

Sistema de Cinemática & Telemetria - Plataforma de Stewart

Visão Geral

Interface completa para montar poses, visualizar o resultado em 3D e acompanhar a plataforma física em tempo real. A página `kinematics.html`, apoiada pelo script `kinematics.js`, conversa com o backend FastAPI (`app.py`) para calcular cinemática inversa, validar limites e enviar comandos via serial.

Características

- Controle manual da pose com inputs numéricos e sliders sincronizados para X/Y/Z (± 60 mm, Z 400–600 mm) e Roll/Pitch/Yaw ($\pm 25^\circ$).
- Pré-visualização 3D instantânea no `canvas-preview`, renderizada a partir da resposta do endpoint `/calculate`.
- Painel live (`canvas-live`) que replica a pose real recebida por WebSocket e mostra o comprimento de cada atuador.
- Monitoramento do IMU/MPU com leitura de Roll, Pitch, Yaw e quaternion formatado quando enviado pelo BNO085.
- Botão “Aplicar na Bancada” condicionado à validade do cálculo; dispara `/apply_pose` e envia `spxmm6x=...` para o ESP32.
- Ações auxiliares: reset rápido, recalibração do MPU, indicadores de erro/toast e watchdog para telemetria.

Arquitetura

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

- **POST `/calculate`**: recebe `PoseInput`, executa `platform.inverse_kinematics`, calcula percentuais de curso e devolve `PlatformResponse` com `actuators`, `base_points`, `platform_points` e flag `valid`.
- **POST `/apply_pose`**: reaproveita a pose calculada, verifica novamente a validade, converte para curso com `lengths_to_stroke_mm` e envia `spxmm6x` via `SerialManager`. Retorna `{ applied, valid, setpoints_mm }`.
- **POST `/serial/send`**: usado para enviar comandos diretos como `{"command": "recalibra"}` quando o usuário solicita recalibração do MPU.
- **GET `/serial/status`**: informa se já existe porta conectada, permitindo que o frontend inicialize `serialConnected` ao carregar.
- **WebSocket `/ws/telemetry`**: transmite eventos `motion_tick` (com `pose_cmd`) e pacotes de telemetria normalizados usados no canvas live.

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

- Inicializa componentes compartilhados via `initCommonSerialControls` e carrega o menu (`nav-menu.js`).

- Expõe funções globais: `setupInputSync`, `getPoseFromUI`, `calculatePosition`, `applyToBench`, `resetPosition`, `recalibrateMPU`, `updatePreviewMeasures`, `updateLiveMeasures`, `updateMPUData`, `initLocalTelemetryWS`.
- Depende de utilitários:
 - `draw3DPlatform` (de `three-utils.js`) para desenhar base/plataforma/atuadores.
 - `normalizeTelemetry` e `applyLiveTelemetry` (de `telemetry-utils.js`) para tratar o feed live.
 - `showToast`, `setSerialStatus`, `serialConnected`, `serialPorts` (de `common.js`) para UX, conexão e estado global.

Painéis de `kinematics.html`

1. **Status da Porta Serial** (`connection-status`): LED e texto atualizados conforme `serialConnected`/porta ativa.
2. **Painel do Acelerômetro** (`mpu-data-section`): botão "Recalibrar", valores de Roll/Pitch/Yaw com duas casas e quaternion formatado ($q = a + bi + cj + dk$).
3. **Controles de Pose**: inputs `x-pos`, `y-pos`, `z-pos`, `roll`, `pitch`, `yaw` sincronizados aos sliders; botões "Calcular Posição", "Resetar" e "Aplicar na Bancada".
4. **Medidas dos Atuadores**: blocos `preview-piston-*` (resultado do cálculo) e `live-piston-*` (telemetria), coloridos em verde quando válidos e vermelho quando fora do limite.
5. **Visualização 3D**: `canvas-preview` (resposta do cálculo) e `canvas-live` (telemetria) renderizados pelo Three.js.

