Potentiostat Communication and Data Processing

Kurt Manrique-Nino, Tompkins Cortland Community College; Dr. Peter Ko, CHESS
CLASSE, Cornell University, Ithaca NY 14853, U.S.A.

Abstract

This report describes the process performed to establish a communication link between a potentiostat and a computer running the scientific software SPEC, allowing both instruments to run their respective experiments in parallel. Also, this document describes the development of seven data processing scripts using the Python to analyze and reduce the data generated by electrochemical and X-ray diffraction experiments at the Cornell High Energy Synchrotron Source (CHESS).

Introduction

The Cornell High Energy Synchrotron Source (CHESS) uses a potentiostat in conjunction with SPEC software to perform operando or in situ X-ray experiments on energy materials. The **potentiostat** is an instrument used to perform electrochemical experiments such as, studying the performance of batteries, or the catalytic properties of materials in fuel cells. **SPEC** is a UNIX-based software package used for instrument control and data acquisition used in X-ray experiments (Certified Scientific Software).

Since both instruments typically run in two separate instances, users would operate them in two separate interfaces. It would be convenient if both instruments can communicate by sending signals to each other every time an event occurs during the experiments. The intended interaction between the potentiostat and SPEC is shown in Figure 1. This will allow both systems to operate simultaneously and to facilitate the processing of data generated by both electrochemical and X-ray diffraction experiments.

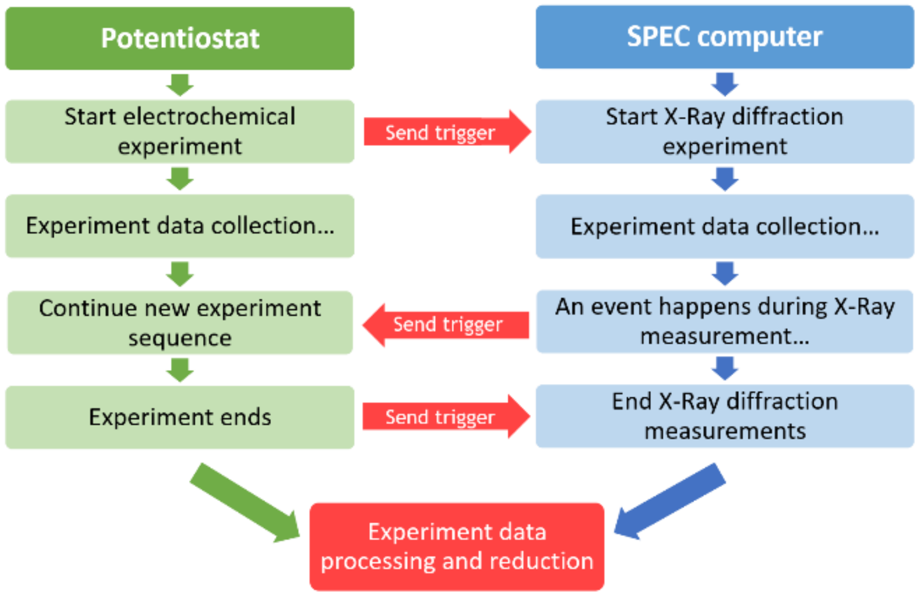

Potentiostat
SPEC computer
Start electrochemical experiment
Send trigger
Start X-Ray diffraction experiment
Experiment data collection...
Experiment data collection...
Continue new experiment sequence
Send trigger
An event happens during X-Ray measurement...
Experiment ends
Send trigger
End X-Ray diffraction measurements
Experiment data processing and reduction

Figure 1: Flowchart showing the intended behavior between the potentiostat and SPEC software.

After both, electrochemical and X-ray diffraction experiments, are complete, it is necessary to process the experimental data. Only small portions of the raw X-ray data contain useful information, and therefore must be ‘reduced’.

This project had three main goals to address:

- Creating a send/receive signal communication link between the potentiostat and SPEC.
- Develop (user-friendly) data processing scripts to analyze and reduce data generated from electrochemical and X-ray experiments using Python.
- Produce documentation for experimental hardware/software setup, and data processing scripts.

Potentiostat-SPEC Communication

Potentiostat overview

The potentiostat used in this project is a BioLogic SP-200, and it can be operated through a Windows-based software called EC-Lab. This software allows the operator to run different experiment techniques, applications and to send/receive signals to an external instrument. The current setup interacts with a computer running SPEC, a UNIX-based software used for “instrument control and data acquisition for X-ray diffraction” (Certified Scientific Software).

