

# SIMULAÇÃO DE ELEVADOR

com controle fuzzy e comunicação MQTT



## Overview

Nosso objetivo é simular o comportamento realista de um elevador, usando:

- Controle Fuzzy PD para suavidade e precisão;
- Comunicação MQTT para comandos em tempo real;
- Interface gráfica interativa.



# Controle Direto

Velocidade de Movimento é  
diretamente proporcional à Potência do Motor

## Inicialização

```
if self.inicializando:  
    self.tempo_inicial += self.Ts  
    potencia_inicial = min((self.tempo_inicial / 2.0) * 0.315, 0.315)  
  
    self.posicao_atual = (  
        self.k1 * self.posicao_atual * 0.999 + potencia_inicial * self.k2_inicializacao * sentido  
    )
```



# Tempo de inércia

É o tempo que o elevador continua se movendo somente com a força residual, sem atuação ativa do controle fuzzy.

```
else: # Inercia
    erro = self.sp - self.posicao_atual
    delta_erro = erro - self.erro_anterior
    delta_erro = np.clip(delta_erro, -1, 1)

    if self.desligar_fuzzy:
        if self.potencia_inercia is None:
            self.potencia_inercia = self.fuzzy.output.get('PotenciaMotor', 0.25)

        self.potencia_inercia *= 0.90

        self.posicao_atual = (
            self.k1 * self.posicao_atual * 0.9995 + self.potencia_inercia * self.k2_fuzzy * sentido
        )
```

# Fuzzy

```
else: # Controle Fuzzy
    classe_erro = self.classificar_erro(erro)
    classe_delta = self.classificar_delta_erro(delta_erro)

    # Condição de parada suave
    if classe_erro == 'pequeno' and classe_delta == 'variando':
        self.em_fase_parada = True

    self.fuzzy.input['Erro'] = abs(erro)
    self.fuzzy.input['DeltaErro'] = abs(delta_erro)
    self.fuzzy.compute()

    #ajustando a intensidade da força do motor quando chega próximo ao destino
    potencia = self.fuzzy.output.get('PotenciaMotor', 0.25)
    if self.em_fase_parada:
        potencia *= 0.85

    self.posicao_atual = (
        self.k1 * self.posicao_atual * 0.9995 + potencia * self.k2_fuzzy * sentido
    )
```



# Condição de parada

```
# Condição de parada suave
if classe_erro == 'pequeno' and classe_delta == 'variando':
    self.em_fase_parada = True
```

```
# Classificadores para condição de parada
def classificar_erro(self, erro):
    erro_abs = abs(erro)
    if erro_abs < 0.02:
        return 'muito_pequeno'
    elif erro_abs < 0.05:
        return 'pequeno'
    else:
        return 'grande'

def classificar_delta_erro(self, delta_erro):
    delta_abs = abs(delta_erro)
    if delta_abs < 0.005:
        return 'estável'
    elif delta_abs < 0.02:
        return 'variando'
    else:
        return 'instável'
```



