VTIPY2

Introduction

Welcome to vtipy2! This software can be used to perform variable temperature electrochemical impedance spectroscopy using temperature controlled stages from *Linkam* and impedance analysers from *Solartron Analytical* and *Biologic*. This software provides an interface to automate these experiments, but much of the code could be readily adapted to accomodate similar (or very different) functionality, using similar hardware.

This software was developed as part of the *Faraday Undergraduate Summer Experience* program led by *The Faraday Institute*. This sofware was developed as part of and with supervision from the quantum materials group at the University of Cambridge.

Quick Start / Example

Provided an environment has been fully setup to begin experiments, this quick start guide can be followed to take measurements without concern for setup. This section would be most useful to a new user using an infrastructure which has already been setup, as per the *Setup and Dependencies* section.

1. Ensure desired hardware is connected.
    - Both the *Biologic SP-200* and *Solartron 1260* can both be connected to the PC, but ensure that only one stage is connected to the PC.
    - **IMPORTANT**: Double check that the connected stage is the stage you wish to use, and remember to enter the *same* stage name in the next step. Due to time constraints, I wasn't able to succesfully implement a 'detect stage' functionality. This is addressed in the *Issues and Future Functionality* section.
2. Start correct *python* environment and run `vtipy2_setup.py`.
    - (Windows only) Start a terminal instance where there should be a 32 bit version of *python* installed, with all of the correct dependencies. This terminal instance could be a *powershell* window or an *anaconda* prompt, for example.
    - Navigate to the *vtipy2* directory on your machine, by using `ls` and `cd` commmands in *powershell*, for example.
    - Enter the command `python vtipy2_setup.py`.
    - **IMPORTANT**: Enter the correct stage name, as reminded before.
    - Enter the name of the analyser you wish to use, as prompted.
    - If all has gone as planned, `python vtipy2_setup.py` should return a 'Setup Successful!' message. The purpose of this command is to setup and debug the equipment, so hopefully any errors returned should be informative in rectifying any issues in the setup. Once errors are rectified, run the command again, until all errors are fixed.
    - *Optional*: To further check the analyser, `python test_analyser.py` can be used, to ensure the analyser is reading out reasonable numbers. It should perform an impedance scan and plot the result.
    - *Optional*: To further check the stage, `python stop_+_temp_plot.py` can be used. This provides a live plot of the current temperature of the stage. This command also sends a command to the stage to stop heating, so can also be used if you need to stop heating in an emergency.
3. Run `vtipy2.py`.
    - Provided `python vtipy2_setup.py` returned a 'Setup Successful!' message, `python vtipy2.py` can now be run, and should not return any errors. If any are, it could be a sign of a

wider environment issue, please see the *Setup and Dependencies* section.

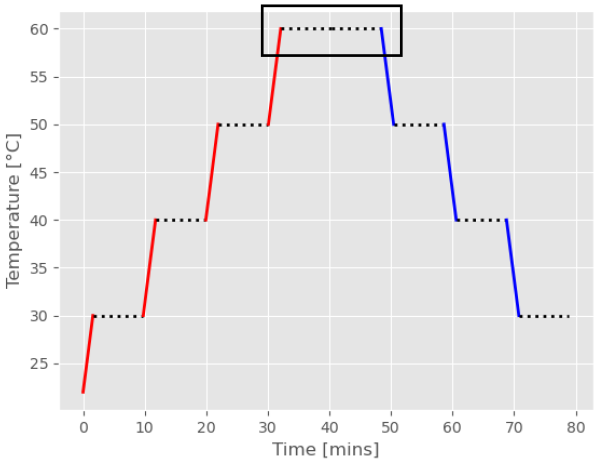
- Enter an experiment name, and some notes on the experiment, then press *Begin*.
- Enter ramp parameters for your experiment for the ramps required. Most are self-explanatory, perhaps apart from the last one 'Scan @ first T'. Please see the relevant section below for more details. Once all required ramps have been input, press *Finish*.
- Examine the displayed schematic and any presented warnings. If satisfied, press *Begin Experiment*.
- If an emergency stop is needed, press the **STOP** button, at the bottom left of the interface.
- Once the experiment is complete, the **STOP** button will become a **FINISH** button, which will neatly close the application.

