ECSE 4235: Embedded Systems II

Aubrey David, Nevan Mukherjee

Final Project CP3

11/13/2025

Roles and Responsibilities

Nevan – “Client-side” implementation (App)

Aubrey – “Server-side” implementation (Raspberry Pi, headless Bluetooth, program running on boot)

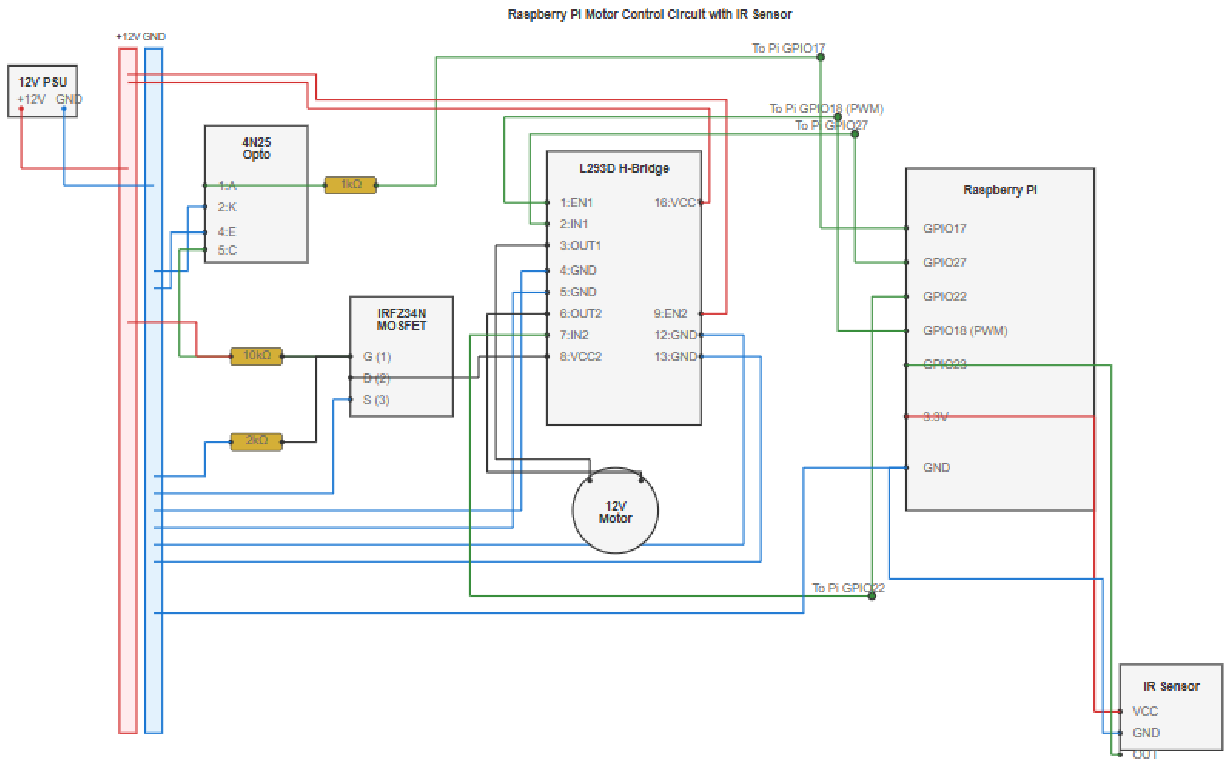
Code

https://github.com/nm67037/Nevan_Aubrey_4235_AI_Project/tree/d3afafd867df4560c90ddbac2c77b8b1e965db86/CP3

Table of Contents

Updated Motor Schematic

The only change is slightly larger font for better readability. The IR sensor has also been physically built into the system for RPM reporting.



AI Progress

The team has continued to use Google's Gemini 2.5 Pro for the deliverables. Claude 4.5 Sonnet was used to improve the schematic.