The potentiostat uses five cables to gather the electrochemical experiment data. The red cables are for the working electrodes, the blue cables are the counter electrodes and the white cable is the for reference electrode. The red and blue cables are labelled either ‘P’ or ‘S’, where they are used for the control/measurement of current and potential respectively. There is a black cable which is the ground terminal. This cable is not often used. The electrodes are connected to the potentiostat through a serial port. This cable must not be connected or disconnected at any time while the potentiostat is powered.

Hardware setup

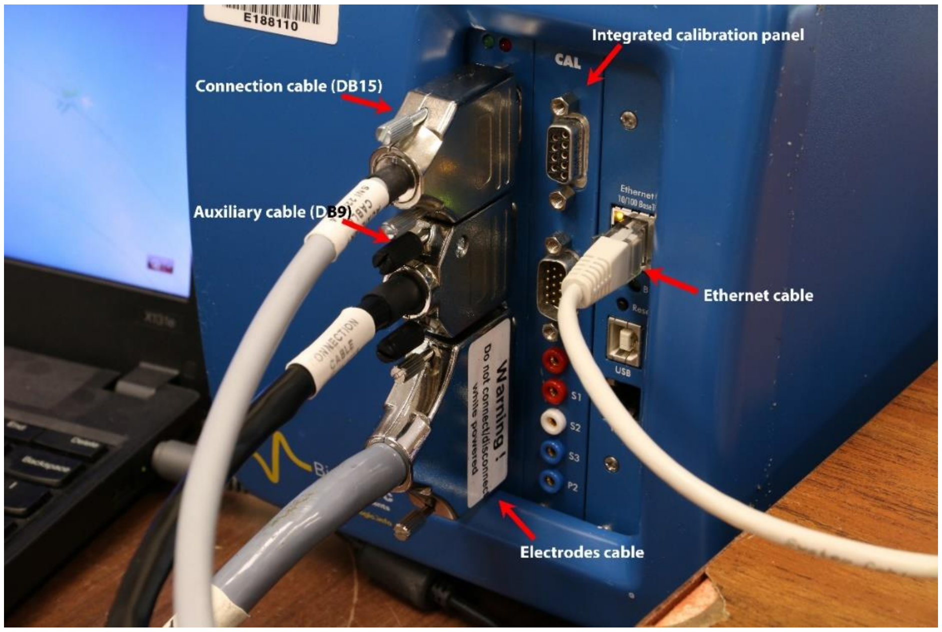

Integrated calibration panel
Connection cable (DB15)
Auxiliary cable (DB9)
Ethernet cable
Electrodes cable
Warning ! Do not connect/disconnect while powered

Figure 2: BioLogic SP-200 Potentiostat (labeled connections)

This setup consists of a BioLogic SP200 potentiostat, two serial cables (connection cable DB15 and auxiliary cable DB9), a BioLogic analog filter, three BNC cables, two BNC jumper cables, a voltage to frequency converter, a BNC patch panel connected to a National Instrument Counter Card installed within the SPEC computer.

The procedure to connect the potentiostat and the SPEC computer is the following:

1. Connect the connection (DB15) and auxiliary (DB9) cables from the potentiostat (see Figure 2) to the analog filter on their respective ports. The DB15 cable will power to the analog filter and the DB9 will transmit data from the potentiostat to the analog filter and vice-versa.
2. Connect one BNC cable to the "Trigger IN" port of the analog filter (see Figure 3); connect the other end to and available DIO port (digital input/output port) of the BNC patch panel.
3. Connect one BNC cable to the "Trigger OUT" port of the analog filter (See Figure 3); connect the other end to the voltage input port (In – V:F) of an available voltage to frequency converter.