The "Scan at First Temperature" Parameter

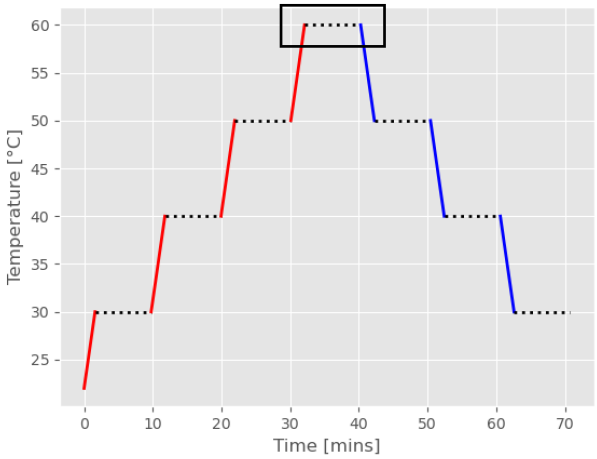
On entering the parameters for a ramp, the last parameter 'Scan @ first T' could cause confusion. It asks the question: *would you like to perform an impedance scan at the first temperature point of the ramp?*

This becomes relevant whena ramp starts at the same temperature that a previous ramp ended, as below, being 60°C. On the left of the figure is a plot of two ramps both scaling from 30°C to 60°C in steps of 10°C, where the 'Scan @ first T' parameter is set to 'y' for both the ramps. As highlighted by the black rectangle, this causes the temperature point of 60°C to be scanned twice (shown by the two consecutive dotted lines). Whereas on the right hand side of the figure, a plot is shown where the second ramp has the 'Scan @ first T' parameter set to false, or 'n'. This ensures that only one scan is carried out at the 60°C mark.

This paramater is left as an option as both behaviours may be desired by a user.



yes, yes



yes, no

Available Hardware

In order to carry out variable temperature electrochemical impedance measurements, means to heat and cool a sample alongside means to measure its impedance are required. To heat/cool samples the *HS350-PB4* and *TS1000-PB4* stages are used in the original setup for which this software is intended, and they are both reflected in the code. To measure the impedance of the sample, the *Solartron 1260* from *Solartron Analytical* and the *SP-200* from *Biologic* are used. Both are reflected in the capability of the application.

Software

---

virtual .py

To demo or to aid in any equipment setup, this `virtual.py` file is available. In order to test hardware and software behaviour, perhaps to add new hardware, 'virtual' stages and analysers have been made available in

this file. These can be elected to be the chosen stage and/or analyser for the experiment, such that no real hardware is interacted with. These choices can be made in `vtipy2_setup.py`. These virtual pieces of hardware simply return random numbers or predictable behaviour when necessary.

solartron .py

The `solartron.py` file contains the `solartron1260` class, used to interact with the *Solartron1260* impedance analyser. It uses the NI GPIB interface to pass commands to the analyser. To perform an impedance scan, the `measure_impedance` method should be called. Returned is a thread that measures the impedance. This thread can be monitored from an external calling program. For example, `thread.isalive()` will be `False` on completion of the scan.

For more information about the analyser and how to use it, consult the *Solartron 1260* manual.

biologic .py

The `biologic.py` file contains the `SP_200` class, used to interact with the *SP-200* impedance analyser. Unlike the `solartron1260` `measure_impedance` measurement, programmatic control is carried out by the unit itself, once commands are passed. An `SP_200()` object simply passes the parameters and waits until data is presented to it by the physical unit. The *potentio electrochemical impedance spectroscopy* technique is used on the device. It returns data in a format indentical to that of the `solartron1260` class. Also similarly, a monitorable thread is returned by the `measure_impedance` method.

linkam .py

The file `linkam.py` contains the functionality of the *HFS350-PB4* and the *TS1000-PB4* stages. It primarily contains means to communicated with the Linkam software development package, used to programmatically control *Linkam* stages. To do so, it requires a valid software licence file which can be obtained from *Linkam*. It should be named `linkam_license.lsk`. Please see the Setup and Dependencies section for more information on what dependencies the *Linkam* functionality requires.

vtipy2_setup .py

The file `vtipy2_setup.py` is a setup script for the wider application `vtipy2.py`. It asks the user for input of their choice of the available stages and analysers. It aids in the connection of the PC to the hardware, and raises errors cleanly should there be any such that a user should be able to easily debug them. It produces a file that is required in order to successfully start *vtipy2*.

IMPORTANT: As mentioned in the quick start guide, ensure the selected stage is the stage that is physically connected. For more information see the *Issues and Future Functionality* section.

test_analyser .py

The script `test_analyser.py` is a script to test the validity of a physically connected analyser. It requires the hardware configuration from `vtipy2_setup.py`, so therefore assumes a successful connection. Some user input is taken to specify values for an example impedance sweep, which is then carried out. The data returned by the sweep is then plotted in the form of a *Nyquist* plot. An experimental setup user should then use this to verify that the analyser is outputting reasonable data.

stop_+_temp_plot .py

The script `stop_+_temp_plot.py` can be used to primarily stop any heating functionality of a connected stage, perhaps in an emergency. It can also then be used to check and monitor the temperature of the stage, being as it displays a live temperature plot of the stage.

vtipy2 .py

The file `vtipy2.py` is the main application file of *vtipy2*. It programmatically controls a stage and analyser to perform variable temperature electrochemical impedance spectroscopy. It also displays an experiment

dashboard and a temperature live plot, such that a user can monitor the state of an experiment. See the *Quick Start / Example* section for a guide on the usage of *vtipy2*.