Aubrey:

It's clear that Gemini is not very familiar with specific system functions in the Raspberry Pi. Two big struggles in getting this checkpoint working were 1) getting the Pi to connect via Bluetooth "headlessly" and 2) changing our C program to run properly on reboot. it was easy enough to remove this confirmation, at the cost of reduced security, which Gemini was sure to warn me about. However, re-establishing a connection would not work. Gemini found the issue quickly: The phone forgot the Pi, but the Pi would not forget the phone. The initial solution Gemini came up with for this problem was to try and clear its cache upon reboot. However, the command it wanted to run did not work, and I would say so, but it would suggest that again a few prompts later. I eventually came up with the idea of hard coding the MAC address of the phone we were testing with and removing it from the Bluetooth cache with some script than ran on reboot. This solution worked, after some more refinement with Gemini.

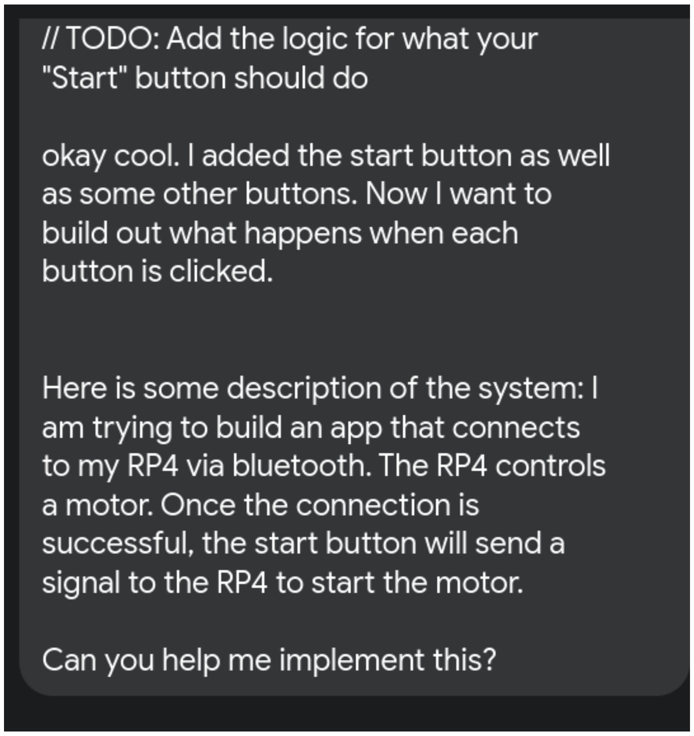
Another struggle was changing the current C program to accept commands through a Bluetooth RFCOMM channel instead of the keyboard. Gemini initially set this new program up to use a channel that was already in use, causing it to fail and timeout. I had to ask for troubleshooting ideas and give their results back to Gemini before it realized this issue. There were other problems with the program itself, such as not checking the connection properly and not starting/stopping the motor with the commands from the app, but Gemini did a good job of noticing these bugs quickly and providing a fix.

There were also a bunch of small errors it would make, such as giving me a command to run without having "sudo" before it like it should. It's favorite turn of phrase to give me is "smoking gun" when referring to a clue revealing the "real" problem I was having for any given prompt. It's also very confident that the new fix it provided will absolutely do the job. For more information on the AI interactions, I had Gemini write an interaction log, "Aubrey_AI_CP3".

Nevan:

At the beginning of our working session, we just had the button to scan for devices on the phone app. So, the first order of business was to get the other necessary buttons on the app UI (stop, start, faster, slower, etc). Gemini made it very simple: add the button to my .xml layout file, and then edit the main activity code to show the button. Gemini even posted the entire xml file (what it should be) and showed a snippet of the main activity that it was suggesting to edit. I implemented these suggested edits.

Once I saw the buttons showing up on the emulator in Android Studio I told Gemini a little bit about our system:



// TODO: Add the logic for what your "Start" button should do

okay cool. I added the start button as well as some other buttons. Now I want to build out what happens when each button is clicked.

