Lab 0x04 – Line Following Integration Overview

Summary of Changes

The current Lab 0x04 codebase merges the robust control framework from **Lab 0x03** with the new IR-sensor-based line-following behavior.
Major updates include:

1. **Restored and Improved Core Drivers**
   - Brought back the refined `battery_droop.py`, `motor.py`, `encoder.py`, and `closed_loop.py` from Lab 0x03.
   - Battery droop compensation now scales all motor efforts automatically to maintain consistent performance as batteries discharge.
   - Retained non-blocking cooperative scheduler (`cotask.py`, `task_share.py`) for reliable multitasking.
2. **New Line-Following Functionality**
   - Added an `IR_sensor.py` module that reads four analog IR sensors, calibrates for white/black, and computes a centroid error.
   - Added a new **`steering_task.py`** (formerly "steering" placeholder) that interprets the IR centroid, computes a yaw correction, and publishes left/right motor efforts.
   - Integrated line-follow control into `motor_task.py`: when the `lf_enable` flag is high, it uses the steering task's efforts instead of manual or closed-loop velocity control.
3. **Expanded User Interface Commands**
   - The Bluetooth UI (`ui_task.py`) now supports:
     • `w` → calibrate IR sensors on white background
     • `b` → calibrate IR sensors on black line
     • `l` → toggle line-follow mode on/off
     • existing commands (`g`, `k`, `e`, `t`, `v`, etc.) remain unchanged.
4. **Simplified Scheduler Setup**
   - `main.py` instantiates all tasks—UI, Steering, Motor, and optional Stream—using the same cooperative structure from Lab 0x03.
   - Removed blocking test scripts and redundant data-collection logic; everything now runs continuously and non-blocking.
   - Only essential runtime files are needed on the Romi.

In short: the robot can now follow a black line autonomously while still supporting manual effort and velocity modes for comparison.

Testing Plan for Line Following

1. **Setup & Calibration**
    1. Power on the Romi and connect via Bluetooth (PuTTY or VS Code serial terminal, 115200 baud).
    2. Place the robot on the **white mat** and send **`w`** → records white reflectance values.
    3. Move the robot so the IR array is centered on the **black line** and send **`b`** → records black reflectance values.
    4. Verify calibration messages print (“Calibrating white…”, “Calibrating black…”).
2. **Enable Line-Following Mode**
    - Send **`l`** to toggle the line-following flag ON.
    - The UI will report mode status in the terminal.
3. **Start the Motors**
    - Send **`g`** (“GO”) to enable the motor task.
    - The steering task now drives differential efforts based on the IR centroid:
        - Left motor speeds up if line drifts left, right motor speeds up if line drifts right.
    - Observe the Romi following the circular line on the mat.
4. **Stop or Abort**
    - Send **`k`** to stop the motors immediately.
    - Send **`l`** again to toggle line-follow OFF and return to manual modes.
5. **Tuning (if needed)**
    - Adjust `Kp_line` (in `steering_task.py`) for steering sensitivity.
      Typical starting range: 20–40.
    - Adjust `base_effort` for desired forward speed.
    - Re-calibrate (`w`, `b`) if lighting or surface changes.
6. **Optional Data Streaming**
    - Send **`s`** to log position/velocity data for analysis; behavior does not block line-following operation.

Expected Behavior

After calibration and enabling line-follow mode, the Romi should move forward and continuously steer to keep its IR array centered over the black line. The motors should automatically slow and correct when the robot drifts off-center, producing smooth, continuous circular motion.

Current State:
All modules are integrated, non-blocking, and verified for scheduler compatibility. The system is ready for physical testing on the Romi chassis.