

AI Interaction Log: Phase 4 (IR Feedback & PID Control)

Date Range: November 17, 2025 **Objective:** Implement closed-loop speed control using an IR sensor and a PID controller on the Raspberry Pi.

1. Initial Implementation (The Logic Gap)

Prompt: "Time to work towards project completion! specifically the feedback loop between the ir sensor and the motor."

- **AI Contribution:** Generated `parmco_server.c` (v1) featuring a Proportional-Integral-Derivative (PID) controller, a state machine for Manual/Auto modes, and a parsing logic for target RPMs.
- **The Issue:** When switching to "Auto Mode," the motor would not turn on.
- **Debugging:** We realized that switching the `mode` variable did not inherently trigger the `motor_start` variable.
- **Resolution:** Added logic to force `motor_running = 1` and `gpio_write(MASTER_ON, 1)` immediately upon receiving the 'Auto' command.

2. The "Zero RPM" Mystery (Hardware Mismatch)

Prompt: "The RPM is being read as 0 constantly... even when trying to set an RPM in automatic mode."

- **AI Contribution:** Suggested creating a minimal diagnostic tool (`debug_rpm.c`) to isolate hardware signals from the complex server code.
- **The Issue:** The C code was configured for `FALLING_EDGE` and `PULL_DOWN` resistors. However, cross-referencing a provided Python script revealed the hardware setup actually required `RISING_EDGE` and `PULL_UP`.
- **Resolution:** Updated `parmco_server.c` to use:

```
C
set_pull_up_down(pi, SENSOR_PIN, PI_PUD_UP);
callback(pi, SENSOR_PIN, RISING_EDGE, rpm_callback);
```

3. The "Noise Storm" (Signal Processing)

Prompt: "The motor starts spinning at a high speed, then stops for a second... getting carried away." **Log Analysis:** Act=3736 (Noise) -> Err=-2536 -> Speed=0%.

- **Analysis:** The sensor was picking up electrical noise from the motor, triggering interrupts thousands of times per second. The PID controller reacted to these fake "overspeeds" by cutting power to zero, causing oscillation.
- **AI Contribution:** Designed a multi-stage software filter stack:
 - **Hardware Glitch Filter:** Configured pigpio to ignore pulses shorter than 100µs.
 - **Physics Cap:** Implemented logic to reject any RPM reading > 12,000 (physically impossible for the motor).
 - **Smoothing:** Replaced raw readings with an Exponential Moving Average ($\text{smooth} = 0.5 * \text{new} + 0.5 * \text{old}$).
- **Resolution:** Stable RPM readings were achieved, preventing the PID controller from panicking.

4. Taming the Controller (PID Tuning)

Prompt: "The motor attempts to correct itself! however, it goes between stopping entirely or changing its speed a ton."

- **Analysis:** The PID gains (K_p, K_i, K_d) were too aggressive. The controller was reacting too violently to small errors ("Bang-Bang" control).
- **AI Contribution:**
 - **Simplified:** Disabled Integral (I) and Derivative (D) terms to reduce complexity.
 - **Gentle P:** Reduced Proportional gain (K_p) to 0.01.
 - **Slew Rate Limiter:** Added a MAX_CHANGE_PER_LOOP constant to ensure motor power could only change by 5% per second, regardless of error size.
- **Resolution:** The motor now ramps up and down smoothly to meet targets.

5. The Protocol Mismatch (Data Visualization)

Prompt: "RPM reporting works... but the app reads RPM: 0."

- **Analysis:** The C code was sending debug-friendly data (DATA:RPM, Target, Mode) but the Android app was hard-coded to look for the specific prefix RPM:..

- **Resolution:** Reverted the C code output format to strictly match the legacy app requirement:

C

```
snprintf(data_str, sizeof(data_str), "RPM:%d\n", rpm_smooth);
```

6. Packet Fragmentation (Command Parsing)

Prompt: "When sending a desired rpm... log shows APP->PI: 'r', then ':', then '1'..."

- **Analysis:** Bluetooth data arrives in streams, not packets. The C code was trying to process single characters, breaking multi-digit commands like r:1000.
- **AI Contribution:** Wrote a C State Machine (`parse_input_byte`) to buffer incoming characters until a full command or newline was detected.
- **Resolution:** The Pi can now reliably parse complex commands like r:1200\n even if they arrive split across multiple Bluetooth packets.