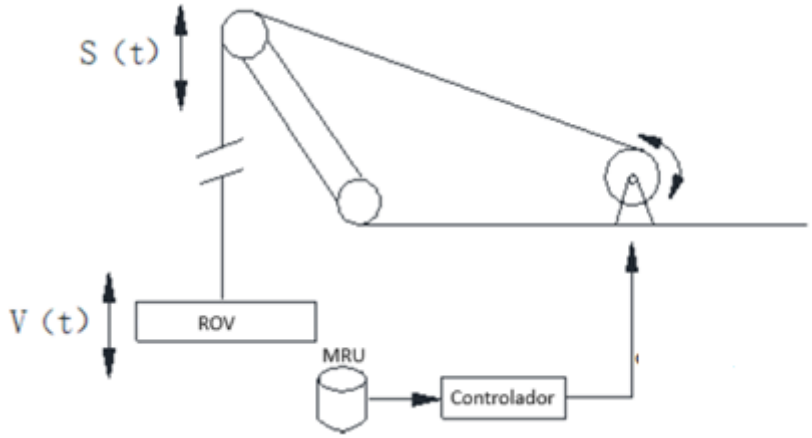
Autor: Matheus de Miranda – Eng. De Controle e Automação/Analista de ROV

Active Heave Compensation (AHC) para ROV – Simulação em Python

1. Introdução



Em operações offshore com ROVs (Remotely Operated Vehicles), o movimento vertical da embarcação causado por ondas — conhecido como *heave* — é transmitido ao veículo pelo umbilical.
Esse movimento pode prejudicar inspeções e intervenções, aumentando riscos e desgaste do sistema.

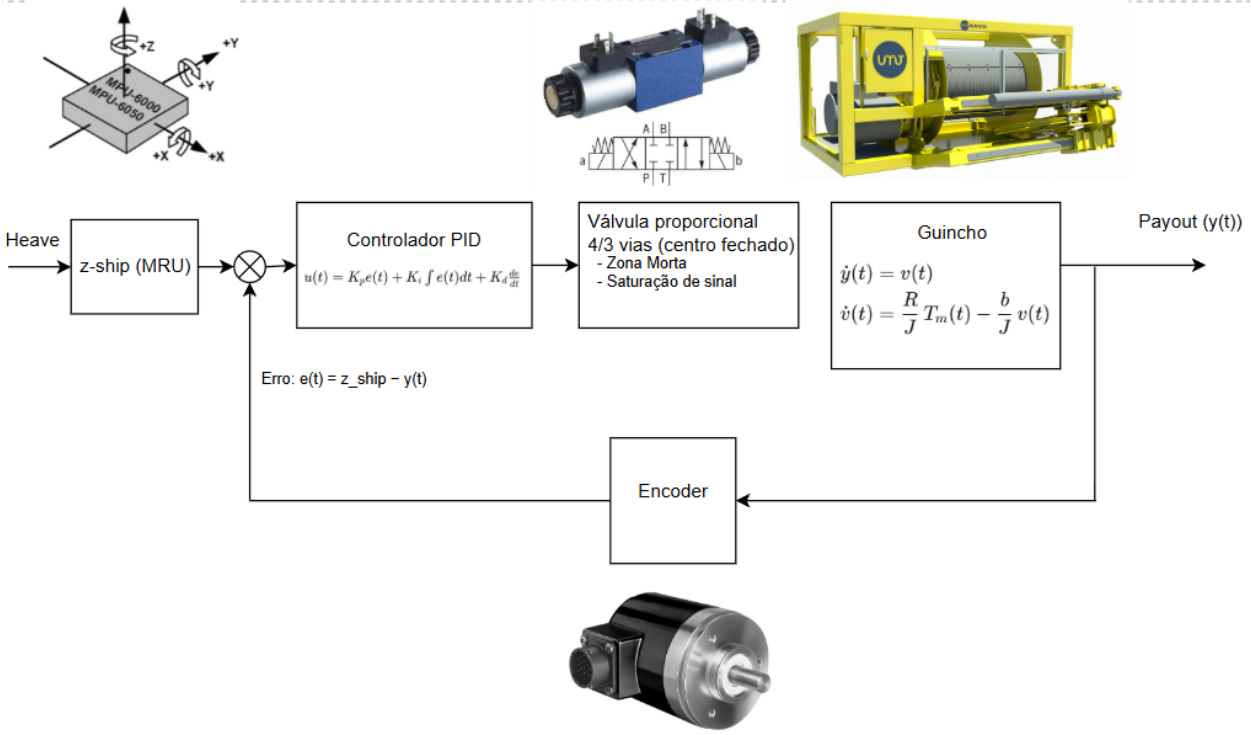
O **Active Heave Compensation (AHC)** é um sistema de controle que atua no guincho do TMS (Tether Management System) para cancelar o efeito do heave, mantendo o ROV estável em relação ao fundo.

