-- coding: utf-8 --

""" Dogzilla Flask server - Giữ hỗ trợ 'speed' cho turnleft/turnright để tương thích client hover. - BỔ SUNG lệnh đặt độ cao tuyệt đối qua chuột: `setz` (và tuỳ chọn `adjustz`).

Mapping API: - POST /control { "command": "forward|back|left|right|turnleft|turnright|stop|setz|adjustz|status", "step": <int>, "speed": <int>, "value": <int>, # cho setz (độ cao tuyệt đối) "delta": <int> # cho adjustz (độ cao tương đối) }

- forward/back/left/right/turnleft/turnright:
  - Nếu có "speed": dùng speed (đã clamp) làm biên độ điều khiển
  - Nếu không có "speed" nhưng có "step": dùng step (tương đương)
  - Nếu không truyền gì: dùng STEP_DEFAULT
- stop: gọi stop()
- setz: đặt Z tuyệt đối (độ cao), clamp về [Z_MIN, Z_MAX]
- adjustz: chỉnh Z tương đối (delta), clamp kết quả về [Z_MIN, Z_MAX]

- status: trả các thông tin server/robot

- GET /camera -> MJPEG stream (client mở bằng cv2.VideoCapture(URL)) """ import os import time import threading from typing import Optional

from flask import Flask, request, jsonify, Response

====== CONFIG (ENV hoặc mặc định) ======

HTTP_PORT = int(os.environ.get("HTTP_PORT", "9000")) CAMERA_INDEX = int(os.environ.get("CAMERA_INDEX", "0")) DOG_PORT = os.environ.get("DOGZILLA_PORT", "/dev/ttyAMA0") DOG_BAUD = int(os.environ.get("DOGZILLA_BAUD", "115200")) STEP_DEFAULT = int(os.environ.get("STEP_DEFAULT", "8")) # dùng khi không truyền step/speed FRAME_W = int(os.environ.get("CAM_WIDTH", "640")) FRAME_H = int(os.environ.get("CAM_HEIGHT", "480")) FRAME_FPS = int(os.environ.get("CAM_FPS", "30"))

Ngưỡng tốc độ xoay (phù hợp client TURN_SPEED [-70,70])

TURN_MIN = int(os.environ.get("TURN_MIN", "-70")) TURN_MAX = int(os.environ.get("TURN_MAX", "70"))

Độ cao Z (cho setz/adjustz)

Z_MIN = int(os.environ.get("Z_MIN", "75")) Z_MAX = int(os.environ.get("Z_MAX", "110")) Z_DEFAULT = int(os.environ.get("Z_DEFAULT", "105"))

====== IMPORT LIB & INIT ROBOT ======

try: from DOGZILLALib import DOGZILLA except Exception: try: from dogzilla import DOGZILLA except Exception as e: DOGZILLA = None print("Cannot import DOGZILLA class:", e)

app = Flask(**name**)

Robot instance

dog = None if DOGZILLA is not None: try: dog = DOGZILLA(port=DOG_PORT, baud=DOG_BAUD, verbose=False) print(f"[DOGZILLA] Connected on {DOG_PORT} @ {DOG_BAUD}") except Exception as e: print("[DOGZILLA] Init error:", e) dog = None else: print("[DOGZILLA] Library not found. Running without robot.")

====== CAMERA (OpenCV V4L2 + MJPG) ======

cv2 = None camera = None

def _init_camera(): global cv2, camera try: import cv2 as _cv2 cv2 = _cv2 cam = cv2.VideoCapture(CAMERA_INDEX, cv2.CAP_V4L2) # MJPG giúp nhẹ CPU và phù hợp MJPEG stream cam.set(cv2.CAP_PROP_FOURCC, cv2.VideoWriter_fourcc(*"MJPG")) cam.set(cv2.CAP_PROP_FRAME_WIDTH, FRAME_W) cam.set(cv2.CAP_PROP_FRAME_HEIGHT, FRAME_H) cam.set(cv2.CAP_PROP_FPS, FRAME_FPS) if not cam.isOpened(): print(f"[Camera] Cannot open index {CAMERA_INDEX}") camera = None return camera = cam print(f"[Camera] Opened index {CAMERA_INDEX} ({FRAME_W}x{FRAME_H} @ {FRAME_FPS}fps)") except Exception as e: print("[Camera] Init error:", e) camera = None

_init_camera()

def _gen_mjpeg(): if camera is None or cv2 is None: # Stream rỗng để client không treo while True: time.sleep(0.5) yield (b'--frame Content-Type: image/jpeg

') else: while True: ok, frame = camera.read() if not ok: time.sleep(0.01) continue ok, buf = cv2.imencode(".jpg", frame) if not ok: continue jpg = buf.tobytes() yield (b'--frame Content-Type: image/jpeg

' + jpg + b' ')

@app.route("/camera") def camera_feed(): return Response(_gen_mjpeg(), mimetype="multipart/x-mixed-replace; boundary=frame")

====== HELPERS ======

def _clamp(v: int, lo: int, hi: int) -> int: return lo if v < lo else hi if v > hi else v

server-side current Z (nếu lib không cho đọc trực tiếp)

