Painel PID & Atuadores - Plataforma de Stewart

Visão Geral

`actuators.html` reúne tudo o que é necessário para operar e depurar o controle PID de cada pistão: conexão serial, ajustes de setpoint (global e individual), monitoramento em tempo real, sintonização de ganhos/offsets/feedforward e comandos manuais direto para o ESP32. Os scripts `actuators.js` e `chart-utils.js` trabalham juntos para exibir telemetria via WebSocket (`/ws/telemetry`) e registrar dados em IndexedDB.

Características

- Cartões de status e conexão serial padronizados (lista automática de portas, botões Conectar/Desconectar).
- Gráfico interativo (Chart.js) com até 6 pares "posição atual x setpoint" + gravação em IndexedDB e exportação CSV.
- Controles de setpoint global (todos os pistões) e individual, com histórico no console lateral.
- Envio de parâmetros PID: ganhos individuais, ganhos globais, feedforward (avanço/recuo) e offsets de curso.
- Comandos manuais rápidos (`/pid/manual/A|R|ok`) para testes ponto a ponto.
- Console integrado mostrando TX/RX e alertas, além de painel com últimos valores de telemetria.
- Reconexão automática do WebSocket enquanto `serialConnected` permanecer verdadeiro.

Arquitetura

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

- Endpoints seriais: `GET /serial/ports`, `POST /serial/open`, `POST /serial/close`, `GET /serial/status`.
- Telemetria: WebSocket `/ws/telemetry` envia `Y` (posições), `sp_mm`, `mpu`, mensagens `raw`.
- PID:
  - `POST /pid/setpoint` – recebe `PIDSetpoint` (global e individual).
  - `POST /pid/gains`, `GET /pid/gains`, `GET /pid/gains/{piston}` – leitura/ajuste por pistão.
  - `POST /pid/gains/all` – aplica mesmos ganhos em todos os pistões.
  - `POST /pid/feedforward` e `/pid/feedforward/all` – U0 de avanço/recuo.
  - `POST /pid/settings` / `GET /pid/settings` – `dbmm` e `minpwm`.
  - `POST /pid/offset` e `/pid/offset/all` – correções de referência.
  - `POST /pid/manual/{action}` – ações discretas (A=avanço, R=recuo, ok=parar).
  - `POST /pid/select/{piston}` – seleciona pistão ativo para comandos manuais/offsets.

Frontend (`actuators.html` + `actuators.js` + `chart-utils.js`)

- `chart-utils.js`:
  - `initDB()` cria `TelemetryDB.telemetry`. 4 functions manage Chart data, recording, toggles, export.
- `actuators.js`:

- `initLocalTelemetryWS()` abre WebSocket, monitora heartbeat (5 s) e chama `handleTelemetry`.
- `handleTelemetry(data)` atualiza UI textual (`telem-sp`, `telem-yX`), injeta os dados no gráfico (via `updateChart`), registra log RX e calcula toasts.
- `sendCommand`/`sendFreeCommand` auxiliam o terminal manual (enviando strings para `/serial/send`).
- `sendSetpointGlobal`, `sendSetpointIndividual` chamam `/pid/setpoint` e atualizam `currentSetpoints`.
- `sendPIDParams`, `sendFeedforward`, `sendOffsets` disparam para os endpoints correspondentes usando valores da UI.
- `manualAdvance/Retact/Stop` usam `/pid/manual/A|R|ok`.
- `checkExistingConnection()` retoma sessão aberta, atualiza select, guarda porta em `localStorage` e chama `initLocalTelemetryWS`.
- `updateConnectionStatus()` roda periodicamente para manter LED e texto atualizados.
- Ao carregar a página: inicializa IndexedDB + gráfico, aplica `initCommonSerialControls()`, inicia WebSocket e verifica serial.