Este estudo apresenta um **modelo simplificado** de AHC implementado em **Python**, com ênfase na modelagem de controle e análise de desempenho em diferentes condições de mar.

---

2. Arquitetura do Sistema

O modelo representa a malha de controle de forma simplificada:



- **MRU**: Mede deslocamento vertical do navio.

  O MRU é um sensor inercial de alta precisão usado para medir os movimentos da embarcação, incluindo **heave** (movimento vertical), *pitch* e *roll*.
  No contexto do AHC, ele fornece em tempo real o deslocamento vertical da embarcação, que será usado como **referência** para que o sistema de controle calcule o quanto o guincho deve compensar.

Exemplo de MRU

No modelo, o MRU é representado pelo sinal sintético de heave gerado no Python. Na prática, utiliza acelerômetros e giroscópios com filtragem para estimar o movimento.

- **Controlador PID**: Calcula o comando de torque para compensar o heave.

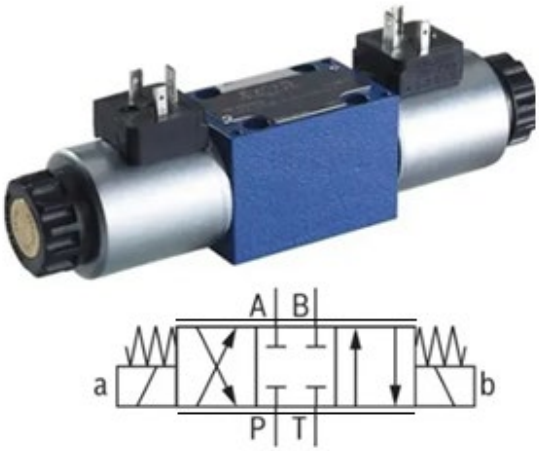
  O controlador **Proporcional–Integral–Derivativo** compara a posição medida do guincho (encoder) com a posição de referência (MRU) e calcula o comando necessário para reduzir o erro.

  P (Proporcional): reage proporcionalmente ao erro atual.

  I (Integral): corrige erros acumulados no tempo.

  D (Derivativo): reage à taxa de variação do erro, antecipando mudanças.
  No modelo, o PID produz uma saída elétrica simulada (comando) que representa a abertura da válvula proporcional.

- **Válvula proporcional 4/3 Vias acionada por solenoide**: Converte comando elétrico em torque hidráulico, com zona morta e saturação.



  Elemento eletro-hidráulico que converte um sinal elétrico em vazão de óleo para o motor hidráulico do guincho.

  Zona morta (*dead zone*): pequena faixa de comando na qual não há movimento, representando folgas e ineficiências mecânicas/hidráulicas.

  Saturação: limite máximo de torque ou vazão que a válvula consegue fornecer, impedindo atuação além da capacidade física.

No modelo, o sinal do PID passa primeiro pela zona morta e depois é limitado por saturação de torque.

- **Guincho**: Responsável por enrolar ou desenrolar o cabo armado para compensar o movimento vertical transmitido ao ROV+TMS.

- **Encoder**: Sensor que mede a posição angular ou linear do cabo no guincho.

Fornece feedback em tempo real para o controlador PID fechar a malha de controle.

No modelo, a posição integrada (θ\thetaθ) da velocidade do guincho faz o papel do encoder.

Encoders usados em AHC precisam ter:

- **Alta resolução** (milhares de pulsos por volta) para detectar pequenas variações de posição.
- **Baixa latência** de leitura, garantindo resposta rápida do sistema.
- **Alta confiabilidade**, pois falhas no encoder podem levar à perda de compensação e risco à operação.

