Controle por Acelerômetro (MPU/BNO085) - Plataforma de Stewart

Visão Geral

`accelerometer.html` + `accelerometer.js` fornecem uma interface leve para controlar roll/pitch/yaw da plataforma usando dados de um MPU-6050 ou BNO085. O frontend mostra o sensor em tempo real, calcula as medidas dos pistões sem renderização 3D e, quando autorizado, envia comandos diretos ao backend FastAPI (`/mpu/control`) para aplicar o movimento no hardware.

Características

- Indicadores de conexão serial, porta ativa e botão de recalibração (`recalibra`) para alinhar o sensor.
- Painel "Controle Ativo" com toggle principal e slider de sensibilidade (0–100% → escala 0.0–1.0 usado nos ângulos).
- Exibição contínua de Roll/Pitch/Yaw com barras percentuais (clamp ±5°) e, quando presente, quaternion completo do BNO085.
- Pré-visualização das medidas dos seis pistões (cores verde/vermelho conforme validade), calculada via `/calculate`.
- WebSocket dedicado (`WS_URL`) com heartbeat, reconexão automática e filtro para mensagens `telemetry_mpu` ou `telemetry_bno085`.
- Aplicação opcional da pose no hardware: os comandos só são enviados ao ESP32 se `controlEnabled` estiver marcado e a serial conectada.

Arquitetura

Backend (`interface/backend/app.py`)

- `POST /serial/ports`, `/serial/open`, `/serial/close`, `/serial/status`: gerenciamento da porta USB (reutilizado de `common.js`).
- `POST /serial/send`: recebe `{ "command": "recalibra" }` para disparar a rotina de calibração no ESP32.
- `POST /calculate`: calcula os comprimentos absolutos e percentuais dos atuadores para a pose estimada (visualização no painel de pistões).
- `POST /mpu/control`: aplica a pose no hardware (roll/pitch/yaw limitados, x/y fixos, z = 530 mm por padrão, além do parâmetro `scale`).
- WebSocket `/ws/telemetry`: envia mensagens com `type=telemetry_mpu` ou `telemetry_bno085`, contendo `mpu` (roll/pitch/yaw) e, quando disponível, `quaternions`.

Frontend (`interface/frontend/scripts/accelerometer.js`)

- Constantes importantes:
  - `DEFAULT_Z_HEIGHT = 530` mm (pose neutra).
  - `WS_UPDATE_INTERVAL` (~30 FPS) e `CONTROL_UPDATE_INTERVAL` (~10 Hz) para limitar chamadas.
  - `limitAngles()` restringe os ângulos entre –5° e +5°.
- `updateMPUDisplay` e `updateQuaternionDisplay` mantêm o painel numérico/barras sincronizado.

- `calculateKinematicsFromMPU(mpu)`:
    1. Limita ângulos recebidos.
    2. Aplica a escala definida pelo slider.
    3. Limita novamente.
    4. Monta a pose (X/Y=0, Z=DEFAULT_Z_HEIGHT).
    5. Chama `/calculate` e atualiza `currentPlatformData` + `updatePistonMeasures`.
- `sendMPUControl(mpu)`:
    - Respeita `controlEnabled`, `serialConnected` e `CONTROL_UPDATE_INTERVAL`.
    - Limita/escala ângulos e envia `POST /mpu/control` (JSON com pose + scale).
- `initTelemetryWS()`:
    - Garante que apenas um socket exista, limpa timers e abre `WS_URL`.
    - Implementa heartbeat (3 s) e reconexão se ficar >5 s sem mensagens.
    - Processa somente `telemetry_mpu`/`telemetry_bno085`, acionando `updateMPUDisplay`, `updateQuaternionDisplay`, `calculateKinematicsFromMPU` e `sendMPUControl` (quando ativo).
- `checkExistingConnection()`:
    - Consulta `/serial/status`, atualiza LED, botões e select.
    - Reabre o WebSocket, mostra toast e aplica a pose neutra (530 mm) via `/mpu/control`.
- `DOMContentLoaded` flow:
    1. Faz ping em `/serial/ports` para verificar backend.
    2. Executa `loadSerialPorts()` e `checkExistingConnection()`.
    3. Chama `calculateKinematicsFromMPU` com ângulos zero (preview inicial).
    4. Inicia `initCommonSerialControls()`, listeners de recalibração, toggle e slider.
    5. Sobrescreve `window.initTelemetryWS` para usar a versão local.