Seções da Página

1. **Status + Conexão Serial** – mesmo layout usado nas demais telas (select, atualizar, conectar, desconectar).
2. **Gráfico de Telemetria** – canvas principal com botões:
   - Iniciar/Pausar gravação.
   - Limpar dados.
   - Resetar zoom.
   - Exportar CSV.
   - Checkboxes por pistão para mostrar/ocultar curvas de posição e setpoint.
3. **Setpoints**:
   - Controle global: define `sp_mm` para todos de uma vez (slider + input).
   - Controle individual: inputs e botões por pistão.
4. **PID**:
   - Campos de Kp/Ki/Kd + feedforward (u0 avanço/recuo) por pistão.
   - Botões para aplicar nos seis simultaneamente.
5. **Offsets e Seleção de Pistão** – define offsets em mm e seleciona qual eixo está “armado” para comandos manuais.
6. **Controles Manuais** – botões Avançar / Recuar / Parar e campo para enviar comandos livres supervisionados.
7. **Console & Telemetria Instantânea** – área tipo terminal que lista TX/RX/time e pequenos cards com os últimos valores Y1…Y6, SP global e status da gravação.

Fluxo de Funcionamento

1. Usuário conecta a porta → `initLocalTelemetryWS()` abre o WebSocket e começa a receber `telemetry`.
2. Cada mensagem:
   - Preenche `telem-yX` e `telem-sp`.
   - `updateChart()` adiciona um ponto (posições reais + setpoints guardados).

    - Se gravação estiver ativa (`chartRecording=true`), os pontos também vão para o IndexedDB.
3. Operações do painel enviam requisições REST:
    - Setpoints e ganhos usam JSON (`fetch` com body).
    - Feedforward “All” e setpoint global aproveitam query strings simples.
    - Comandos manuais/offsets logam no console com tag `[TX]` ou `[INFO]`.
4. “Exportar CSV” consulta IndexedDB completo e baixa o arquivo `telemetria_<timestamp>.csv`.
5. Ao sair ou trocar de página, `beforeunload` para e limpa o gráfico para evitar consumo de memória.

Passo a Passo de Uso

1. Rodar o backend (`python app.py`) e abrir `actuators.html`.
2. Conectar a porta serial via seção “Conexão Serial”.
3. Iniciar o gráfico (botão “Iniciar Gravação”) se quiser histórico.
4. Ajustar setpoint global ou individual conforme o teste desejado.
5. Refinar PID:
    - Edite Kp/Ki/Kd dos pistões e clique em “Aplicar”.
    - Caso queira usar o mesmo valor em todos, utilize o bloco “Aplicar em Todos”.
6. Enviar offsets/feedforward caso detecte desvios.
7. Acionar “Avançar/Recuar/Parar” para testar movimentos discretos.
8. Exportar CSV para análise offline quando terminar.

Configurações Avançadas

- **Janela do gráfico**: altere `maxDataPoints` (em `chart-utils.js`) para mais/menos pontos visíveis. Lembre-se que valores altos consomem mais CPU.
- **Persistência**: `TelemetryDB` fica no navegador. Limpe manualmente caso queira zerar sem usar “Limpar”.
- **Limiares de reconexão**: `heartbeatTimer` em `actuators.js` dispara reconexão após 5 s sem mensagens; ajuste se o backend tiver intervalos maiores.
- **Cores e layout**: `PISTON_COLORS` define RGBA para cada eixo – útil para refletir padrões usados em outras documentações.
- **Automação**: as funções globais (ex.: `window.sendSetpointGlobal`) permitem chamar comandos direto pelo console do navegador para scripts de teste.

Troubleshooting

- **Gráfico não aparece**:
    - Confirme que Chart.js foi carregado antes de `chart-utils.js`.
    - Cheque a aba Console para erros de canvas inexistente (`id="telemetry-chart"`).
- **Sem dados no gráfico mesmo conectado**:
    - Veja se o backend realmente está transmitindo telemetria (procure logs `TX`/`RX` no console do backend).
    - Verifique se `window.serialConnected` está `true`. Caso contrário, o WebSocket não tenta reconectar.
- `fetch /pid/...` **retorna erro 422**:
    - Campos vazios ou inválidos (ex.: `NaN`). Insira números com ponto e em mm/graus coerentes.
- **Exportação CSV baixa arquivo vazio**:
    - Grave alguns segundos antes. Sem dados no IndexedDB o arquivo virá somente com cabeçalho.

- **Console mostra “WebSocket sem mensagens”**:
  - Pode ser apenas ausência temporária de telemetria; porém, se persistir, clique em “Desconectar”/“Conectar” para reabrir o socket e verifique cabos.

Autor

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

Github
Linkedin
Portfólio

Última atualização: Novembro 2025