_current_z = Z_DEFAULT _z_lock = threading.Lock()

def _setz(z: int) -> str: """Đặt Z tuyệt đối, clamp và gửi xuống robot.""" global _current_z z = _clamp(int(z), Z_MIN, Z_MAX) if dog is None: with _z_lock: _current_z = z return f"ok: setz({z}) (robot not connected)" try: # Ưu tiên API translation nếu có if hasattr(dog, 'translation'): dog.translation('z', z) elif hasattr(dog, 'setz') and callable(getattr(dog, 'setz')): dog.setz(z) else: # Fallback: nếu không có API, coi như không hỗ trợ return "error: setz unsupported by DOGZILLA lib" with _z_lock: _current_z = z return f"ok: setz({z})" except Exception as e: return f"error: {e}"

def _adjustz(delta: int) -> str: global _current_z with _z_lock: target = _current_z + int(delta) return _setz(target)

def _resolve_value(step: Optional[int], speed: Optional[int], *, is_turn: bool) -> int: """Ưu tiên 'speed' (khi nút hover gửi speed), fallback sang 'step' rồi mặc định.""" if speed is not None: val = int(speed) if is_turn: val = _clamp(val, TURN_MIN, TURN_MAX) return val if step is not None: return int(step) return STEP_DEFAULT

def _apply_motion(cmd: str, *, step: Optional[int] = None, speed: Optional[int] = None) -> str: """Ánh xạ các lệnh di chuyển cơ bản.""" if dog is None: return "robot not connected"

```
is_turn = cmd in ("turnleft", "turnright")
val = _resolve_value(step, speed, is_turn=is_turn)

try:
    if cmd == "forward":
        dog.forward(val)
    elif cmd == "back":
        dog.back(val)
    elif cmd == "left":
        dog.left(val)
    elif cmd == "right":
        dog.right(val)
    elif cmd == "turnleft":
        dog.turnleft(val)
    elif cmd == "turnright":
        dog.turnright(val)
    elif cmd == "stop":
        dog.stop()
    else:
        return f"unknown command: {cmd}"
except Exception as e:
    return f"error: {e}"

if cmd == "stop":
    return "ok: stop"
```

```
if is_turn:
    return f"ok: {cmd}(speed={val})"
return f"ok: {cmd}({val})"
```

====== ENDPOINTS ======

@app.route("/control", methods=["POST"]) def control(): data = request.get_json(silent=True) or {} cmd = str(data.get("command", "")).lower().strip()

```
# parse params
raw_step   = data.get("step")
raw_speed  = data.get("speed")
raw_value  = data.get("value")  # for setz
raw_delta  = data.get("delta")  # for adjustz

# validate numeric when present
step = None
if raw_step is not None:
    try:
        step = int(raw_step)
    except Exception:
        return "invalid step", 400, {"Content-Type": "text/plain;
charset=utf-8"}

speed = None
if raw_speed is not None:
    try:
        speed = int(raw_speed)
    except Exception:
        return "invalid speed", 400, {"Content-Type": "text/plain;
charset=utf-8"}

# dispatch
try:
    if cmd in ("forward", "back", "left", "right", "turnleft", "turnright",
"stop"):
        res = _apply_motion(cmd, step=step, speed=speed)
    elif cmd == "setz":
        if raw_value is None:
            return "missing value", 400, {"Content-Type": "text/plain;
charset=utf-8"}
        res = _setz(raw_value)
    elif cmd == "adjustz":
        if raw_delta is None:
            return "missing delta", 400, {"Content-Type": "text/plain;
charset=utf-8"}
        res = _adjustz(raw_delta)
    elif cmd == "status":
        # allow POST to fetch json status too
        return status()
    else:
        return f"unknown command: {cmd}", 400, {"Content-Type": "text/plain;
charset=utf-8"}

    if res.startswith("ok"):
        return res, 200, {"Content-Type": "text/plain; charset=utf-8"}
    else:
        return res, 400, {"Content-Type": "text/plain; charset=utf-8"}

except Exception as e:
    # Nếu thư viện vẫn dùng list cho serial.write, pyserial có thể báo lỗi.
    # Khắc phục trong lib: đổi self.ser.write(tx) ->
self.ser.write(bytearray(tx))
    print("[/control] error:", e)
    return f"error: {e}", 500, {"Content-Type": "text/plain; charset=utf-8"}
```

@app.route("/status", methods=["GET", "POST"]) def status(): with _z_lock: cz = _current_z s = { "robot_connected": dog is not None, "turn_speed_range": [TURN_MIN, TURN_MAX], "step_default": STEP_DEFAULT, "z_range": [Z_MIN, Z_MAX], "z_current": cz, } if dog is not None: try: s["battery"] = dog.read_battery() except Exception: s["battery"] = None try: s["fw"] = dog.read_version() except Exception: s["fw"] = None return jsonify(s)

@app.route("/") def root(): return jsonify({ "status": "ok", "endpoints": { "control": "POST /control {command: forward|back|left|right|turnleft|turnright|stop|setz|adjustz|status, step?: int, speed?: int, value?: int, delta?: int}", "camera": "GET /camera (MJPEG)", "status": "GET/POST /status" } })

====== CLEANUP ======

def _cleanup(): try: if dog is not None: try: dog.stop() except Exception: pass except Exception: pass try: if camera is not None: camera.release() except Exception: pass

import atexit atexit.register(_cleanup)

if **name** == "**main**": print(f"[Server] HTTP_PORT={HTTP_PORT} CAMERA_INDEX={CAMERA_INDEX} DOG={DOG_PORT}@{DOG_BAUD}") print(f"[Server] TURN_SPEED_RANGE=[{TURN_MIN},{TURN_MAX}] STEP_DEFAULT={STEP_DEFAULT}") print(f"[Server] Z_RANGE=[{Z_MIN},{Z_MAX}] Z_DEFAULT={Z_DEFAULT}") try: app.run(host="0.0.0.0", port=HTTP_PORT, threaded=True) finally: _cleanup()