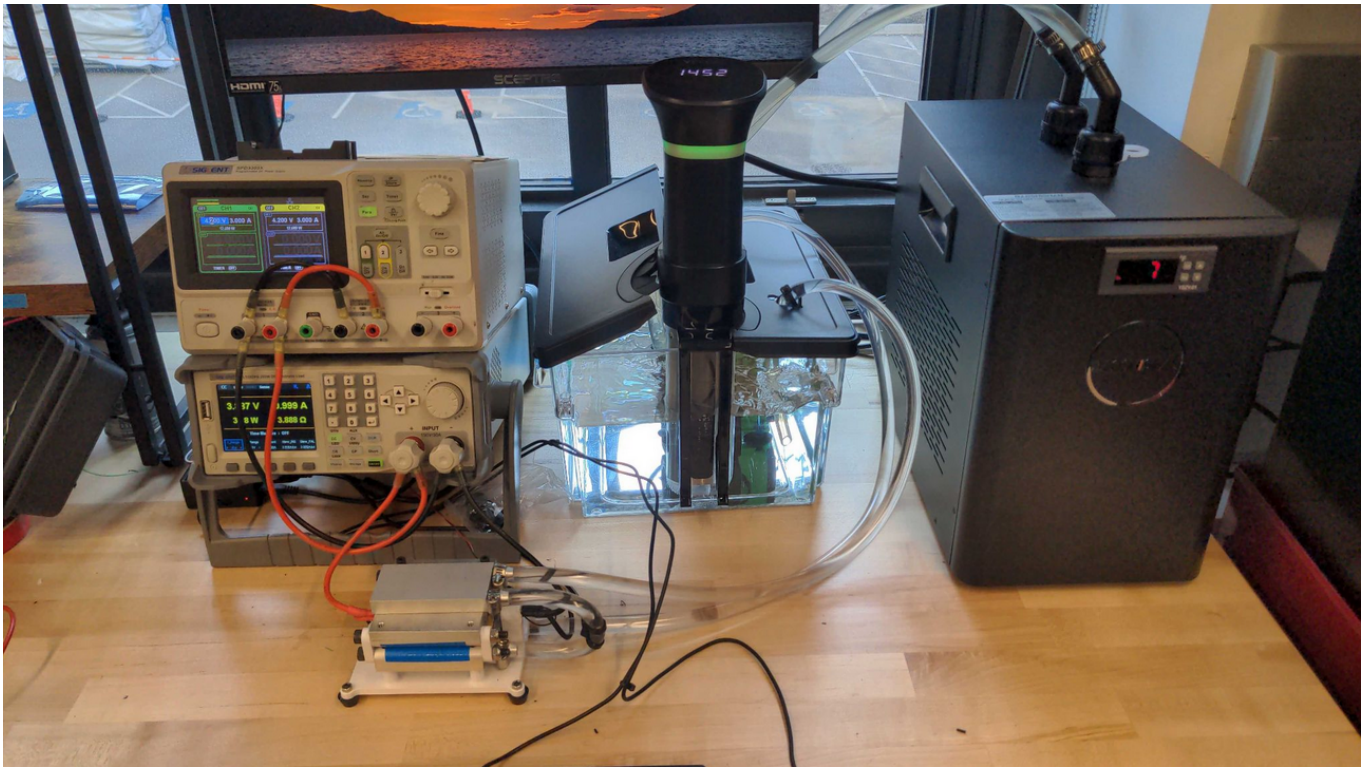
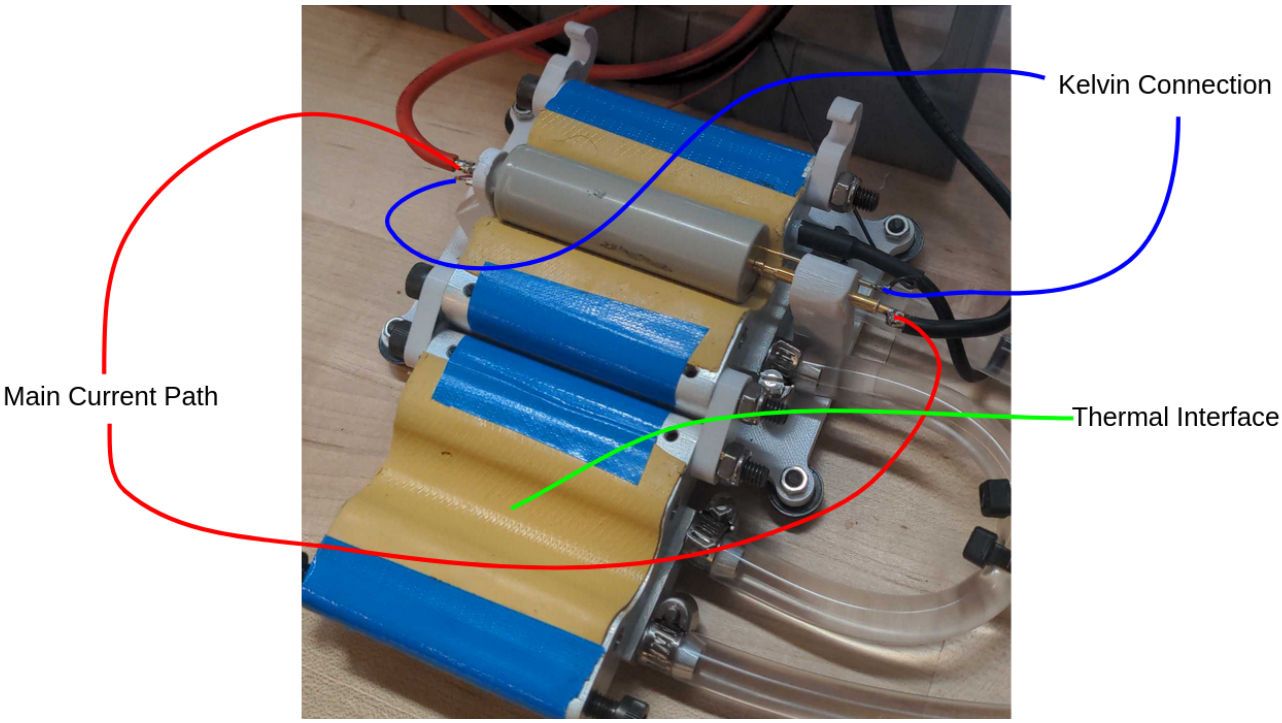
The cell tester allows us to repeatedly charge and discharge cells in order to observe their response to a controlled current input. This is important for cell modeling and characterization.