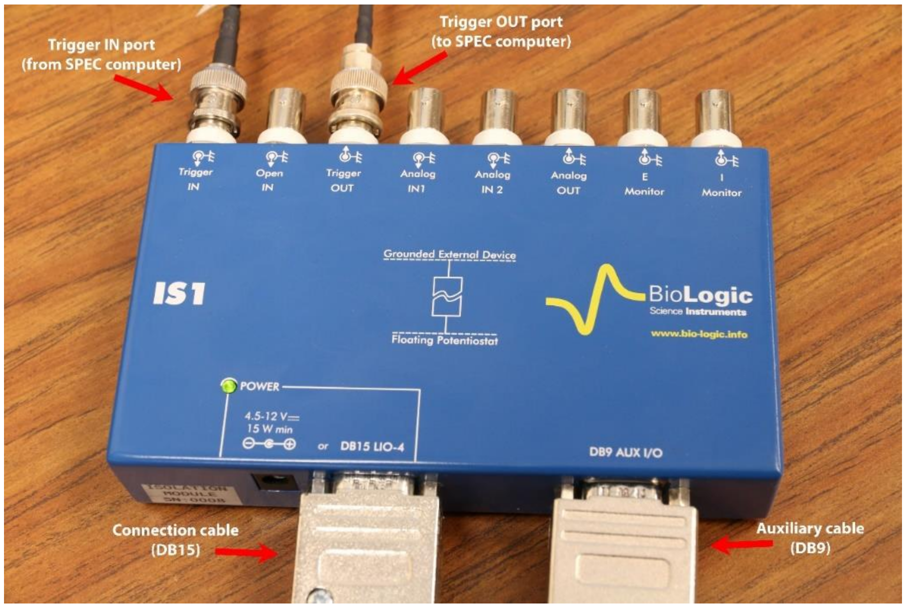


Figure 3: BioLogic Analog Filter (labeled connections)

4. On the voltage to frequency converter, connect one jumper cable to the frequency output port (Out – V:F) and connect the other end to the frequency input port (In – F:F).
5. Connect one BNC cable to the frequency output port (Out – F:F) of the voltage to frequency converter, and connect the other end to an available counter port of the BNC patch panel. NOTE: the selected counter port must be specified on the SPEC software for proper linkage.
6. Connect the last jumper cable to the OUTPUT port of the BNC patch panel. Then, connect the other end of the cable to the GATE port the BNC patch panel.

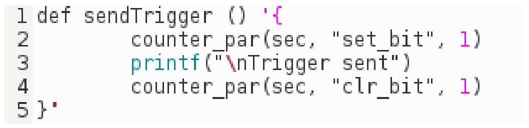
<u>Software setup</u>

The EC-Lab software allows the operator to control the potentiostat by executing a variety of techniques and applications to run electrochemical experiments. The user could create a sequence of electrochemical experiments through this interface and set the order in which they will be executed. This action can be performed through the parameters settings window, located on the left side of the window. In order for the potentiostat to receive a signal from the SPEC computer, it must use the

Trigger in – TI technique. The Trigger out – TO technique is used to send a signal from the potentiostat to the SPEC computer. The output signal sent to SPEC must be converted from voltage to frequency and then send to the SPEC computer through the BNC patch panel.

The SPEC software is used to send/receive signals and process the data generated by the potentiostat. The SPEC language uses a syntax similar to the C language and its user interface behaves like a BASIC language interpreter (Certified Scientific Software). In order to receive and send signals between both systems, two macros must be defined in SPEC:

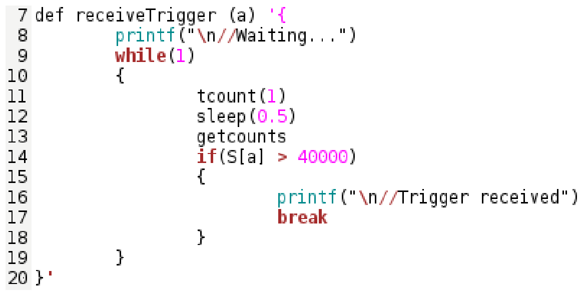
Send trigger macro sends a signal to the potentiostat from SPEC. In this example, DIO1 port with a counter named "sec" was used. Line 2 will set the DIO1 port on. If a different DIO port was used, the user should update this line accordingly. Line 3 prints on the SPEC interface that the trigger was sent. And finally, line 4 will turn the DIO1 port off. Again, if a different DIO port was used, the user should update this line accordingly.



```
1 def sendTrigger () '{
2         counter_par(sec, "set_bit", 1)
3         printf("\nTrigger sent")
4         counter_par(sec, "clr_bit", 1)
5 }'
```

