Firmware PID com Filtro Anti-Spike (ESP32-S3)

Visão Geral

`esp32s3_codes/pid-control-filter-spike-bno/pid-control-filter-spike-bno.ino` é o firmware responsável por fechar o loop dos seis atuadores lineares da plataforma de Stewart. Ele recebe a referência de posição via serial (em milímetros), lê o feedback analógico dos encoders lineares, aplica filtro anti-spike + mediana e gera comandos PWM com feedforward, anti-windup e compensações individuais. O mesmo firmware também escuta dados de orientação via ESP-NOW para exibir roll/pitch/yaw (MPU6050 ou BNO085) e usa esse canal para recalibração remota do sensor.

Principais Recursos

- **Recepção ESP-NOW híbrida**: detecta automaticamente pacotes `OrientationData` (MPU6050 – roll/pitch/yaw) ou `TelemetryData` (BNO085 – inclui quaternions), mantendo os valores em variáveis voláteis.
- **Filtro anti-spike dinâmico** combinado com mediana-3 e média móvel (parametrizável) para suavizar os sinais de feedback.
- **PID por pistão** (`Kp/Ki/Kd` independentes) com feedforward assimétrico (`U0_adv/U0_ret`), offset de curso (mm) e deadband ajustável.
- **Controles manuais** via serial: seleção de pistão, avanço/recuo com PWM fixo ou "modo free" (IN1/IN2 em HIGH).
- **Calibração eletrônica**: cada canal possui par `V0/V100` e flag `hasCal`, permitindo mapear tensão (0–3.3V) para mm.
- **Telemetria CSV** contínua (33 ms) incluindo setpoint, posições reais, PWM aplicado e orientação (com ou sem quaternions).
- **Proteções internas**: limitação de integrador, anti-windup com tracking (`Tt_tracking`), PWM mínimo configurável e watchdog básico de serial.

Arquitetura Geral

Camada	Descrição
ESP-NOW RX (`setupEspNowReceiver`)	Configura o ESP32-S3 como STA, inicializa ESP-NOW e registra `onReceive`. Cada pacote recebido atualiza `oriRoll/Pitch/Yaw` e, no caso do BNO085, `oriQw/Qx/Qy/Qz`.
Aquisição analógica	`readFeedbackVoltages` realiza `analogRead` múltiplas vezes, salva no buffer para mediana e atualiza `fbV_raw`.
Filtragem	`filterFeedbackVoltage` aplica mediana-3, spike detection (rejeita valores que variam além de `SPIKE_THRESH` proporcional ao desvio) e média móvel (`MA_N`). Resultado em `fbV_filt`.
Conversão para mm	`voltsToMM` usa `V0/V100 + offset_mm` e saturação (`Lmm`) para obter `y_mm`.

Camada	Descrição
Controle PID	Dentro da rotina principal (`loop`), para cada pistão calcula erro `e_mm`, derivada, integra com leak/anti-windup e gera PWM limitado. Feedforward assimétrico compensa zona morta.
Comandos manuais	Flags `manual_retract/manual_advance` permitem aplicar PWM fixo (`RETRACT_PWM/ADV_PWM`) apenas no pistão selecionado (`selIdx`).
Serial Parser	Aceita comandos textuais (`sel=`, `spmm=`, `spmm6x=`, `kpmm=`, etc.) separados por `\n`, atualizando o estado imediatamente.
Telemetria	A cada 33 ms imprime CSV com cabeçalho dinâmico (adiciona colunas de quaternions quando presentes).

Pinagem e PWM

- **Feedback analógico (`FB_PINS`)**: `{1, 2, 3, 4, 5, 6}` (use ADC1 para ruído menor).
- **PWM (`PWM_PINS`)**: `{8, 18, 9, 10, 11, 12}` usando LEDC (freq. 20 kHz, resolução 8 bits).
- **IN1/IN2 por pistão (`pistons[]`)**: `{{16, 17}, {13, 14}, {35, 36}, {21, 38}, {39, 37}, {41, 42}}` controla o sentido (avançar/recuar/free). Ex.: pistão 1 usa GPIO16/17.

Principais Parametrizações

Variável	Descrição
`Lmm[6]`	Curso útil de cada atuador (mm).
`SP_mm[6]`	Setpoint atual (mm).
`Kp_mm/Ki_mm/Kd_mm`	Ganhos PID individuais.
`U0_adv/U0_ret`	Feedforward para compensar zona morta em avanço/recuo.
`offset_mm`	Compensação fixa no feedback (ex.: +2 mm).
`deadband_mm`	Janela onde o PID desliga (mantém free e esvazia integrador).
`MIN_PWM`	PWM mínimo aplicado quando o controle exige movimento (evita ficar abaixo do atrito estático).
`T_leak`, `Tt_tracking`, `I_LIM`	Constantes de anti-windup e escoamento de integrador.