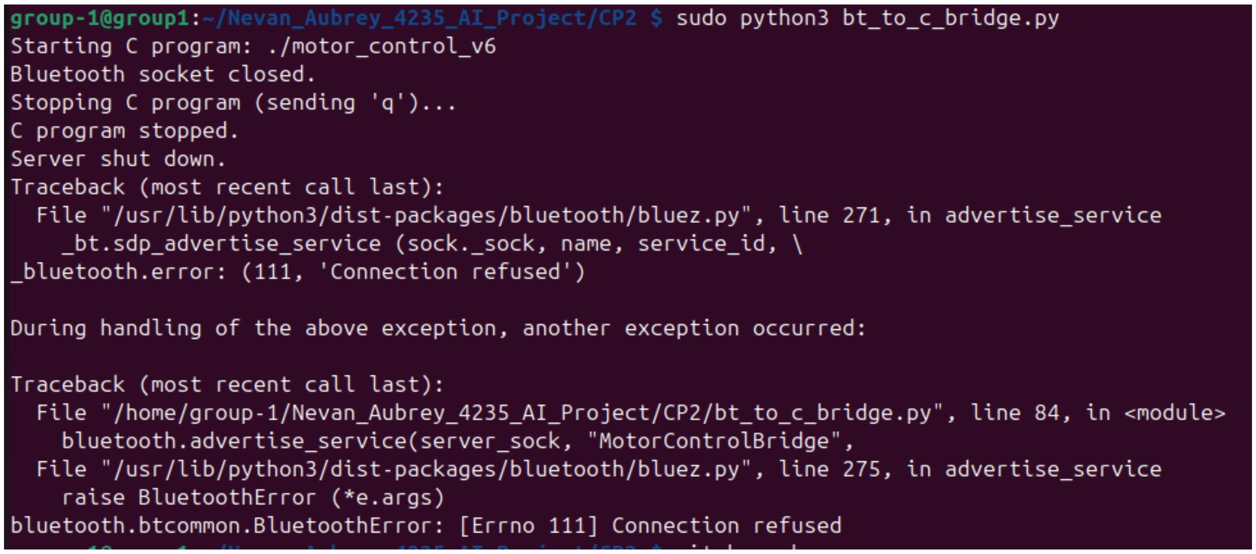
Here is some description of the system: I am trying to build an app that connects to my RP4 via bluetooth. The RP4 controls a motor. Once the connection is successful, the start button will send a signal to the RP4 to start the motor.

Can you help me implement this?

Gemini then proceeded to give me a bunch of details, showing me what sections of the main activity code it would edit. Normally, I would try to start implementing these suggested edits, but Aubrey suggested I send Gemini the c script that sets up the local bluetooth server on the RP4. I sent Aubrey’s C script. More specifically, I sent the program that used user keyboard input to control the motor for CP2.

Gemini read this and suggested a python bridge script that would essentially take a button press from the app, translate that into a “keyboard input” like “s” or “f” and the python would then send that virtual keyboard input to the script Aubrey made that controls the motor based on real keyboard input.

I tried doing what it told me to, but I got some errors:

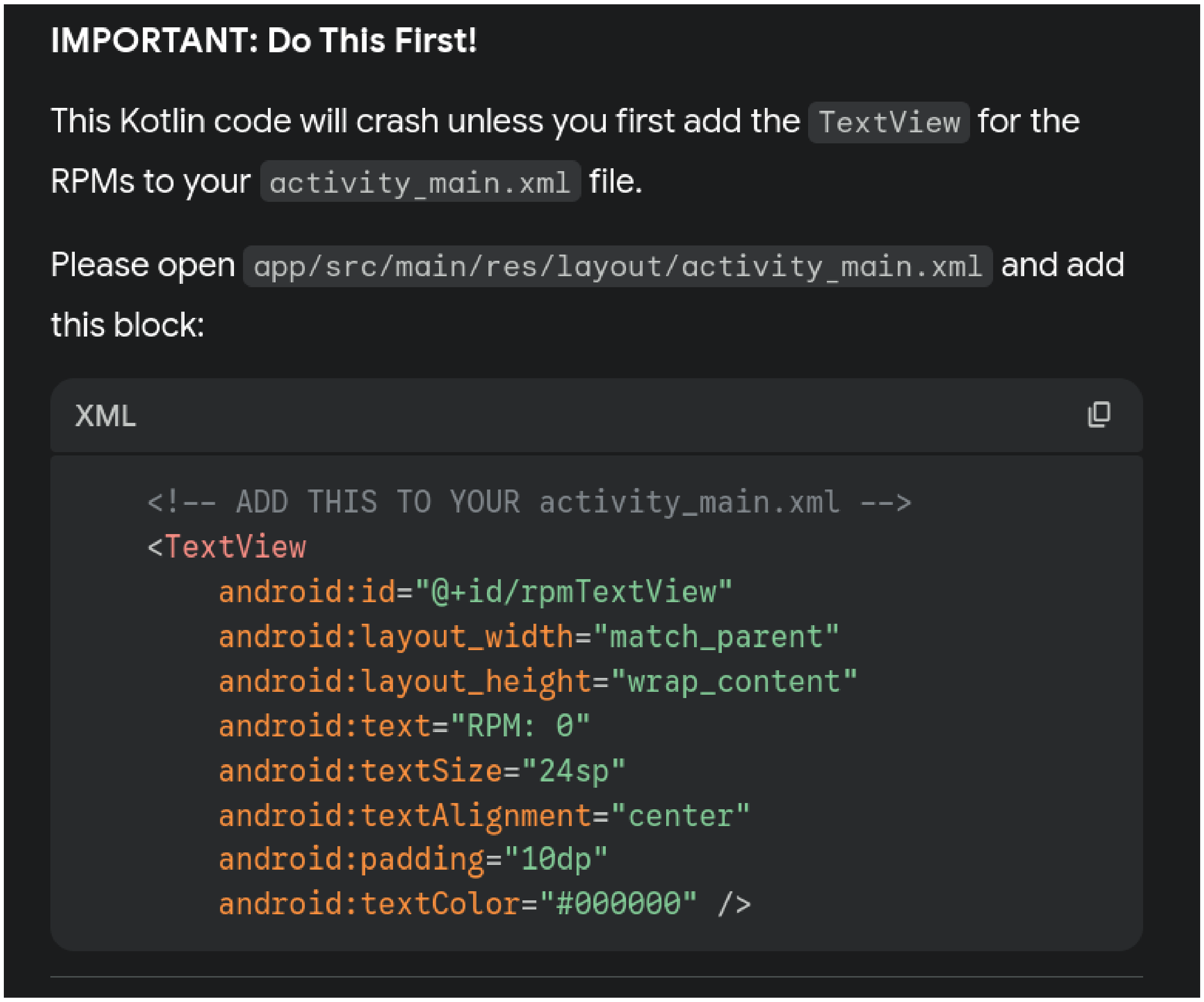


```
group-1@group1:~/Nevan_Aubrey_4235_AI_Project/CP2 $ sudo python3 bt_to_c_bridge.py
Starting C program: ./motor_control_v6
Bluetooth socket closed.
Stopping C program (sending 'q')...
C program stopped.
Server shut down.
Traceback (most recent call last):
  File "/usr/lib/python3/dist-packages/bluetooth/bluez.py", line 271, in advertise_service
    _bt.sdp_advertise_service (sock._sock, name, service_id, \
_bluetooth.error: (111, 'Connection refused')

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "/home/group-1/Nevan_Aubrey_4235_AI_Project/CP2/bt_to_c_bridge.py", line 84, in <module>
    bluetooth.advertise_service(server_sock, "MotorControlBridge",
  File "/usr/lib/python3/dist-packages/bluetooth/bluez.py", line 275, in advertise_service
    raise BluetoothError (*e.args)
bluetooth.btcommon.BluetoothError: [Errno 111] Connection refused
```

I didn't try to dig deeper into these errors because Aubrey told me she actually had a more updated C program that acted and still acts as a sophisticated local Bluetooth server. She predicted that the Python Bridge program would not be needed. So, I sent Gemini Aubrey's updated C script.

After Gemini saw the new C program, it pointed out that the new program used channel communication and the previous attempt was trying to use UUIDs. Gemini then gave a brief description on the difference and then started showing code edits. I asked for the entire kotlin file updated with the appropriate edits to allow the app to communicate with Aubrey's updated c script.

What is a bit funny is that Gemini had a really hard time just giving me the whole kotlin file. I had to ask it multiple times, and it just wouldn't. At one prompt, it said this:



IMPORTANT: Do This First!