# Controle Fuzzy PD

Erro  
Entrada

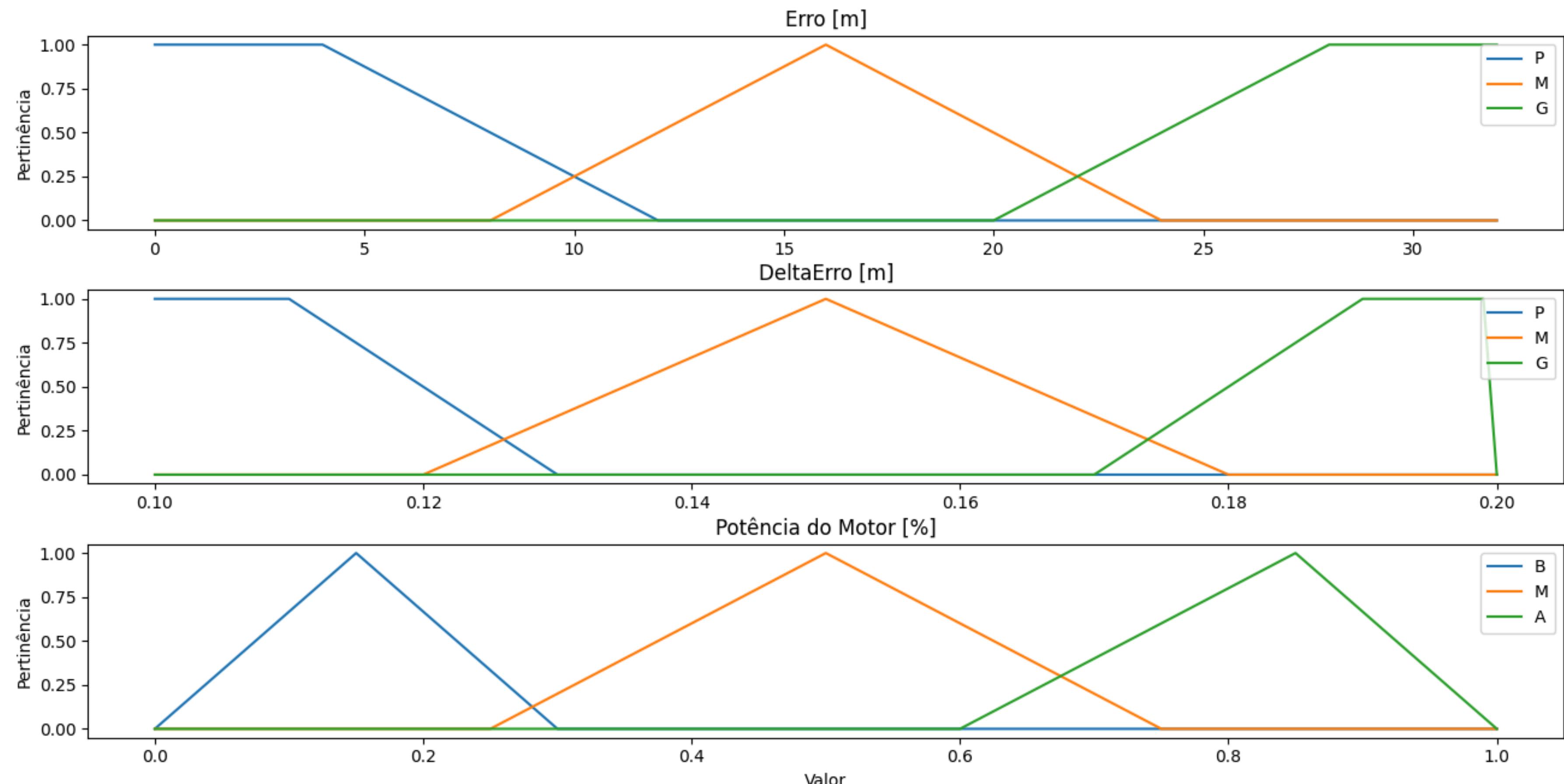
```
Erro['P'] = fuzz.trapmf(Erro.universe, [0, 0, 4, 12])
Erro['M'] = fuzz.trimf(Erro.universe, [8, 16, 24])
Erro['G'] = fuzz.trapmf(Erro.universe, [20, 28, 32, 32])
```

DeltaErro  
Entrada

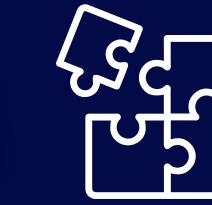
```
DeltaErro['N'] = fuzz.trapmf(DeltaErro.universe, [-10, -10, -3, -1])
DeltaErro['ZE'] = fuzz.trimf(DeltaErro.universe, [-2, 0, 2])
DeltaErro['P'] = fuzz.trapmf(DeltaErro.universe, [1, 3, 10, 10])
```