I have just finished doing a sweep of tests for the P42A at 25C. I want you to try the P45B at 25C next. They are on my desk in the cardboard box. The texts below will explain how to use the celltester to do this.

Hardware setup



The hardware of the cell tester is comprised of a list of things. From left to right, the raspberry pi, the electronic load and supply, the cell holding jig, the sous vide and water storage, the **two** pumps inside the sous vide tank, and the aquarium chiller.



The cell testing jig is what interfaces the cell with the cell tester. It serves multiple purposes. First, it provides good electrical contact using pogo pins, or spring pins to carry the high currents of the cell. It also provides a four-wire kelvin connection to allow for accurate voltage sensing. Secondly, it is glued to water cooling blocks that allow for the temperature of the cell (or at least the outside surface) to be held constant during charging and discharging. The cell interfaces with the aluminum sinks using the thermal interface material, or the `cheese`. Finally, the block also has a hole for a temperature sensor to be inserted, allowing for the raspberry pi to record temperature through a cycle.

Plugging in the equipment

These items all need to be plugged in for the cell tester to function, so take some time at the beginning to see where all things are plugged in and make sure that it is highly unlikely that they will be unplugged for the rest of the day. Each item has a switch that can be used instead of unplugging, except the raspberry pi which doesn't ever need to be unplugged or the **two** pumps, each which will need to be manually unplugged at the end of the day. The two outlets for the plugs are right behind the black tool box to the right of the cell testing setup.

Finally, the cell can be inserted into the cell testing jig.
EXCEPT YOU SHOULDN'T UNTIL YOU READ THE ORDER OF OPERATIONS!

Inserting the cell

First, compress the negative side pogo pins with the negative side of the cell. Then, push the cell down into the `cheese`. Make sure the positive side of the cell is touching the positive electrode, so maybe slide the cell upwards to make them mate. Then, sandwich the cell

between the two aluminum blocks. The two clips at the back will make sure even pressure is applied between cells, and is easy to engage.

Removing the cell

To remove the cell, first undo the latches at the back. Maybe use a screwdriver to push them if it is too hard. Then slowly lift the top block. The `cheese` is sticky, so be slow when prying it open. If the `cheese` flares out from the aluminum, just pat it down. Removing the cell from the bottom piece is quite hard. It might be tempting to use a hand tool, but don't. Prying from the positive side of the cell is the most effective, just to break the stick of the `chese`. If you can, maybe find a very thin plastic or string to lie under the cell, so that you can pull up on it to get a better grip to pull up the cell. Put that string closer to the positive side.

Order of operations

This is probably one of the most important things to know when operating the cell tester. There is a **specific order** in which certain actions must be done when powering on the celltester, otherwise catastrophic damage can occur. Likewise, while powering off the celltester, the same caution must also be used.

The problem lies specifically with the insertion time of the cell into the cell jig. For some reason, while the electronic supply specifically is booting (when the screen displays the siglent logo at startup), the **output of the supply is a dead short**. This is bad because the cell doesn't like to have its terminals shorted, and is capable of discharging over 100A. On top of this, it is possible to damage and break our supply, which can lead to more catastrophes down the road.

Note: I figured all of this out by doing exactly the opposite of what I described, and found that in the span of 2 seconds the cell desoldered the sens wire in the last jig I had. The supply took it like a champ, and no damage seems to have occured.

It is very easy to absentmindedly do these steps out of order. So what should you do to avoid this?

Step 1: When **powering up** the cell tester, make sure the **last** operation you do is inserting the cell into the cell jig.

Step 2: When **powering down** the cell tester, make sure that the **first** operation you do is removing the cell from the cell jig.

Nothing should go wrong if you follow these simple steps.

If something goes wrong

Elijah and I have bought sand buckets and placed them directly underneath the cell tester. These will suppress a battery fire. The cell tester only uses a single cell, and therefore the total damage possible is somewhat minimized. Despite this, the cell is still capable of bursting into flames (very easily done accidentally, for example forgetting that the voltage setpoint (which is not very easily visible on screen) of the supply is at 10V (instead of 4.2V duh...)). To suppress a lab bench battery fire, I think the best action is to dump the sand on the table on the jig. Have some finesse and drizzle it on top to concentrate the sand on the jig.

If you are not willing to do this, then step back and call the fire department to take care of it. MIT's policy is to not fight fires. Just keep in mind that we might be out of the lab for a while after a battery fire...

Typical order of operations

1. Make sure that there is no cell inside the cell jig.
2. Power on the load and supply.
3. Turn on the sous vide, making sure the temperature is set to a desired value and that the timer won't run out (set it to 12hr+ it doesnt matter)
4. Turn on the aquarium chiller to reject slow heat accumulation
5. Plug in both pumps (make sure the water is high enough, above the green pump). Add some more distilled water found underneath the table if needed.
6. Insert the cell. Make sure to leave it there for at least 5 min after the water has come up to temperature to let the temperature of the cell equalize.

Software setup

The raspberry pi has all the code necessary to run the load and supply. Login using the lab code.