This Kotlin code will crash unless you first add the `TextView` for the RPMs to your `activity_main.xml` file.

Please open `app/src/main/res/layout/activity_main.xml` and add this block:

```xml
    <!-- ADD THIS TO YOUR activity_main.xml -->
    <TextView
        android:id="@+id/rpmTextView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="RPM: 0"
        android:textSize="24sp"
        android:textAlignment="center"
        android:padding="10dp"
        android:textColor="#000000" />
```

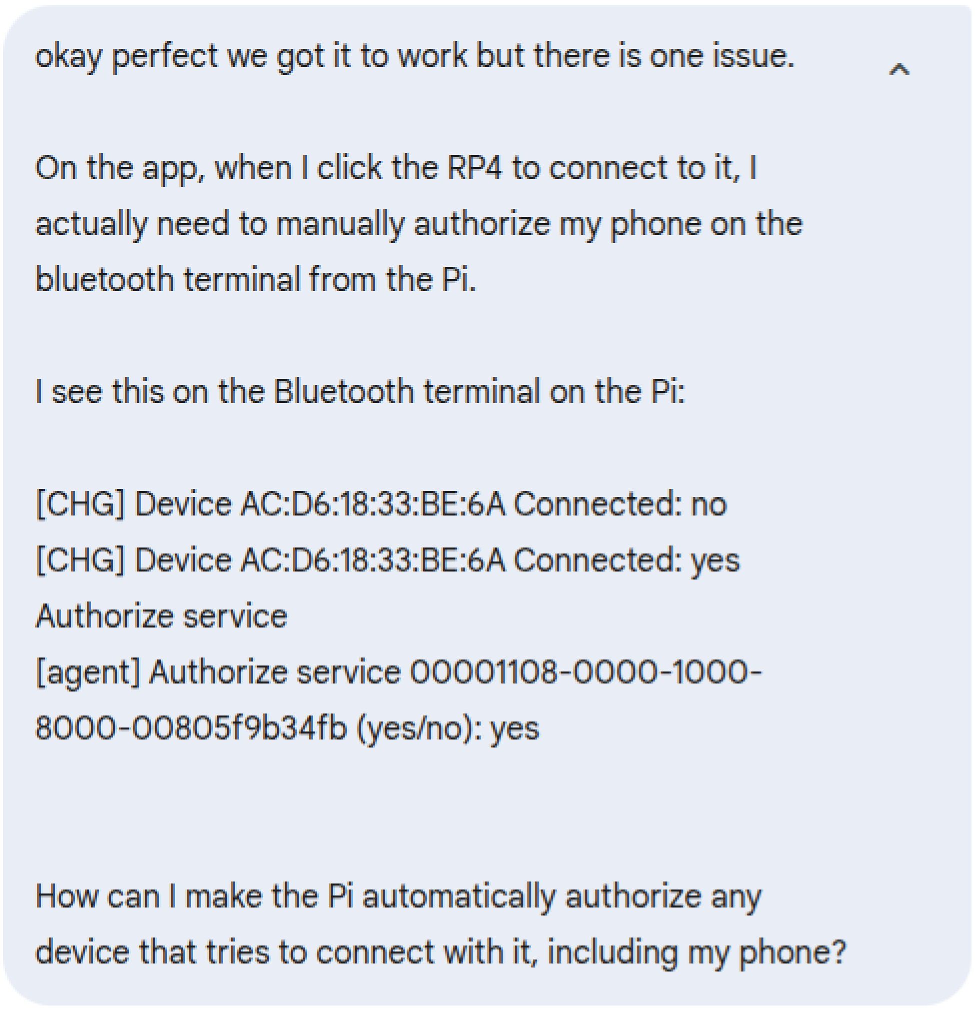
I then posted my updated xml so Gemini could find comfort in the fact that I did make these important edits. And then after one more try, it finally gave me the code to copy and paste into Android Studio. I guess sometimes I just need to give it what it wants for it to give me what I want.