Figure 4: sendTrigger() SPEC macro

Receive trigger macro receives a trigger from the potentiostat. This macro needs to enter one parameter. The parameter entered must be the address or name of the port of the BNC patch panel, where the signal will be received. Line 8 prints a message that SPEC is waiting for a signal from the potentiostat. In line 11 there is a *tcount(t)* macro, which starts the timer counting for t seconds. Then line 12 will execute a *sleep(t)* function that will stop the counter for *t* seconds. Line 9 through 19 will execute an infinite loop, where it constantly checks for a condition. This condition is stated in line 14, if the count exceeds 40000 (which corresponds to approximately 4 volts), it will print a message on the SPEC interface and stop the loop.



```
 7 def receiveTrigger (a) '{
 8         printf("\n//Waiting...")
 9         while(1)
10         {
11                 tcount(1)
12                 sleep(0.5)
13                 getcounts
14                 if(S[a] > 40000)
15                 {
16                         printf("\n//Trigger received")
17                         break
18                 }
19         }
20 }'
```

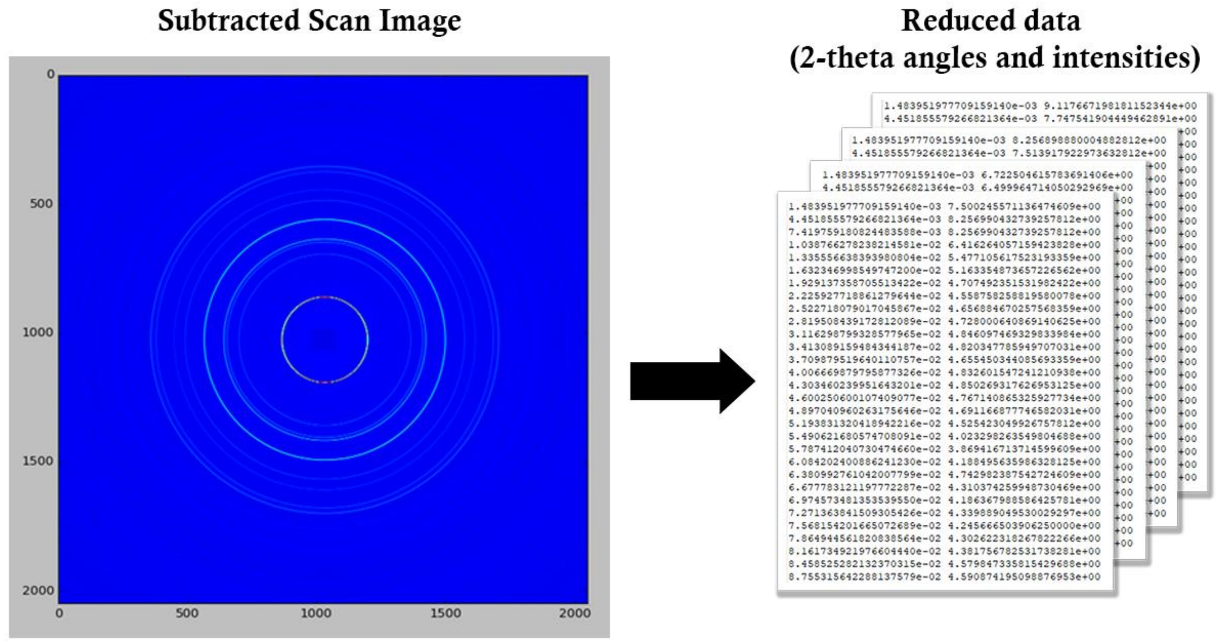
Figure 5: receiveTrigger(input) SPEC macro.

Data Processing

After the electrochemical and X-ray diffraction experiments are finished, the data collected is stored in the CLASSE central server. The data generated by both experiments are saved independently from each other on separate locations, which makes the data management challenging. As previously mentioned, the X-ray data contain information that is not useful or relevant. To address this issue, we developed three types of data processing scripts to analyze the raw data generated by the electrochemical and X-ray diffraction experiments:

1. Image subtraction and data reduction: subtracts the background image from each experimental raw frame, averages the each subtracted set of frames, and then reduces the data by performing Azimuthal Integration on each image. Four scripts belong to this group:

- **image_subtraction.py:** subtracts the background image from all scan image frames, calculates an average for each individual scan, and then saves the subtracted images in a directory.
- **single_image_subtraction.py:** performs the image subtraction and average for one single scan.
- **image_data_reduction.py:** it accepts a directory containing the subtracted images and outputs the reduced data in 2θ angles of diffraction or Q-Space, and X-ray intensity.
- **image_subtraction_data_reduction.py:** performs the image subtraction and data reduction.



```
1.483951977709159140e-03 9.117667198181152344e+00
4.451855579266821364e-03 7.747541904449462891e+00
```

```
1.483951977709159140e-03 8.256898880004882812e+00
4.451855579266821364e-03 7.513917922973632812e+00
```

```
1.483951977709159140e-03 6.722504615783691406e+00
4.451855579266821364e-03 6.499964714050292969e+00
```

```
1.483951977709159140e-03 7.500245571136474609e+00
4.451855579266821364e-03 8.256990432739257812e+00
7.419759180824483588e-03 8.256990432739257812e+00
1.038766278238214581e-02 6.416264057159423828e+00
1.335556638393980804e-02 5.477105617523193359e+00
1.632346998549747200e-02 5.163354873657226562e+00
1.929137358705513422e-02 4.707492351531982422e+00
2.225927718861279644e-02 4.558758258819580078e+00
2.522718079017045867e-02 4.656884670257568359e+00
2.819508439172812089e-02 4.728000640869140625e+00
3.116298799328577965e-02 4.846097469329833984e+00
3.413089159484344187e-02 4.820347785949707031e+00
3.709879519640110757e-02 4.655450344085693359e+00
4.006669879795877326e-02 4.832601547241210938e+00
4.303460239951643201e-02 4.850269317626953125e+00
4.600250600107409077e-02 4.767140865325927734e+00
4.897040960263175646e-02 4.691166877746582031e+00
5.193831320418942216e-02 4.525423049926757812e+00
5.490621680574708091e-02 4.023298263549804688e+00
5.787412040730474660e-02 3.869416713714599609e+00
6.084202400886241230e-02 4.188495635986328125e+00
6.380992761042007799e-02 4.742982387542724609e+00
6.677783121197772287e-02 4.310374259948730469e+00
6.974573481353539550e-02 4.186367988586425781e+00
7.271363841509305426e-02 4.339889049530029297e+00
7.568154201665072689e-02 4.245666503906250000e+00
7.864944561820838564e-02 4.302622318267822266e+00
8.161734921976604440e-02 4.381756782531738281e+00
8.458525282132370315e-02 4.579847335815429688e+00
8.755315642288137579e-02 4.590874195098876953e+00
```

Figure 6: image_subtraction_data_reduction.py script product.

2. Contour map and plot generation: takes the reduced data from the subtracted images and generates a plot of temperature or potential vs. time for the electrochemical experiments, and a contour graph showing the intensity variation for the X-ray diffraction experiments. Two scripts belong to this group:

- **potential_v_time.py:** generates a plot (potential vs. time), a contour map (angle of diffraction, time and X-ray intensities), and a zoomed-in version of the contour map based on the data generated by electrochemical and X-ray experiments.
- **temperature_v_time.py:** generates a plot (temperature vs. time), a contour map (angle of diffraction, time and X-ray intensities), and a zoomed-in version of the contour map based on the data generated by electrochemical and X-ray experiments.

3. Scan file parsing: a supplementary script that parses a SPEC log file, extracts the experimental data, and saves the information into a separate text files for each scan within a specific directory.