Use <ctrl + alt + t> to open a shell. Navigate to the correct directory using cd.

```
cd ~/BatteryResearch/cellTester
```

In this folder you will find many files.

```
user@user-GWTC71427:~/Documents/BatteryResearch/cellTester$ ls
CAD       cycle_cell.py   device_bridge.py  graphing    info.log     pyvisa_note.txt
celldata  cycle_cell.py~  docs              HF_testing  __pycache__  temp_sens.py
```

Some files that are of importance:

- info.log: A log file that keeps track of what the cell tester is doing with timestamps. Open and read the bottom of this file if you don't know what the cell tester did.
- celldata/: A directory containing all of the data collected by the cell tester. Inside you can make new directories to collect data on different cells or temperatures.
- temp_sens.py: script used to asynchronously measure block temperature
- current.temperature: file used to store most recent temperature reading and time (not shown above cuz of .gitignore not uploading it)
- cycle_cell.py: main script that commands load and supply intelligently

Using the cell tester

Temperature sensing

First, we should run `temp_sens.py` to have the raspberry pi read temperature asynchronously to our main code. Either open a new terminal, navigate to the correct directory, and run

```
cd ~/BatteryResearch/cellTester
python temp_sens.py
```

or run

```
python temp_sens.py &
```

in the shell you have already opened to spawn a child process that will run in the background and detach the temperature sensing from the shell. Hit enter to clear the line, visually resetting back to normal.

If you forget to do this step or have done it incorrectly, the main script will inform you and refuse to run.

If you would like to watch the temperature live during a discharge, we can use the `watch` command in linux. Open a new terminal, navigate to the `cellTester` directory, and run

```
watch -n 0.1 cat current.temperature
```

Main script

To run the main script and have the cycler begin testing, we run `cycle_cell.py`. This is the only file you will need to edit in order to change the behavior of the cell tester, for example

quitting one day and resuming testing at the same point the next day. The code block I think you are most likely to change has been marked with

```
### BEGIN MIGUEL

### END MIGUEL
```

The code block as of last git commit looks like this:

```
    #### BEGIN MIGUEL ####

    # do cycle at 1 Amp
    # this is extremely slow, a 4.2 Ah cell should take 4.2 hours
    # will actually be slightly less, due to voltage sagging +
    # termination occuring at voltage setpoint
    charge_cell(data_path, 6)
    time.sleep(60) # allow cell dynamics to die out
    discharge_cell(data_path, 1, pulse_current=None)

    # cycle with intermittent pulsing
    # pulse at 2x discharge current
    #currents = [1, 5, 10, 15]
    currents = [5, 10, 15]
    for current in currents:
        # wait 1 min for cell to recover
        time.sleep(60)

        charge_cell(data_path, 6)

        # wait 1 min for cell to settle
        time.sleep(60)

        discharge_cell(data_path, current, pulse_current=current*2)

    #### END MIGUEL ####
```

Inside contains the logic for what the celltester will do when you run the main script. For example, if you run this script it will first attempt a 6A charge followed by a no pulse 1A discharge. The discharge will take about 4.2 hours, not including the charge time. If you run out

of time in a day and need to pick up next day, you can comment out these lines to have the cycler progress with the rest of the code.

Speaking about picking up the next day, the only appropriate time to cancel a cycle in progress is only when the cell is charging. There are no repercussions to stopping the cycler while charging since we are not necessarily invested in capturing what the cells do while charging. **Stopping the cycler when a cell is discharging will make the data unusable.** This is fine, if you are willing to repeat the discharge. A proper discharge curve is completely uninterrupted from start to finish. The cycler will still log all data, even while charging, just because I can.

Also make sure that the code that you write always has a charge cycle before a discharge cycle. For each discharge cycle to be usable, we need to make sure the cell is in a predictable internal state which we can achieve by charging using the same parameters beforehand. Just call the function to charge at 6A like in the code above.