To communicate with hardware, it is dependent on the `virtual.py`, `solartron.py`, `biologic.py` and `linkam.py` modules, assuming that all available hardware is to be used regularly. The mentioned modules should come stored in a 'hardware' directory.

Setup and Dependencies

Python

A 32-bit python environment is needed to be able to use all of the available hardware, although 64-bit environment is possible should the use of the *Biologic SP-200* not be needed. Below is a list of external python libraries that should be installed in the environment:

- numpy
- matplotlib
- ctypes
- microscope
- win32api (test for your setup, may not be needed)
- pyvisa (solartron use)

Linkam

In order to use the *Linkam* SDK to control *Linkam* hardware, the following files must be in the working directory of *vtipy2*:

- linkam_license.lsk
- LinkamSDK.dll
- LinkamSDK.pdb

The file *linkam_license.lsk* is a software license file required to use the development kit. A valid license file can be obtained from *Linkam*.

IMPORTANT: *Linkam* provides versions *LinkamSDK.dll* and *LinkamSDK.pdb* for both 32 bit use and 64 bit use. It is important to match the bitness of the files with the bitness of the python environment used.

Solartron

Assuming the `pyvisa` library is installed, the only dependency talking to the solartron unit has software wise the *NI-VISA* backend, which can be installed from here. Once installed, the download from the link should present a package manager. From this install the *NI-VISA* driver. The *NI-488.2* driver was also found to be useful, but perhaps optional from system to system (untested).

Biologic

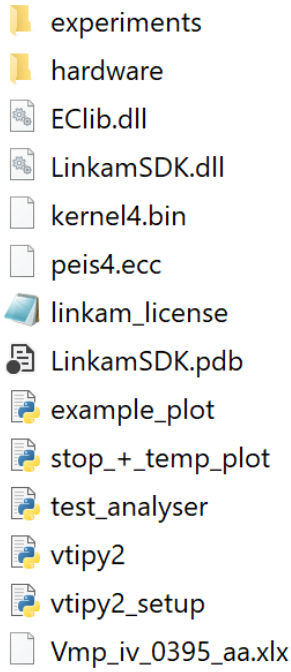
To successfully use the potentio electrochemical impedance spectroscopy technique with the *Biologic* unit, the below files must be in the working directory of *vtipy2*:

- EClib.dll (32 bit only)
- kernel4.bin
- Vmp_iv_0395_aa.xlx
- peis4.ecc

If a 64 bit python environment is necessary, *Biologic* also supply a file names *EClib64.dll* which should be used in place of *EClib.dll*. This however is untested with python by *Biologic* and has been shown not to work in this setup, though there is a chance it could work with persistence.

Complete Setup in Directory

A complete directory setup for full use of *vtipy2* with use of all the hardware is shown below. This assumes a 32 bit python environment with all of the external libraries above installed, and that the *NI-VISA8 backend has been successfully installed.



Note that the *experiments* directory is not sctrictly required, however it comes with some example data that can be used to test `example_plot.py`.

Most dependencies are not co-dependent, so files related to only one piece of equipment can be safely omitted if that certain piece of equipment is not to be used.

Issues and Future Functionality

- **IMPORTANT**: There is no capability to ensure that the connected stage is the stage chosen in the software by the user. While in theory the Linkam SDK shouldn't allow for a stage to overheat due to this error, for example, it is untested in this setup. An attempt was made to create this capability, but unfortunately I ran out of time on the internship. Some notes on how this may be fixed can be found in `vtipy2_setup.py`. If/when implemented, this functionality should be tested rigourously.
- Some *Biologic* functionality untested, rigorous testing needed.
- *Biologic* development package displays a "confirm?" window when passing parameters that must be manually clicked away by a user. This may need communication with *Biologic* to fix.
- The calculated estimated time assumes a cooling pump is in use, however if this isn't the case, the estimated time can be out (on the order of minutes). This is especially prevalent at room temperatures. To fix this, perhaps an exponential cooling rate could be measured and implemented in the estimated cooling time.
- There is no functionality for the detection of a connected cooling pump. In theory, the Linkam controller should detect and use it automatically but *vtipy2* has no way of doing this. Implementing the cooling pump into the physical setup is to be carried out after the internship.
- On initialisation of the USB connection to the Linkam controller, 'product/vendor' is output to the terminal. This is mainly an aesthetic issue, however purely 'blocking output' may lead to errors going unreported.