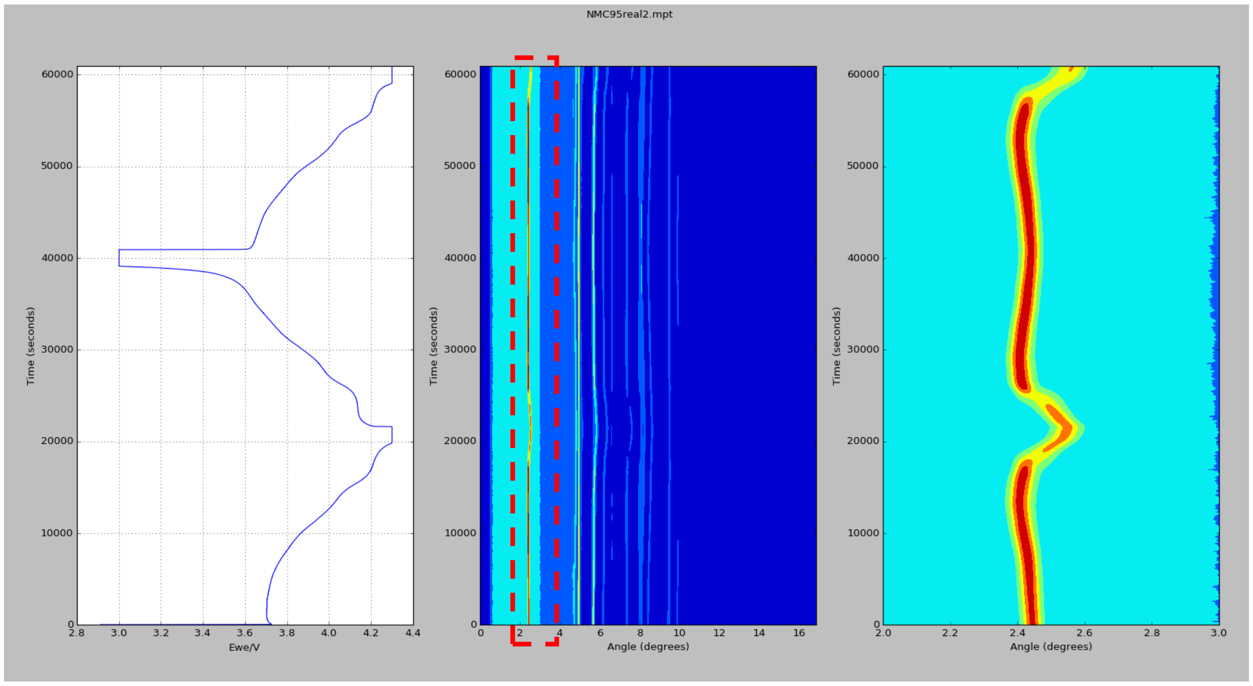


Figure 7: From left to right; plot (potential v. time), contour map (angle v. time), and zoomed-in contour map (angle v. time)

Summary

By the end of the project, we successfully established a communication link between the potentiostat and the SPEC computer. This will allow both instruments to operate on the same instance. Both electrochemical and X-ray diffraction experiments when running will have the same time stamps, which will ease the experiment's data processing phase.

For the data processing, we tested each script with different experiment samples to verify that it operated as expected. They are available in Python 2 and Python 3, and can be executed on any operating system, as long as there is an instance of the Python interpreter installed in the computer. We discovered that a computer with at least 16 gigabytes of RAM is required to run the scripts *image_subtraction.py* and *image_subtraction_data_reduction.py*, if the user plans to process a large set of experimental data. The time it takes to process the data depends on the CPU performance, network traffic if the experimental data is stored on a server, and/or disk type; where solid state drives are significantly faster than regular hard disk drives.

Acknowledgements

I would like to thank my mentor Dr. Peter Ko for his help, guidance, and knowledge on this project; and the great opportunity to participate in the Summer Research for Community College Students program 2018 at Cornell University. This work is based upon research conducted at the Cornell High Energy Synchrotron Source (CHESS) which is supported by the National Science Foundation under the award DMR-1332208.