Potênciac  
Saída

```
PotenciaMotor['B'] = fuzz.trimf(PotenciaMotor.universe, [0.0, 0.15, 0.3])
PotenciaMotor['M'] = fuzz.trimf(PotenciaMotor.universe, [0.25, 0.5, 0.75])
PotenciaMotor['A'] = fuzz.trimf(PotenciaMotor.universe, [0.6, 0.85, 1.0])
```



# Base de Regras



```
regras = [
    ctrl.Rule(Erro['P'] & DeltaErro['P'], PotenciaMotor['M']),
    ctrl.Rule(Erro['P'] & DeltaErro['M'], PotenciaMotor['B']),
    ctrl.Rule(Erro['P'] & DeltaErro['G'], PotenciaMotor['B']),
    ctrl.Rule(Erro['M'] & DeltaErro['P'], PotenciaMotor['A']),
    ctrl.Rule(Erro['M'] & DeltaErro['M'], PotenciaMotor['M']),
    ctrl.Rule(Erro['M'] & DeltaErro['G'], PotenciaMotor['B']),
    ctrl.Rule(Erro['G'] & DeltaErro['P'], PotenciaMotor['A']),
    ctrl.Rule(Erro['G'] & DeltaErro['M'], PotenciaMotor['M']),
    ctrl.Rule(Erro['G'] & DeltaErro['G'], PotenciaMotor['M']),
]
```

# MQTT



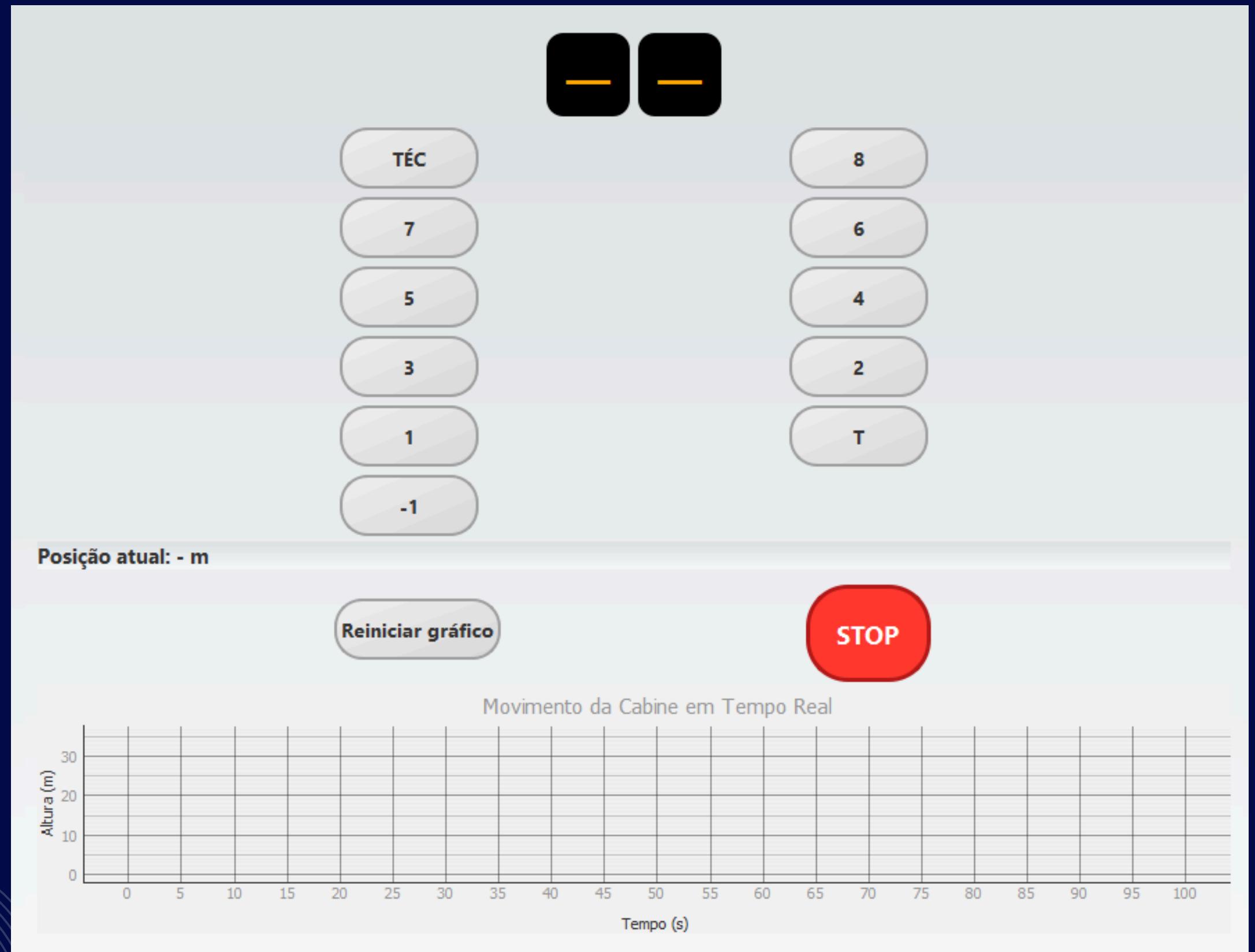
```
def conectar(self, broker="localhost", porta=1883):  
    self.client.connect(broker, porta, 60)  
    self.client.loop_start()
```