Making changes to a file

If you are a noob, use nano. If you are a chad, use vim.

```
nano cycle_cell.py
```

Running the main script

Type

```
python cell_cycle.py
```

The script will ask which cell you are currently using. It will show a list of possible entries. If you would like to make an entry that doesn't exist, exit out of the script (ctrl + c) and to to the celldata directory. Make a directory that corresponds with your testing configuation for cell and temperature before running the script, for example:

```
cd celldata
mkdir P45B_50C
cd ..
python cycle_cell.py
```

The script will search the celldata directory and allow you to choose from the the directories inside. Type the number in the list that corresponds to the testing configuration that you are doing.

Stopping the cell tester

Simply (Ctrl + C). The main script catches the shutdown signal and makes sure to turn off both the supply and load.

Bugs in your code

Please confine yourself to working in the area that I have designated. As far as I am concerned, all of the code I have written is in working order. Any small change may have unexpected results and in this case may come with real consequences. If at any time the main script errors and terminates without shutting down the supplies, try a few of these options to turn off the supplies:

1. Try run the script again. The script begins by connecting to, and zeroing all values in both the load and supply, before asking which configuration is being used.
2. Manually turn off the supply and load using the interface, NOT THE POWER BUTTONS WHICH COULD SHORT THE OUTPUT (THE SUPPLY SPECIFICALLY). The load will require you to press shift + another button to escape remote control mode (it will tell you on screen), but the supply is much easier and will just shut off with one button.
3. If all else fails, remove the cell while a charge or discharge is occurring by hand. It should never come to this but IDK.

What I want you to do

Cycle the P45B at 25C. Get each of the following curves:

1. 1A discharge with no pulses
2. 1A discharge with pulses
3. 5A discharge with pulses
4. 10A discharge with pulses
5. 15A discharge with pulses

The pulses should be 2x the discharge current. The code currently on github reflects this. The code on github as it is right now should capture all the curves listed above if it were left to run continuously. Note the max discharge current possible by the load is 30A.

I do not care to collect data for charges, feel free to use them as pause points for turn off the cell cycler.

Make sure to pull the code I have written by using git! Read below to find out how.

Uploading cell data to github

At the end of the day, it is a good idea to backup all the data collected. We will use git to do this. The celltester pi has a key to the repo, so you can do it all from the raspberry pi. I can also give you a personal key if you want to remotely view the code on your laptop.

Adding the key to ssh-agent

Every time you open a shell / terminal to use github, you need to add the ssh key to have the appropriate permissions to access the github. Do this by typing:

```
ssh-add ~/.ssh/id_rsa_git
```

That might not work because I don't exactly remember the file name of the key, but choose the file in the `.ssh/` directory found in the home directory that doesn't end in the extension `.pub`.

Pulling my initial changes

Run

```
git pull
```

Once.
Yay!

Pushing changes to github

Once you have finished testing or have made changes to the code that you want to push, type the following:

```
git add --all
git commit -m 'a short description of the code that I, Miguel, have written'
git push
```

If something goes wrong, ask someone else in the lab to help you.

Recap + Safety

While discharging, it is important that someone be in the room while cycling is occurring at the very least. The easiest and best way to observe cycling is sitting nearby and periodically

checking, making sure that the state of the tester makes sense and that it is working. It should be ok to leave for brief moments, moreso when the cell is cycling at lower currents. Take the situation seriously and don't be afraid to turn off the cycler if you need to.

Recap:

1. Don't do the order of operations incorrectly.
2. Make sure to turn everything on, and let the cell get to temperature before starting testing.
3. Navigate to `cycle_cell.py` and make appropriate changes to what the cycler should do next.
4. Start temp sensing
5. Start `cycle_cell.py`
6. Wait...
7. Periodically check on system
8. Either stop or wait to finish
9. Upload data
10. Remove cell from jig (FIRST!) (Don't do the order of operations incorrectly)