP'], Pmotor['B']),
    ctrl.Rule(Erro['B'] & DeltaErro['MP'], Pmotor['MB']),

    ctrl.Rule(Erro['MB'] & DeltaErro['MN'], Pmotor['A']),
    ctrl.Rule(Erro['MB'] & DeltaErro['ZE'], Pmotor['MB']),
    ctrl.Rule(Erro['MB'] & DeltaErro['PN'], Pmotor['MB']),
    ctrl.Rule(Erro['MB'] & DeltaErro['PP'], Pmotor['MB']),
    ctrl.Rule(Erro['MB'] & DeltaErro['MP'], Pmotor['MB'])
]
```

Sistema de Controle de Posição e velocidade

```
# ======== Sistema de controle ==========
controle = ctrl.ControlSystem(regras)
simulador = ctrl.ControlSystemSimulation(controle)
```

Regime Inercial

```python
if tempo_total <= tempo_aceleracao:
    # Regime de aceleração linear
    pmotor_saida = pot_max_inercial * (tempo_total / tempo_aceleracao)
    k1 = 1 if erro_atual >= 0 else -1
    pos_atual = k1 * pos_atual * 0.999 + pmotor_saida * k2
```



Atualização do Controle Fuzzy e Dinâmica da Posição do Elevador

```
simulador.input['Erro'] = erro_atual
simulador.input['DeltaErro'] = delta_erro
simulador.compute()

pmotor_saida = simulador.output['Pmotor'] / 100
pos_atual = pos_atual * 0.9995 + (pmotor_saida * 0.212312 * k)
erro_anterior = erro_atual
```

Condição de parada

```python
if erro_atual < 0.05 and abs(delta_erro) < 0.05:
    print("\n\nCondição de parada atingida!!")
    pmotor_saida = 0
    condicao_parada = False
    if grafico:
        # plot com tempo correto
        plt.figure()
        plt.plot(tempos, trajetoria, label='Posição (m)', color='b')
        plt.axhline(pos_setpoint, color='r', linestyle='--', label='Setpoint')
        plt.xlabel('Tempo (s)')
        plt.ylabel('Posição [m]')
        plt.title('Simulação de Movimento do Elevador')
        plt.legend()
        plt.grid(True)
        plt.show()
        grafico = False
```

Envio da dados para a interface via MQTT

```python
# Enviar via MQTT
dados = {"tempo": tempo_total, "posicao": pos_atual}
mensagem = json.dumps(dados)
publicar_mqtt("C213/elevador/trajetoria", mensagem)
```