Now that I had the app program and we had Aubrey's local server program, I asked Gemini what to do. Gemini then proceeded to give me a bunch of commands, but that didn't work. Aubrey told me to try certain terminal commands she had in her Gemini log and quickly deduced that her program actually has the server script always running in the Pi background. She made Gemini do this so the user doesn't need to interact with the Pi at all. So, I didn't have to do anything at this stage. As long as the app program was written such that it was able to communicate with the Pi program, nothing else needed to be done at this point.

I then tried testing the app on my phone. I was not seeing the nearby bluetooth devices pop up on the app. I provided Gemini the kotlin program and asked what was wrong. It told me it was likely an app permissions issue. It was – I didn't allow the app to access my location. I guess AI can be good at fixing simple human errors, like forgetting to turn on the power to a circuit and wondering why it's not working, or forgetting to give the app permissions and wondering why it's not working

Once I got the permission issue fixed, I realized that I actually needed to go into the bluetooth control terminal in the Pi to manually approve my phone trying to connect. At the time, we were working with a monitor but we wanted to move away from it to achieve stand alone mode.

I asked Gemini what to do to fix this issue:


okay perfect we got it to work but there is one issue.

On the app, when I click the RP4 to connect to it, I actually need to manually authorize my phone on the bluetooth terminal from the Pi.

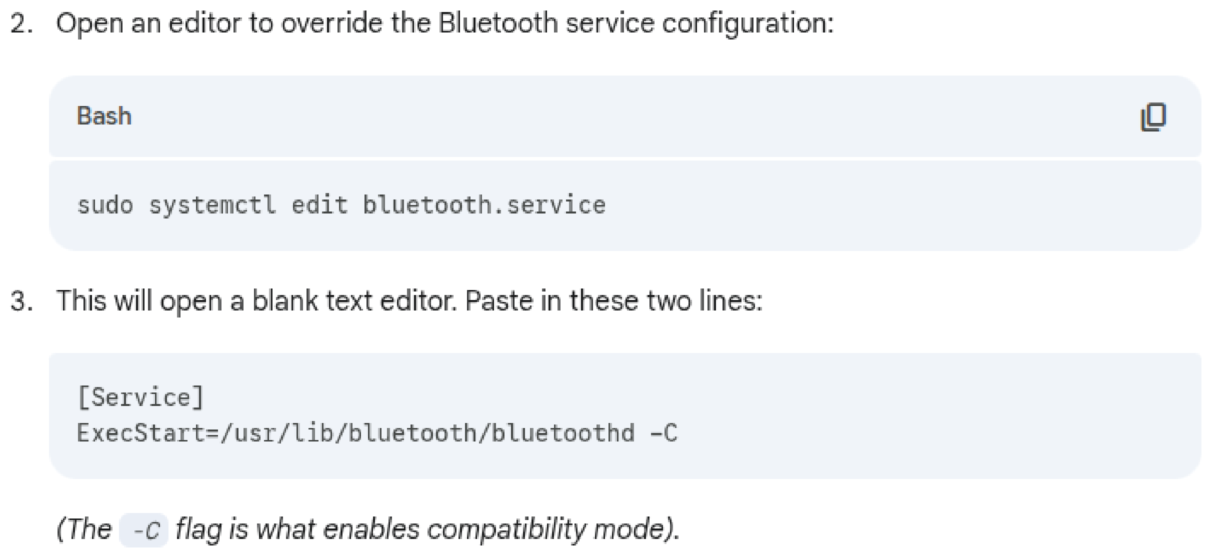
I see this on the Bluetooth terminal on the Pi:

```
[CHG] Device AC:D6:18:33:BE:6A Connected: no
[CHG] Device AC:D6:18:33:BE:6A Connected: yes
Authorize service
[agent] Authorize service 00001108-0000-1000-
8000-00805f9b34fb (yes/no): yes
```

How can I make the Pi automatically authorize any device that tries to connect with it, including my phone?

Gemini at first wanted to help me have the Pi remember my phone's specific MAC address and put it on a whitelist of sorts, but I was getting confused. So, I asked Gemini if there is a way for the Pi to accept all devices trying to connect to it via bluetooth.

