1) Completed Stuff

The full project repository can be found here (some branches may not yet be merged to main)

Bluetooth communication

I've successfully implemented and tested Bluetooth Low Energy (BLE) communication between the RPi Pico and my desktop PC. The microcontroller periodically transmits a 16-bit measurement over BLE using the MicroPython `aioble` library (and a configured timer interrupt). The desktop PC receives and stores each of these measurements, to a list, using interrupt requests over a set period of time. The `Bleak` Python BLE library provides us with this bluetooth interface.

RPi Pico MicroPython Script

```
import aioble
import asyncio
import bluetooth
from machine import Pin
from sense import Sensor

# Device name
DEVICE_NAME = "temp-sense"

# Custom service
SERVICE_UUID = bluetooth.UUID(0x9999)

# org.bluetooth.characteristic.analog
ANALOG_UUID = bluetooth.UUID(0x25A8)

# org.bluetooth.characteristic.gap.appearance.xml
GATT_APPEARANCE_GENERIC_SENSOR = const(1344)

ADV_INTERVAL_US = const(250000)

LED_GPIO = const(16)
ADC_PORT = const(2)
SAMPLING_RATE_MS = const(500)
TIMER_ID = const(-1)

async def main():
    service = aioble.Service(SERVICE_UUID)
    characteristic = aioble.Characteristic(service, ANALOG_UUID, read=True,
notify=True)
    aioble.register_services(service)

    Sensor(adc_port=ADC_PORT, sampling_rate_ms=SAMPLING_RATE_MS,
timer_id=TIMER_ID, bt_char=characteristic)

    led = Pin(LED_GPIO, Pin.OUT)
```

```
    led.value(0)
    while True:
        async with await aioble.advertise(
            ADV_INTERVAL_US,
            name=DEVICE_NAME,
            services=[SERVICE_UUID],
            appearance=GATT_APPEARANCE_GENERIC_SENSOR,
        ) as connection: #type: ignore
            led.value(1)
            await connection.disconnected(timeout_ms=None)
            led.value(0)

if __name__ == "__main__":
    asyncio.run(main())
```

Data processing

I've added data processing functions that get accessed by a desktop Python script. These functions take the 16-bit readings, convert them to temperatures, and use the `scipy` library's least squares linear curve fit function to generate the coefficients for a a decaying exponential equation that represents the temperature curve of a hot liquid cooling down to room temperature.

Desktop Data Processing Script

```
import argparse
import asyncio
import data_processing as dp
import psql_db
from client_bt import ClientBT

_ANALOG_UUID = "000025a8-0000-1000-8000-00805f9b34fb"
_DEVICE_NAME = "temp-sense"

async def main():
    parser = argparse.ArgumentParser()
    parser.add_argument('-s', '--sampling_duration', type=int, required=True)
    sampling_duration = parser.parse_args().sampling_duration

    db = psql_db.Database
    db.get_params()

    bt = ClientBT(device_name=_DEVICE_NAME, analog_uuid=_ANALOG_UUID)
    await bt.receive_measurements(duration=sampling_duration)

    (a, b, T_amb) = dp.get_decay_params(data=bt.data)

    db.connect()
    db.create_table()
    db.add_record(duration=sampling_duration, a=a, b=b, T_amb=T_amb)
    db.disconnect()
```

```
if __name__ == "__main__":
    asyncio.run(main())
```

PSQL Database

I've added methods for storing and retrieving the coefficients of the generated decaying exponential equation using `psycopg2`.

Graphing

I've also created a separate desktop Python script for retrieving the decaying exponential equation coefficients (from the PSQL DB) and plotting it using `numpy`.

Desktop Graphing Script

```
import psql_db
from data_graphing import plot_temp_func

if __name__ == "__main__":
    db = psql_db.Database
    db.get_params()
    db.connect()
    records = db.get_all_records()
    for record in records:
        print(record)

    db.disconnect()

    data_id = int(input("Enter data_id for desired dataset: "))
    for record in records:
        if record[0] == data_id:
            duration = record[2]
            a = record[3]
            b = record[4]
            T_amb = record[5]

    plot_temp_func(duration_s=duration, a=a, b=b, T_amb=T_amb)
```

2) Incomplete Stuff

Data processing

I'm awaiting the arrival of an insulated negative temperature coefficient (NTC) thermistor. This piece of equipment will enable the measurement of a liquid's temperature without having to worry about the additional parallel resistance that occurs from the liquid shorting the terminals of the thermistor. I need to run a test to make sure that my conversions from analog measurements to degrees celsius are correct.

Optionally, I may want to export the analog measurements and manually curve fit them to a temperature curve using MATLAB. By comparing this MATLAB generated curve to the one generated in my Python script, I may be able to catch minor errors or discrepancies.

PSQL Database

I still need to install and setup the local PSQL server that the Python scripts will interface with on my desktop computer. I also need to do a test run to make sure that the scripts properly create the temperature coefficients table and insert/retrieve records.

Graphing

After retrieving a record from the PSQL table, I need to test the graphing script to make sure it properly plots the temperature curve.

Required Resources

- Local PSQL server
- Insulated NTC thermistor
- Test data processing script
- (optional) Supplementary MATLAB test script to validate curve fitting
- Test graphing script