

mit_joint.cpp

This example demonstrates joint control using MIT_INTEGRATED mode.

This example demonstrates joint control using MIT_INTEGRATED mode. ⚠️ WARNING:

- MIT_INTEGRATED is an advanced joint-level torque control mode.
- It requires precise tuning and understanding of low-level control.
- NOT recommended for non-professional or casual users.

Key Concepts:

- Before switching to MIT_INTEGRATED, the robot must first move to a valid joint position using PLANNING_POS.
- In this example, joint 1 is oscillated back and forth within a predefined range.
- Control loop runs at approximately 250 Hz (calculated via loop interval).
- Position commands are updated on each iteration, while velocity, effort, and gains remain constant.

Example usage:

```
#include "airbot_sdk.hpp"
#include <array>
#include <cmath>
#include <thread>
#include <chrono>
using namespace airbot;

int main() {
    auto sdk = std::make_shared<AirbotSdk>("192.168.0.xxx", 50051);

    // Step 1: Move to initial joint position using PLANNING_POS mode
    sdk->SwitchMode({"arm"}, {RobotMode::PLANNING_POS});
    std::array<double, 6> init_pos = {0.0, -1.19, 1.17, -1.55, 1.51, 0.0};
    sdk->MoveToJointPos(init_pos);

    // Step 2: Switch to MIT_INTEGRATED mode
    sdk->SwitchMode({"arm"}, {RobotMode::MIT_INTEGRATED});

    // Step 3: Prepare control inputs
    std::array<double, 6> &pos = init_pos;
    const std::array<double, 6> vel = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
```

```

const std::array<double, 6> eff = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
const std::array<double, 6> kp  = {90.0, 150.0, 150.0, 25.0, 25.0, 25.0};
const std::array<double, 6> kd  = {2.5, 1.75, 1.5, 0.5, 1.5, 0.5};

// Step 4: Joint 1 oscillation logic
double joint1_upper = 1.57;
double joint1_lower = -1.57;
double delta_pos = 0.0015; // step size per iteration
int direction = 1;

constexpr int loop_count = 8000;
constexpr double interval_ms = 0.8 / 250 * 1000; // ~3.2ms per loop

for (int i = 0; i < loop_count; ++i) {
    sdk->MitJointIntegratedControl(pos, vel, eff, kp, kd);
    pos[0] += direction * delta_pos;
    if (pos[0] >= joint1_upper || pos[0] <= joint1_lower) {
        direction *= -1; // reverse direction when hitting limit
    }
}

// Optional: wait to ensure final commands are applied
std::this_thread::sleep_for(std::chrono::seconds(1) +
                             std::chrono::microseconds(static_cast<int>(loop_

// Step 5: Return to original pose using PLANNING_POS
sdk->SwitchMode({"arm"}, {RobotMode::PLANNING_POS});
sdk->MoveToJointPos(init_pos);

return 0;
}

```