Aubrey David - AI Interaction Log: Checkpoint 3

This document summarizes the iterative development and debugging process between the student (the project manager) and the AI (Gemini, the code author) to achieve the Checkpoint 3 goals.

The primary challenges were:

1. Developing a stable, headless C-based Bluetooth server.
2. Solving the "stale bond" cache issue required to pass the "forget and reconnect" test, especially with iOS.
3. Fixing a non-obvious "boot race" condition caused by headless operation.

Saga 1: Creating the C Server (`parmco_server.c`)

This phase involved converting our keyboard-controlled script into a background server.

- **Problem:** The initial C server would run, but it wouldn't shut down properly. The `systemctl restart` command would hang, forcing a manual `kill` command.
- **Analysis:** We determined the program was "stuck" in a blocking `accept()` call, which prevented it from receiving the `SIGTERM` stop signal from `systemd`.
- **Solution:** The AI rewrote the C code to use a **non-blocking socket** (`fcntl(server_sock, F_SETFL, O_NONBLOCK)`). This allowed the main loop to check for the `keep_running` flag and shut down gracefully.

Final Artifact: `parmco_server.c` (v1)

Saga 2: The "Forget and Reconnect" Failure (The "Stale Bond")

This was the most complex part of the checkpoint. The app would connect once, but after "forgetting" the Pi, it would fail to reconnect.

- **Problem:** The phone (especially iOS) would show a "Pairing Unsuccessful" error.
- **Analysis:** The AI identified this as a "stale bond" issue. The phone was deleting its security key, but the Pi was *remembering* the old key and refusing the new pairing.
- **Iteration 1 (Failure):** We tried to make the Pi "stateless" by using a `systemd` override (`ExecStartPre=rm...`) to clear the Bluetooth cache (`/var/lib/bluetooth/*`) on boot. This failed.

- **Iteration 2 (Failure):** The AI diagnosed that the `ProtectSystem=full` security feature was blocking our script. We tried to bypass this with `ReadWritePaths=`, which *also* failed.
- **Iteration 3 (Success):** The AI pivoted to a simpler, more robust `rc.local` script. This runs at the *end* of the boot sequence, guaranteeing the Bluetooth service is ready. The user provided their phone's MAC address, and the AI wrote the `fix_bluetooth.sh` script to surgically remove *only* that MAC address.

Final Artifact: `fix_bluetooth.sh` and the updated `/etc/rc.local`.

Bash
```bash
# /usr/local/bin/fix_bluetooth.sh
#!/bin/bash
sleep 20 # Wait for bluetoothd to be ready
(
echo "remove XX:XX:XX:XX:XX:XX"
echo "quit"
) | bluetoothctl
```

Saga 3: The "Headless Boot Race"

Even with the `rc.local` fix, the "forget" test would pass *only* if a monitor was plugged in. It failed when "truly headless."

- **Problem:** The test failed when the monitor was unplugged.
- **Analysis:** The AI diagnosed this as a "boot race" condition.
  - **Monitor In:** The Pi's boot is *slower* (initializing video). Our script had time to work.
  - **Monitor Out:** The Pi's boot is *faster*. Our script ran *before* the Bluetooth service was ready, so the `remove` command failed.
- **Solution:** The AI modified the `fix_bluetooth.sh` script, increasing the `sleep` time from 5 to **20 seconds**. This "patient" script now works reliably in both "fast" (headless) and "slow" (monitor) boot modes.

Saga 4: The "Socket Closed" Error (Authorization)

The phone could now pair, but the app would disconnect instantly with a "socket closed" error.

- **Problem:** App connection failed immediately.
- **Analysis:** The AI identified this as a **Service Authorization** failure. The Pi was asking, "Is this app *allowed* to use Channel 22?" but no one was there to answer.
- **Solution:** The AI provided the `bt_agent.py` "doorman" script. This script was added to `rc.local` to run in the background and automatically authorize any pairing *and* service requests without "trusting" them (which avoids the stale bond).

Final Artifact: The final `/etc/rc.local` file.

Bash
```bash
#!/bin/sh -e
/usr/local/bin/fix_bluetooth.sh &
/usr/local/bin/bt_agent.py &
exit 0
```

Saga 5: The "Instant Disconnect" Bug Fix

The app would connect, but then give a `read failed, socket might closed` error.

- **Problem:** The C server was hanging up on the app.
- **Analysis:** The AI identified a bug in its own `parmco_server.c` code. A non-blocking `write()` was failing (with `EAGAIN`, "try again"), but the code was misinterpreting this as a "disconnect" and closing the socket.
- **Solution:** The AI provided a final patch to `parmco_server.c` to properly check for `EAGAIN` and `EWOULDBLOCK` errors, making the connection stable.

Final Artifact: `parmco_server.c` (v2)