- Inserimos um cliente MQTT (novo sistema) se conecta ao broker Mosquitto.
- Ele escuta um tópico (ex: elevador/chamada).
- Quando uma mensagem é publicada nesse tópico, o elevador é acionado para o andar correspondente.

# Interface



Feita em PythonQT5.



# Integração

- Interface envia destino via MQTT
- Simulador processa movimento via fuzzy
- Posição é publicada e GUI se atualiza

```
def on_message(self, client, userdata, msg):  
    try:  
        destino = int(msg.payload.decode())  
        print(f"↗ Andar chamado via MQTT: {destino} m")  
        self.on_chamada_callback(destino)  
    except:  
        print("✗ Erro: mensagem MQTT inválida.")
```

# Resultados



**2** -

TÉC  
7  
5  
3  
1  
-1

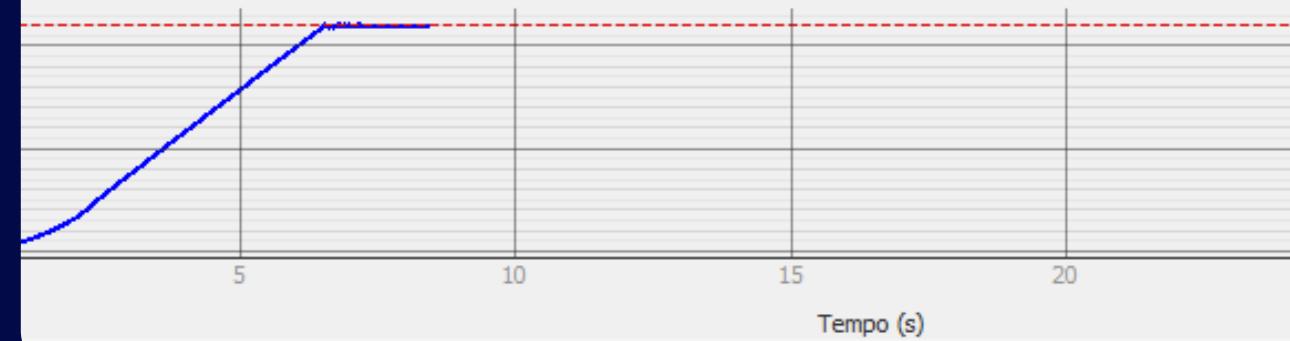
8  
6  
4  
2  
T

o andar 2 (10.98 m)