Painéis da Página

1. **Status da Conexão** – LED com animação “pulse”, texto e porta atualizados em tempo real.
2. **Conexão Serial** – select de portas, botões Atualizar/Conectar/Desconectar (reutiliza estilos e lógica de `common.js`).
3. **Controle Ativo & Sensibilidade** – toggle “Controle Ativo”, slider `scale`, display percentual e botão “Recalibrar Referência”.
4. **Dados do Sensor** – valores numéricos de roll/pitch/yaw (+ barras progressivas) e, se disponível, card com quaternion `w/x/y/z`.
5. **Medidas dos Pistões** – grade com `preview-piston-1..6`, cores variando conforme `valid`.
6. **Checklist / Mensagens** – toasts informam backend offline, tentativa de controle sem serial, etc.

Fluxo de Funcionamento

1. Usuário abre `accelerometer.html`.
2. Scripts carregam portas, verificam conexão existente e calculam uma pose inicial (sem movimentar a plataforma).
3. WebSocket passa a receber telemetria do MPU:
    - Roll/Pitch/Yaw aparecem no painel.
    - Os valores são limitados, escalados e usados para calcular os comprimentos dos pistões (preview).
4. Ao marcar “Controle Ativo”:

    - Cada atualização válida do sensor passa por `sendMPUControl`.
    - A função respeita o `CONTROL_UPDATE_INTERVAL` (~10 Hz) antes de chamar `/mpu/control`.
5. Botão “Recalibrar Referência” dispara `/serial/send` com `recalibra`, exibindo feedback via `showToast`.
6. Ao desconectar a serial ou sair da página, o toggle é desativado automaticamente (listener `beforeunload`).

Passo a Passo de Uso

1. Rodar o backend (`python app.py`) e abrir `interface/frontend/accelerometer.html`.
2. Conectar o ESP32:
    - Clique em “Atualizar”, escolha a porta e pressione “Conectar”.
    - Confirme que o LED ficou verde (`Conectado` + porta exibida).
3. Ativar o sensor:
    - Verifique se Roll/Pitch/Yaw estão sendo recebidos (valores diferentes de `--`).
    - Se necessário, pressione “Recalibrar Referência” para zerar offsets.
4. Ajustar a sensibilidade pelo slider (100% = escala 1.0).
5. Marcar “Controle Ativo”.
    - Se aparecer aviso de serial desconectada, reconecte antes de prosseguir.
6. Monitorar o painel de pistões e, se algum ficar vermelho, reduza os ângulos ou diminua a escala.
7. Desmarcar o toggle antes de desconectar ou encerrar o backend.

Configurações Avançadas

- **Limites de ângulos**: edite `limitAngles()` para ampliar ou reduzir o envelope (ex.: ±8° para roll/pitch).
- **Sensibilidade**: mude o cálculo do slider (`scale = value / 100`) para suportar até 200% (basta alterar `max="200"` e o divisor).
- **Altura neutra**: ajuste `DEFAULT_Z_HEIGHT` caso a plataforma tenha h0 diferente de 530 mm.
- **Taxas de atualização**: personalize `WS_UPDATE_INTERVAL` e `CONTROL_UPDATE_INTERVAL` para balancear responsividade x tráfego serial.
- **Feedback numérico**: `updateCount` e `lastRateCheck` podem ser reutilizados para mostrar Hz efetivos (trecho comentado no HTML).
- **Isolamento das mensagens WebSocket**: a função `initTelemetryWS` pode ser adaptada para conviver com outros handlers usando um dispatcher central (ex.: em `ws-utils.js`).

Troubleshooting

- **Painel mostra “Backend offline”**:
    - Confira o terminal do FastAPI; a página testa `/serial/ports` logo ao carregar e mostra toast de aviso se falhar.
    - Verifique se a porta TCP 8001 está disponível ou ajuste `API_BASE`.
- **Controle não aplica no hardware**:
    - Certifique-se de que `controlEnabled` está marcado e que `serialConnected` é `true`.
    - Observe o console: `sendMPUControl` registra mensagens quando sai cedo por throttle ou falta de serial.
- **Valores de roll/pitch travados**:
    - O WebSocket pode ter desconectado — veja se o log exibe “WebSocket sem mensagens há X s - reconectando”.

    - Clique em “Atualizar” e reconecte a porta para reabrir a telemetria.
- **Pistões sempre vermelhos**:
    - Os ângulos podem estar extrapolando o limite mesmo após o clamp — reduza a escala ou revise `DEFAULT_Z_HEIGHT`.
    - Confira se `/calculate` está retornando `valid=false` (mensagens aparecem no console).
- **Botão “Recalibrar” não faz nada**:
    - A serial precisa estar conectada; o código mostra toast “Conecte-se primeiro...” caso contrário.
    - Veja logs do backend para garantir que o firmware reconhece o comando `recalibra`.

Autor

Guilherme Miyata - Instituto Federal de São Paulo (IFSP)
Trabalho de Conclusão de Curso - 2025

Github
Linkedin
Portfólio

Última atualização: Novembro 2025