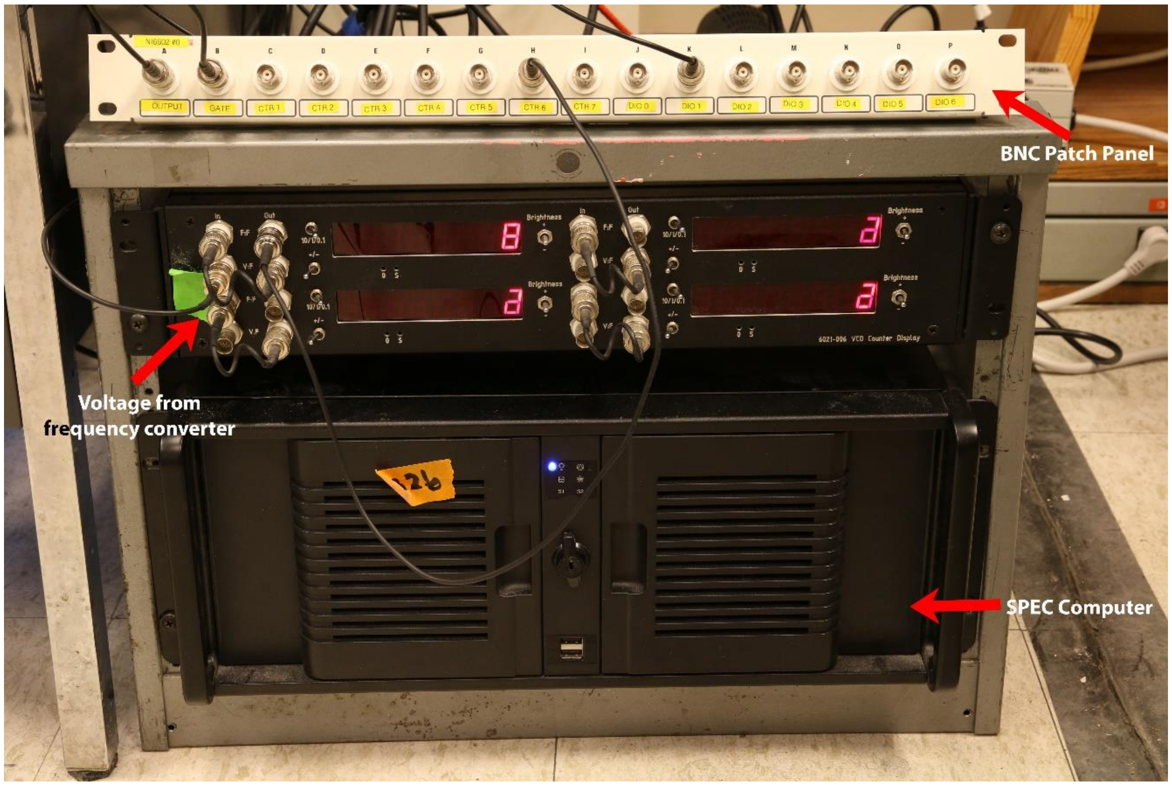
References

1. Certified Scientific Software. “SPEC.” SPEC Manual, www.certif.com/spec_manual/idx.html.

2. BioLogic. “SP2xx / SP-300.” Modular Potentiostat/Galvanostat, 2015, www.bio-logic.net/wp-content/uploads/2015-bio-logic-sp2xx-300.pdf.

3. BioLogic. “EC-Lab.” EC-Lab/BT-Lab Software User's Manual, v11.16, Jan. 2017.

Appendix



Appendix Figure 1: SPEC computer (labeled connections)