  Podem ser do tipo:

- **Incremental** (mede deslocamento relativo e requer referência inicial).
- **Absoluto** (mede posição absoluta sem necessidade de homing).

  Em guinchos de AHC, é comum usar **encoders absolutos** magnéticos ou ópticos, às vezes redundantes, para evitar paradas por falhas.

---

3. Modelagem Matemática

O guincho foi modelado como:

$$\dot{\theta}(t) = \omega(t)$$

$$\dot{\omega}(t) = \frac{T_m(t) - b\,\omega(t)}{J}$$

Onde:

- $\theta(t)$ = posição angular do tambor [rad]
- $\omega(t)$ = velocidade angular do tambor [rad/s]
- $T_m(t)$ = torque do motor após zona morta e saturação [N·m]
- J = inércia equivalente do tambor [kg·m^2]
- b = atrito viscoso [N·m·s/rad]

O **PID** é implementado como:

$$u(t) = K_p e(t) + K_i \int e(t)\,dt + K_d \frac{de(t)}{dt}$$

Não linearidades:

- **Zona morta**: u<0.1⇒Tm=0u < 0.1 \Rightarrow T_m = 0u<0.1⇒Tm=0
- **Saturação**: |Tm|≤100 Nm|

Modelagem do Heave

O heave h(t)h(t)h(t) foi gerado com funções senoidais acrescidas de ruído gaussiano, conforme o cenário:

Mar calmo

$$h_{\text{calmo}}(t) = 1,0 \cdot \sin(2\pi \cdot 0,1 \cdot t) \; + \; 0,02 \cdot \mathcal{N}(0,1)$$

- Amplitude: 1,0 m
- Frequência: 0,1 Hz
- Ruído: desvio padrão de 0,02 m

Mar médio

$$h_{\text{medio}}(t) = 1,5 \cdot \sin(2\pi \cdot 0,2 \cdot t) \; + \; 0,05 \cdot \mathcal{N}(0,1)$$

- Amplitude: 1,5 m
- Frequência: 0,2 Hz
- Ruído: desvio padrão de 0,05 m

Mar agitado

$$\begin{aligned} h_{\text{agitado}}(t) = {} & 2,0 \cdot \sin(2\pi \cdot 0,25 \cdot t) \quad (\text{swell}) \\ & + \; 0,5 \cdot \sin(2\pi \cdot 1,5 \cdot t) \quad (\text{rajadas}) \\ & + \; 0,2 \cdot \mathcal{N}(0,1) \quad (\text{ruído}) \end{aligned}$$

- Swell de baixa frequência + rajadas de alta frequência
- Ruído: desvio padrão de 0,2 m

---

4. Implementação em Python

O código foi escrito em Python 3.8, utilizando:

- **NumPy**: manipulação de arrays.
- **SciPy**: integração numérica (solve_ivp).
- **Matplotlib**: geração de gráficos.

Estrutura principal:

```
class PID:
    def __init__(self, kp, ki, kd): ...
    def compute(self, error, dt): ...

def gerar_heave(t, tipo): ...

def modelo_ahc(t_now, y, ref_signal): ...

sol = solve_ivp(modelo_ahc, [0, t_final], y0, args=(heave,))
```

Parâmetros:

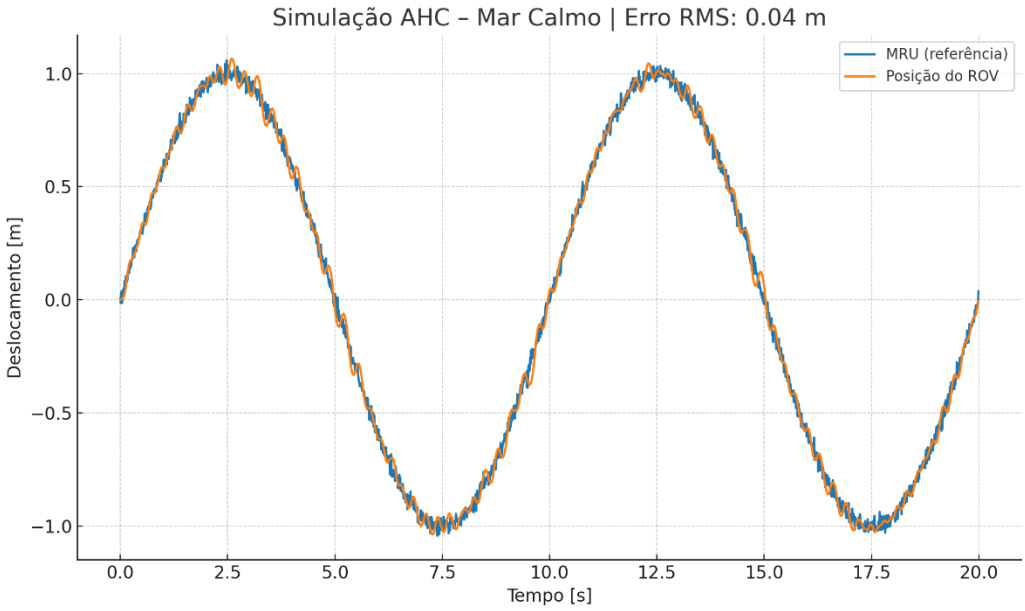
$$K_p = 5 \times 10^4$$
$$K_i = 2 \times 10^3$$
$$K_d = 1 \times 10^4$$
$$J = 5.0,\ b = 2.0$$

---

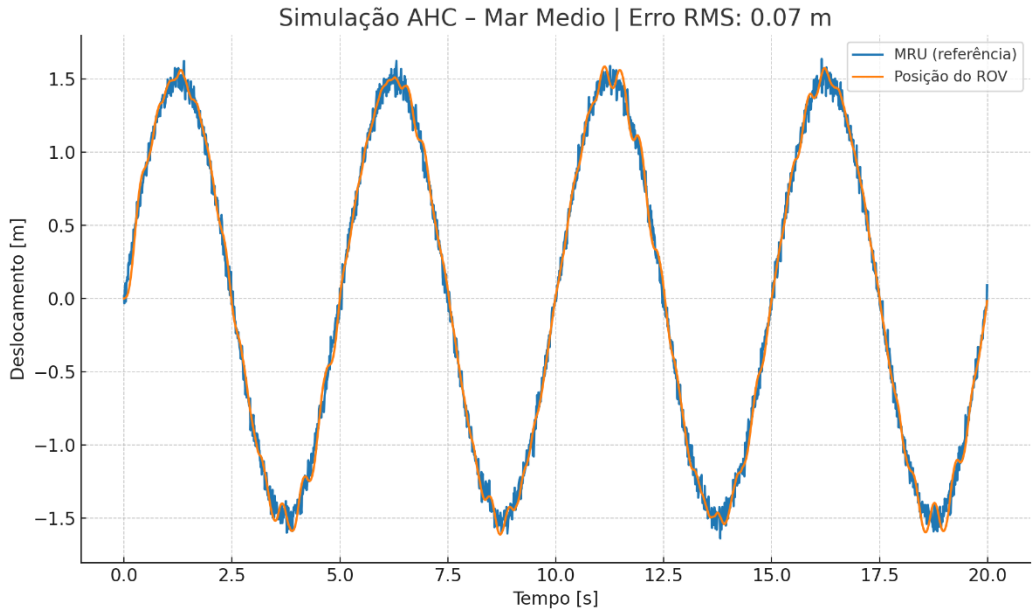
5. Resultados

Gráficos:

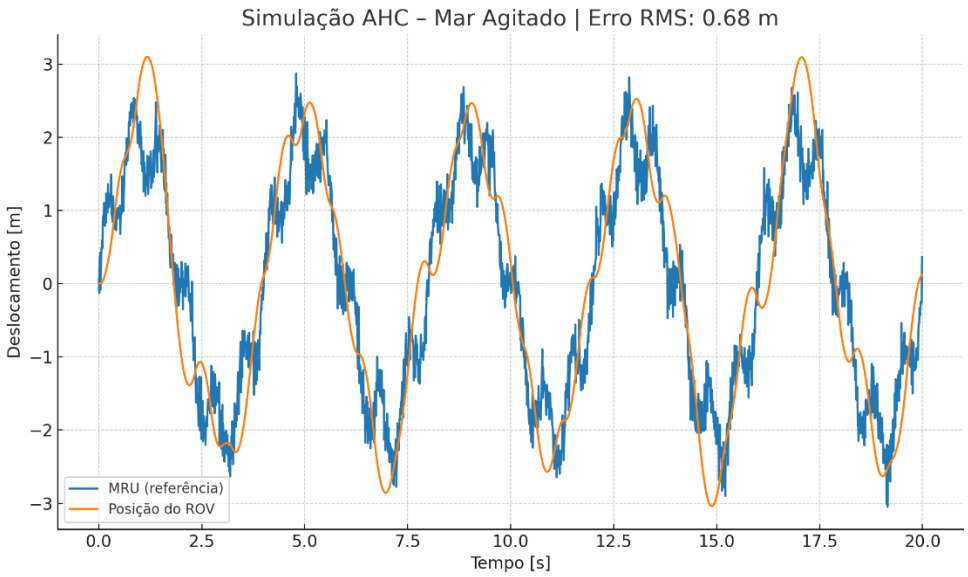
- **Mar calmo**: Rastreamento quase perfeito, controle suave.