Gemini warned me of the security risks of this decision, but eventually told me this important piece of information:



2. Open an editor to override the Bluetooth service configuration:

Bash

```
sudo systemctl edit bluetooth.service
```

3. This will open a blank text editor. Paste in these two lines:

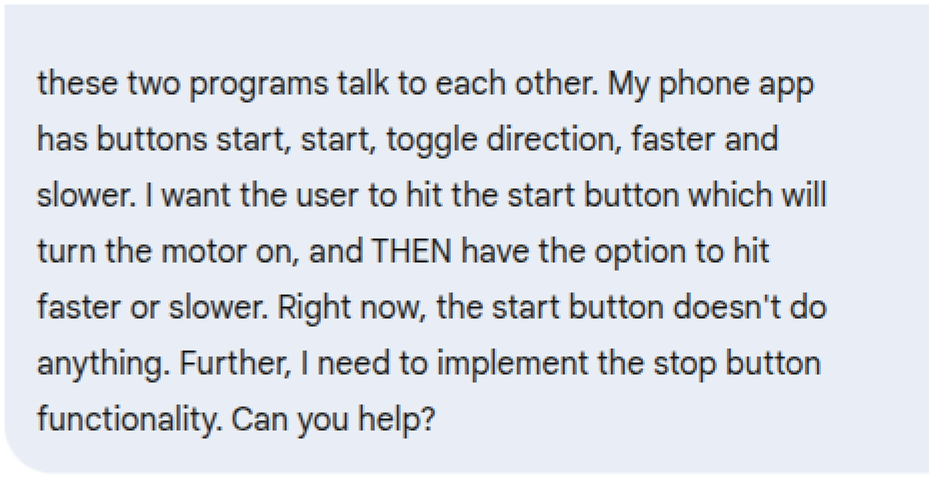
```
[Service]
ExecStart=/usr/lib/bluetooth/bluetoothd -C
```

(The `-C` flag is what enables compatibility mode).

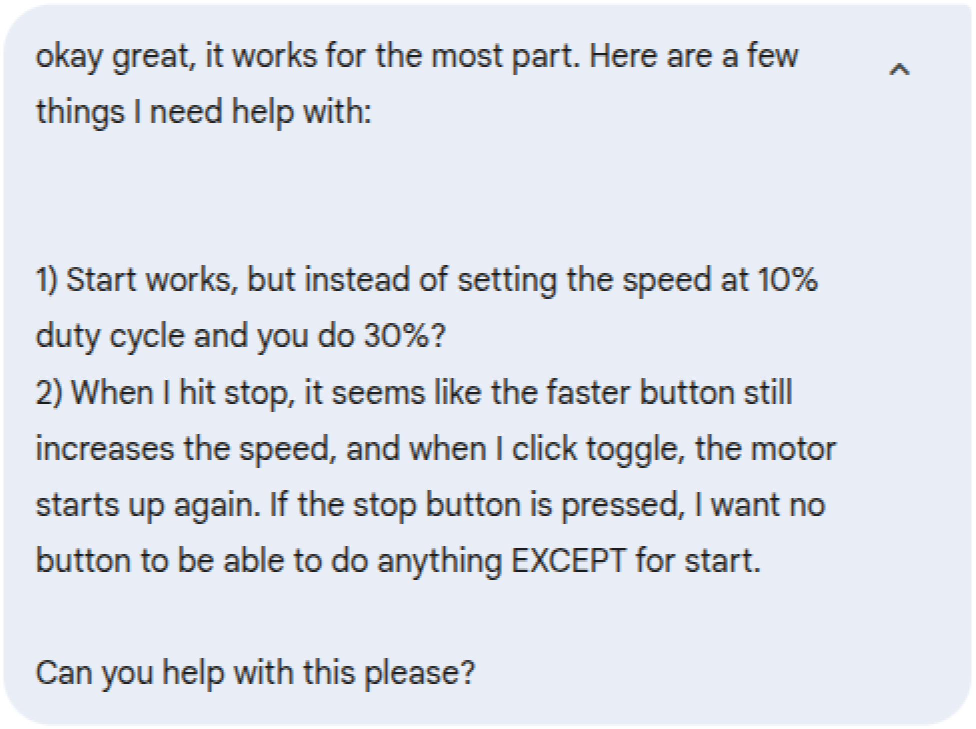
I did as Gemini said and it worked! The phone was able to connect to the Pi without manual approval from the Pi.