Comandos Serial

Todos os comandos são enviados via USB/Serial na forma `texto\n`.

Comando	Função
`sel=X`	Seleciona pistão `1..6` para modo manual.
`spmm=VAL`	Define setpoint global (mm).

Comando	Função
`spmm6x=v1,...,v6`	Define setpoints individuais.
`spmmN=VAL`	Define setpoint do pistão N (1-6).
`kpmm=idx,val` / `kimm=` / `kdmm=`	Ajusta ganho PID do pistão `idx`.
`kpall=VAL` / `kiall=` / `kdall=`	Aplica o mesmo ganho em todos.
`dbmm=VAL`	Atualiza `deadband_mm`.
`lmm=idx,val`	Atualiza curso útil do pistão (ajusta saturação).
`fc=idx,v0,v100`	Atualiza calibração de tensão (`V0/V100`).
`minpwm=VAL`	Define PWM mínimo.
`vmaxmmps=VAL`	Limita derivada permitida (anti-spike).
`cal=idx`	Marca que o pistão `idx` possui calibração válida (`hasCal`).
`u0a=idx,val` / `u0r=idx,val`	Ajusta feedforward individual (avanço/recuo).
`u0aall=VAL` / `u0rall=VAL`	Ajuste em massa dos feedforward.
`offset=idx,val` / `offsetall=VAL`	Ajusta compensação em mm.
`man=adv` / `man=ret` / `man=free`	Ativa avanço, recuo ou libera modo manual.
`recalibra`	(Enviado via serial) Notifica o transmissor ESP-NOW para refazer o zero.

Fluxo Operacional

1. **Boot**: configura serial, PWM, leitura analógica, filtros e ESP-NOW (`setupEspNowReceiver`).
2. **Loop principal**:
   - Lê serial (`parseSerialCommand`), atualizando qualquer parâmetro.
   - Atualiza `dt` (baseado em `micros()`), lê tensões e filtra.
   - Calcula feedback em mm e aplica PID ou modo manual conforme flags.
   - Escreve PWM (`ledcWrite`) e define direção (`setDirAdvance/Return/Free`).
   - A cada 33 ms, imprime linha CSV com telemetria.
3. **ESP-NOW**: sempre que chega pacote, atualiza `oriRoll/Pitch/Yaw` (usados na telemetria) e guarda `lastSenderMac` para eventuais respostas.

Exemplo de Telemetria (MPU6050)

```
ms;SP_mm;Y1;Y2;Y3;Y4;Y5;Y6;PWM1;PWM2;PWM3;PWM4;PWM5;PWM6;Roll;Pitch;Yaw
12345;10.000;9.85;10.12;9.97;9.90;9.88;10.01;80;75;78;82;79;81;1.24;-0.50;0.02
```

No caso do BNO085 o cabeçalho inclui `Qw,Qx,Qy,Qz` e a linha adiciona esses valores ao final.

Dicas de Uso

1. **Calibração dos sensores lineares**: use `fc=idx,v0,v100` para cada pistão, depois `cal=idx` para marcar como válido. Sem essa etapa, `voltsToMM` pode saturar.
2. **Tunagem PID**: ajuste `Kp/Ki/Kd` via comandos individuais ou globais. O feedforward (`u0a/u0r`) ajuda a reduzir o esforço do PID quando os atuadores entram no regime não linear.
3. **Anti-spike**: se o ambiente estiver ruidoso, ajuste `SPIKE_THRESH_BASE/SPIKE_THRESH_PER_MM` (no início do arquivo) ou reduza `vmaxmmps` via serial para suavizar o filtro dinâmico.
4. **Telemetria**: use `pio device monitor` ou qualquer terminal serial configurado em 115200 baud para registrar os dados CSV. Importando em um spreadsheet fica fácil comparar setpoint vs. real.
5. **Integração com o backend**: o FastAPI envia `spmm6x=` com os cursos calculados. Certifique-se de usar a mesma unidade (mm).

Estrutura do Projeto

```
esp32s3_codes/
└── pid-control-filter-spike-bno/
    ├── pid-control-filter-spike-bno.ino   ← firmware principal
    └── (demais arquivos gerados pela IDE, se houver)
```

Autor

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

Github
Linkedin
Portfólio

Última atualização: Novembro 2025