- **Mar médio**: Pequeno atraso, mas desempenho estável.



- **Mar agitado**: Limitação de torque e zona morta reduzem desempenho.



---

6. Conclusão

O estudo demonstra como o **Python** pode ser usado para:

- Implementar modelos de controle de sistemas reais (ROV + AHC).
- Simular cenários e analisar impacto de parâmetros mecânicos e de controle.
- Visualizar resultados de forma clara e rápida.

Apesar de simplificado, o modelo permite explorar:

- Efeito de saturação e zona morta no desempenho.
- Sintonia de PID.
- Testes de robustez em condições extremas.

7. Código da simulação e do controlador

O script ahc_simulacao.py foi escrito em **Python 3.8+** e utiliza três bibliotecas principais:

- **NumPy** → operações matemáticas e geração de sinais.
- **SciPy** → integração numérica das equações diferenciais (solve_ivp).
- **Matplotlib** → visualização gráfica dos resultados.

7.1 Definição dos parâmetros do sistema

Logo no início, são definidos os parâmetros físicos e hidráulicos simplificados:

```
D = 50e-6         # Deslocamento volumétrico [m^3/rad]
K_valve = 1e7     # Ganho hidráulico simplificado (u -> torque)
J = 5.0           # Inércia [kg.m²]
b = 2.0           # Atrito viscoso [N.m.s/rad]
deadzone = 0.1    # Zona morta
dt = 0.01
t_final = 20.0
t = np.arange(0, t_final, dt)
y0 = [0.0, 0.0]  # posição, velocidade
```

- **D** representa o deslocamento volumétrico do motor hidráulico por radiano.
- **K_valve** é o ganho que converte o sinal de controle em torque.
- **J** e **b** modelam o guincho como um sistema de segunda ordem.
- **deadzone** implementa a faixa em que o comando não gera movimento.
- **dt** é o intervalo de tempo da simulação.
- **t_final** define a duração total.

7.2 Geração do sinal de heave

A função gerar_heave cria a referência que simula o movimento vertical da embarcação.

```python
def gerar_heave(t, tipo="calmo", seed=42):
    rng = np.random.default_rng(seed)
    if tipo == "calmo":
        return 1.0 * np.sin(2 * np.pi * 0.1 * t) + 0.02 * rng.standard_normal(len(t))
    elif tipo == "medio":
        return 1.5 * np.sin(2 * np.pi * 0.2 * t) + 0.05 * rng.standard_normal(len(t))
    elif tipo == "agitado":
        swell = 2.0 * np.sin(2 * np.pi * 0.25 * t)
        ruido = 0.2 * rng.standard_normal(len(t))
        rajada = 0.5 * np.sin(2 * np.pi * 1.5 * t)
        return swell + ruido + rajada
    else:
        raise ValueError("tipo de mar inválido")
```

- Usa funções **senoidais** para gerar ondas e rng.standard_normal para ruído gaussiano.
- Cada cenário tem **amplitude**, **frequência** e **nível de ruído** diferentes.
- O parâmetro seed garante reprodutibilidade.