It worked for a while, but our next issue was our motor becoming extremely jittery out of nowhere. I thought maybe the GPIO from the Pi was being noisy, so I used an oscilloscope to check the GPIO waveforms and determine if duty cycle and signal integrity was acceptable, and it was. I thought maybe there were lapses in the signal downstream, so I put a capacitor across the motor wires because I thought it would smooth out any sudden bursts. But that did nothing. Audrey thought the issue might be with the power supply itself. Shockingly, if we left the power supply off for a decent amount of time (as if letting it "rest"), the motor ran smoothly. Perhaps there was a manufacturing issue with that specific power supply we were using at the time. Gemini wasn't that helpful in this stage of the working session.

The next issue was getting the button functionality to work. I gave Gemini the following prompt:


these two programs talk to each other. My phone app has buttons start, start, toggle direction, faster and slower. I want the user to hit the start button which will turn the motor on, and THEN have the option to hit faster or slower. Right now, the start button doesn't do anything. Further, I need to implement the stop button functionality. Can you help?

Gemini gave me the code, and I gave it this follow up prompt:


okay great, it works for the most part. Here are a few things I need help with:

1) Start works, but instead of setting the speed at 10% duty cycle and you do 30%?
2) When I hit stop, it seems like the faster button still increases the speed, and when I click toggle, the motor starts up again. If the stop button is pressed, I want no button to be able to do anything EXCEPT for start.

Can you help with this please?

Gemini gave me the code, and it worked! At this point I just wanted to change the RPM display text color from black to white, which was a very simple fix, as Gemini pointed out:

XML

```xml
    <TextView
        android:id="@+id/rpmTextView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="RPM: 0"
        android:textSize="24sp"
        android:textAlignment="center"
        android:padding="10dp"
        android:textColor="#FFFFFF" /> <!-- <-- THIS LINE IS CHANGED -->
```

That was the end of the significant interactions for this checkpoint 3. I asked Gemini to write an interaction log summarizing the chat, which is attached in this checkpoint repo, if the reader would like further information on the AI interactions.

At the end of our working session, we had the peculiar issue of the Bluetooth connection working only when the monitor was in use. Turns out, the shell script Aubrey uses to make the Pi run the server program on boot takes some time to set up video media (like the monitor) and then do the necessary bluetooth initializing actions. However, without the step of setting up the video media, the Pi enters a sort of race condition, where some steps in the initialization setup happen out of order. With some timing adjustments on the c program's end, the code worked. More information on this can probably be found in Aubrey's AI interaction log since it was her Gemini interactions that gave us this hint.

Further Analysis

I noticed Gemini had a tendency to show me the specific sections of the main activity code to edit, rather than just giving me the whole script to copy and paste. I oftentimes had to coerce Gemini to give me the whole kotlin code, asking it many times, giving it my current kotlin code, etc. I think this might be due to the fact that I typically use AI in a very targeted, pointed way. I remember in the stopwatch project, I tried to just give AI my prompts and take an updated version of the whole code each time, but the code just straight up didn't work. So, I focused on asking it very targeted questions on very small sections of code, to see if I can find the issue myself. In other words, I typically don't treat code like black boxes when working with AI, because if the AI messes up, I have no way of knowing where to start debugging. Rather, I treat the code like a white box – a system whose inner dynamics, structures, and logic I need to understand myself, and something I can use AI to better understand.

However, I’ve never made a phone app before, and the purpose of this project is to see if we can leverage AI to create an effective app quickly despite this lack of experience. Thankfully, Gemini did a great job at just giving me code that works, so I didn’t need to spend a lot of time reverse engineering what Gemini did to try to understand and solve the hidden problem, much like what I did for stopwatch. Perhaps AI has a better time writing high level code like kotlin than a low-level language like ARM assembly, maybe because the AI is not that in tune with the hardware. It is a model at the end of the day, a model with a whole lot of abstraction and indirection.

User docs (software docs, user manual, etc) are still in the works, but we plan on using Gemini to make those as per the project document.

References

1. Gemini 2.5 Pro
2. Claude 4.5 Sonnet