Reiniciar gráfico

STOP

Movimento da Cabine em Tempo Real



Subida

**2** -

TÉC  
7  
5  
3  
1  
-1

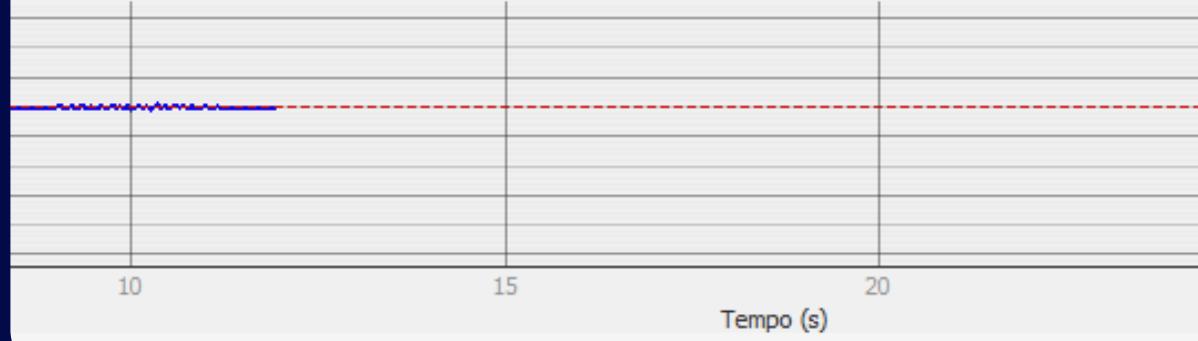
8  
6  
4  
2  
T

o andar 2 (10.97 m)

Reiniciar gráfico

STOP

Movimento da Cabine em Tempo Real



Parado

**T** -

TÉC  
7  
5  
3  
1  
-1

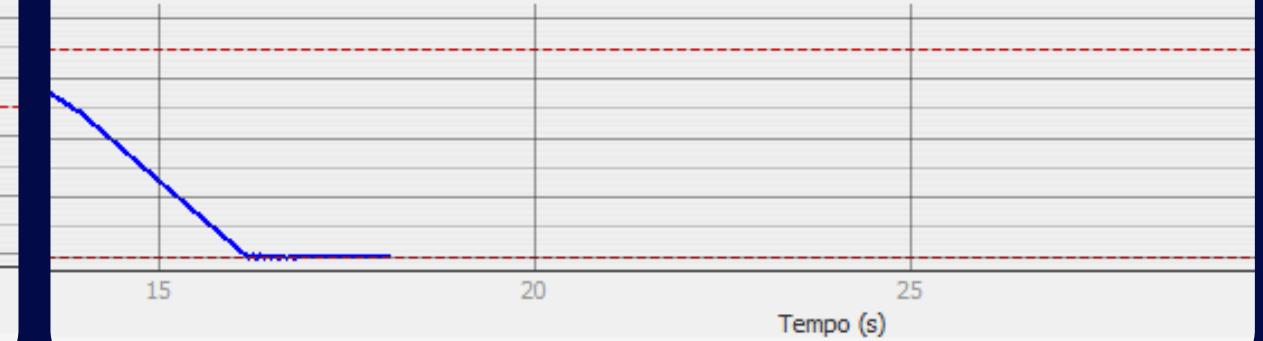
8  
6  
4  
2  
T

o andar T (4.00 m)

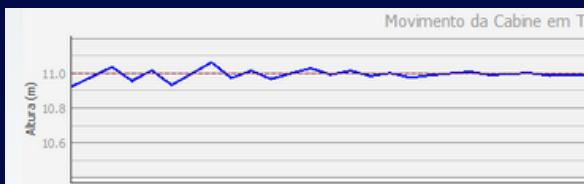
Reiniciar gráfico

STOP

Movimento da Cabine em Tempo Real



Descida





Agradecemos a  
atenção!

*Gerente General*

# SIMULAÇÃO