7.3 Classe do controlador PID

O controle foi encapsulado na classe PID:

```python
class PID:
    def __init__(self, kp, ki, kd):
        self.kp, self.ki, self.kd = kp, ki, kd
        self.int_error = 0.0
        self.prev_error = 0.0

    def compute(self, error, dt):
        self.int_error += error * dt
        derivative = (error - self.prev_error) / dt
        self.prev_error = error
        return self.kp*error + self.ki*self.int_error + self.kd*derivative
```

- **kp, ki, kd** → ganhos proporcional, integral e derivativo.
- int_error acumula o erro ao longo do tempo (parte integral).
- prev_error armazena o erro anterior (para cálculo derivativo).
- compute retorna a ação de controle considerando as três parcelas.

7.4 Função de modelo dinâmico

A função modelo_ahc implementa as equações diferenciais do guincho e a lógica da válvula proporcional.

```python
def modelo_ahc(t_now, y, ref_signal):
    theta, omega = y
    idx = min(int(t_now / dt), len(ref_signal) - 1)
    error = ref_signal[idx] - theta
    u = pid.compute(error, dt)
    if abs(u) < deadzone:
        torque = 0.0
    else:
        torque = D * K_valve * (u - np.sign(u)*deadzone)
        torque = np.clip(torque, -100.0, 100.0)  # saturação
    domega = (torque - b * omega) / J
    dtheta = omega
    return [dtheta, domega]
```

- theta = posição do guincho (m), omega = velocidade (m/s).
- error = diferença entre referência (MRU) e posição atual (encoder).
- **Zona morta**: comandos pequenos são anulados.
- **Saturação**: torque limitado a ±100 N·m.
- Retorna derivadas para integração pelo solver.

7.5 Integração e execução

Define o cenário e roda a simulação:

```python
cenario = "calmo"  # "calmo", "medio" ou "agitado"

heave = gerar_heave(t, cenario)
sol = solve_ivp(modelo_ahc, [0, t_final], y0, t_eval=t, args=(heave,), method='RK45')
theta = sol.y[0]
```

- solve_ivp integra o sistema usando o método de Runge-Kutta de 5ª ordem (RK45).
- y0 = estado inicial [posição, velocidade].

7.6 Plotagem dos resultados

```
plt.figure(figsize=(10, 6))
plt.plot(t, heave, label="MRU (referência)")
plt.plot(t, theta, label="Posição do ROV")
plt.title(f"Simulação AHC – Mar {cenario.capitalize()}")
plt.xlabel("Tempo [s]")
plt.ylabel("Deslocamento [m]")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

- Mostra a referência (MRU) e a resposta do sistema (posição do guincho/ROV).
- Título adaptado ao cenário.
- Configuração de grade, rótulos e legenda para facilitar interpretação.

8. Considerações Finais

A simulação apresentada demonstra que mesmo um modelo simplificado é capaz de capturar os principais aspectos do controle de *Active Heave Compensation* (AHC) aplicado a operações com ROV. A utilização do **Python** como ferramenta de modelagem e análise se mostrou adequada para:

- **Prototipagem rápida** de sistemas de controle, permitindo ajustar parâmetros e observar resultados de forma imediata.
- **Visualização clara** do impacto das condições de mar e das limitações físicas (zona morta, saturação de torque, inércia) no desempenho do sistema.
- **Exploração de cenários extremos**, fundamentais para avaliar a robustez do controle e antecipar situações críticas que podem ocorrer no ambiente offshore.

O estudo reforça a importância da **modelagem matemática** como etapa inicial no desenvolvimento e teste de soluções de AHC. Mesmo que não substitua ensaios físicos ou simulações mais complexas (com dinâmica completa do ROV, atraso de comunicação e hidráulica detalhada), este tipo de abordagem contribui para:

- **Treinamento** de equipes técnicas, proporcionando entendimento da lógica de funcionamento do AHC.

- **Suporte a decisões de projeto**, permitindo comparar estratégias de controle antes de investir em implementação física.
- **Integração com outras disciplinas**, como instrumentação, automação hidráulica e sistemas embarcados.

Como trabalhos futuros, é possível expandir este modelo para incluir:

- Controle *feedforward* baseado em previsão de ondas.
- Modelagem hidráulica detalhada e resposta de válvulas reais.
- Integração com sensores simulados de MRU e encoders com ruído.
- Testes *Hardware-in-the-Loop* (HIL) com controladores físicos.

Com isso, a simulação serve não apenas como exercício acadêmico, mas também como base para desenvolvimento de soluções mais avançadas e adaptadas às necessidades operacionais da indústria offshore.