Fluxo de Funcionamento

1. Usuário ajusta sliders/inputs → `getPoseFromUI` monta o payload.
2. `calculatePosition` envia `POST /calculate`, exibe indicador de loading e trata mensagem de erro/sucesso.
3. Com resposta válida, salva `currentPlatformData`, atualiza medidas (`updatePreviewMeasures`) e renderiza `canvas-preview` com `draw3DPlatform`; o botão "Aplicar na Bancada" é exibido.
4. `applyToBench` reaproveita `currentPlatformData.pose` e chama `/apply_pose`. Enquanto aguarda, o texto do botão muda para "Aplicando...".
5. `initLocalTelemetryWS` estabelece o WebSocket, mantido ativo por heartbeat; mensagens `motion_tick.pose_cmd` atualizam apenas o preview, enquanto o restante passa por `normalizeTelemetry/applyLiveTelemetry` para alimentar o canvas live e `updateLiveMeasures`.
6. `updateMPUData` mostra/oculta o painel do sensor conforme a telemetria inclua campos `mpu` e `quaternions` válidos.

Telemetria e Serial

- **Inicialização**: `checkExistingConnection` (GET `/serial/status`) sincroniza o estado da porta e dispara `initLocalTelemetryWS` se já houver conexão.
- **WebSocket**: utiliza `WS_URL` definido em `common.js`, registra heartbeat de 3 s e força reconexão quando fica mais de 5 s sem mensagens enquanto `serialConnected` é verdadeiro.
- **MPU/Recalibração**: o botão "Recalibrar" bloqueia interação durante o request `POST /serial/send` e usa `showToast/apply-error` para feedback.

- **Serial Manager:** funções comuns (`listar portas`, `conectar`, `desconectar`) vêm de `common.js`, reaproveitando os mesmos botões da página controller.

Passo a Passo de Uso

1. Iniciar backend

```
cd interface/backend
python app.py
```

Servidor disponível em `http://localhost:8001`.

2. **Abrir o frontend:** carregar `interface/frontend/kinematics.html` em um navegador (Chrome recomendado).
3. **Conectar ao ESP32:**
 - Clique em "Atualizar" para listar portas.
 - Escolha a porta correta e clique em "Conectar".
 - O indicador muda para verde e mostra o nome da porta.
4. **Montar e validar a pose:**
 - Ajuste sliders ou digite valores.
 - Clique em "Calcular Posição" para obter comprimentos e preview 3D.
5. **Aplicar na bancada (opcional):**
 - Confirme se todas as barras estão verdes (válidas) e se a pose faz sentido.
 - Clique em "Aplicar na Bancada". A resposta mostra no toast/log.
6. **Acompanhar telemetria:**
 - Observe `canvas-live`, medidas reais e painel do MPU para validar o movimento físico.

Configurações Avançadas

- **Limites dos sliders** (`interface/frontend/kinematics.html`): ajuste os atributos `min/max` dos inputs para refletir novos envelopes mecânicos.
- **Valores padrão** (`resetPosition` em `kinematics.js`): troque os valores atribuídos aos campos quando o usuário clica em "Resetar".
- **Parâmetros de reconexão** (`kinematics.js`): modifique `heartbeatTimer`, `wsTimer` e `scheduleReconnect` para tornar o watchdog mais agressivo ou permissivo.
- **Formatação do MPU:** personalize o texto montado em `updateMPUData` caso queira suprimir o quaternion ou mudar o formato numérico.
- **Toasts/textos:** mensagens globais ficam concentradas em `showToast` e elementos `error-message/apply-error`, facilitando tradução ou ajustes de UX.

Troubleshooting

- **"Pose inválida" ao calcular:** reduza ângulos/translações, verifique limites (curso 500–680 mm) e consulte logs do backend para entender o motivo do `valid=False`.
- **Botão "Aplicar" não aparece:** execute "Calcular Posição" novamente; o botão só fica visível quando `currentPlatformData` existe e `valid === true`.

- **Telemetria parada:** cheque se o backend está enviando mensagens em `/ws/telemetry`; observe o console do navegador para ver se o heartbeat está tentando reconectar.
- **Painel do MPU oculto:** somente é exibido quando `mpu.roll/pitch/yaw` chegam como números. Garanta que o firmware esteja publicando esses campos.
- **Recalibração falhou:** confirme a conexão serial e se o comando `recalibra` é reconhecido pelo firmware (veja o terminal do backend para erros).

Autor

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

[Github](#)

[Linkedin](#)

[Portfólio](#)

Última atualização: Novembro 2025