Source code: image_subtraction_data_reduction.py

```
###########################################
#    image_subtraction_data_reduction.py
#    Python 2.x
#    Kurt Manrique-Nino
###########################################

import numpy as np
import os
import gc
import matplotlib.pyplot as plt

import pyFAI
import pyFAI.calibrant
from pyFAI.calibrant import ALL_CALIBRANTS
import fabio
from pyFAI.multi_geometry import MultiGeometry

def generate_default_output(root_path):
    default_output_path = '/nfs/chess/aux/cycles/'
    split_path = root_path.split('/')
    for index, subdir in enumerate(split_path):
        if(subdir == 'raw'):
            split_path = split_path[index + 1:-1]
            break
    for i in split_path:
        default_output_path += i + '/'
```

```python
    return default_output_path

def get_subdirs(root):
    for src, dirs, files in os.walk(root):
        return dirs

def build_raw_subdirs(subdirs_list, darkfield_file_path, start_from_scan, darkfield_file_within, darkfield_scan_number):
    darkfield_scan_arr = darkfield_file_path.split('/')
    darkfield_scan_number = int(darkfield_scan_arr[len(darkfield_scan_arr)-3])
    start_from_scan = str(start_from_scan)
    sort_subdirs = []
    if(darkfield_file_within == 'y'):
        for sd in subdirs_list:
            try:
                temp = int(sd)
                if(temp != darkfield_scan_number):
                    sort_subdirs.append(temp)
            except:
                pass
    else:
        for sd in subdirs_list:
            try:
                temp = int(sd)
                sort_subdirs.append(temp)
            except:
                pass
    sort_subdirs.sort()
    sort_subdirs = map(str, sort_subdirs)
    for scan in range(0, len(sort_subdirs)):
        if(sort_subdirs[scan] == start_from_scan):
            sort_subdirs = sort_subdirs[scan:]
            break
    return sort_subdirs

def build_scan_file_paths(path, scan_subdirs):
    scan_paths = []
    ge2_file_paths = []
    for sd in scan_subdirs:
        temp = path + sd + '/ff'
        scan_paths.append(temp)
    for path in scan_paths:
        for src, dirs, files in os.walk(path):
            for file in files:
                if(file.endswith('.ge2')):
                    ge2_file_paths.append(os.path.join(src, file))

    return ge2_file_paths

def build_darkfield_file(darkfield_path):
    skip_lines = 4096
    darkfield_array = np.fromfile(darkfield_path, dtype = 'int16', sep = '')
    darkfield_array = darkfield_array[skip_lines:skip_lines + 2048*2048]
    return (darkfield_array)

def image_substraction(ge2_file_paths, darkfield_file):
    skip_lines = 4096
    frame_resolution = 2048 * 2048
    averaged_subtracted_array = []
    scan_counter = 0
    for scan in ge2_file_paths:
        sliced_frames = []
        subtracted_frames = []
        scan_file = np.fromfile(scan, dtype = 'int16', sep = '')
        scan_file = scan_file[skip_lines:]
        sliced_frames = [scan_file[i:i + frame_resolution] for i in xrange(0, len(scan_file), frame_resolution)]
        print 'Reading scan: ', ge2_file_paths[scan_counter]
        scan_counter += 1
        for frame in sliced_frames:
            subtracted_frames.append(np.subtract(frame, darkfield_file))
        average_frames = np.average(subtracted_frames, axis = 0)
        average_frames = average_frames.astype(int)
        average_frames = average_frames.reshape(2048, 2048)
        averaged_subtracted_array.append(average_frames)
            del scan_file, sliced_frames[:],  subtracted_frames[:]
            gc.collect()
        return averaged_subtracted_array

    def display_single_scan(ge2_file_paths, subtract_scans, display_scan):
        display_scan = display_scan.lower()
        if(display_scan != 'n' or display_scan != '0'):
            for index, path in enumerate(ge2_file_paths):
                split_path = path.split('/')
                scan_number = split_path[len(split_path)-3]
                if(scan_number == display_scan):
                    plt.imshow(subtract_scans[index])
                    plt.show()
                    break
        else:
            print('No scan to display.')

    def save_processed_scans(destination_dir_path, scan_subdirs, subtract_scans, ge2_file_paths, ai, overwrite_files, file_type):
        overwrite_files = overwrite_files.lower()
        dest_subdirs_array = []
        for subdir in scan_subdirs:
            dest_subdir = destination_dir_path + '/reduced/' + str(subdir)
            dest_subdirs_array.append(dest_subdir)
            try:
                os.makedirs(dest_subdir)
            except:
                pass
        for i in range(0, len(subtract_scans)):
            img_diff = subtract_scans[i]
            if(file_type == 1):
                if(overwrite_files == 'y' or overwrite_files == '1'):
                    print 'Overwriting reduced .chi file from: ', ge2_file_paths[i]

                    tth_file_path = dest_subdirs_array[i] + '/' + ge2_file_paths[i][-13:-4] + '.chi'
                    try:
                        os.remove(tth_file_path)
                    except:
                        pass
                    tth_data_x, tth_data_y = ai.integrate1d(img_diff, data_points, unit = '2th_deg')
                    tth_combined_xy = np.vstack((tth_data_x, tth_data_y)).T
                    np.savetxt(tth_file_path, tth_combined_xy)
                else:
                    tth_file_path = dest_subdirs_array[i] + '/' + ge2_file_paths[i][-13:-4] + '.chi'
```

```
                        if(os.path.exists(tth_file_path)):
                            pass
                        else:
                            print 'Reducing to .chi file from: ', ge2_file_paths[i]
                            tth_data_x, tth_data_y = ai.integrate1d(img_diff, data_points, unit = '2th_deg')
                            tth_combined_xy = np.vstack((tth_data_x, tth_data_y)).T
                            np.savetxt(tth_file_path, tth_combined_xy)
                elif(file_type == 2):
                    if(overwrite_files == 'y' or overwrite_files == '1'):
                        print 'Overwriting reduced .qda file from: ', ge2_file_paths[i]

                        qda_file_path = dest_subdirs_array[i] + '/' + ge2_file_paths[i][-13:-4] + '.qda'
                        try:
                            os.remove(qda_file_path)
                        except:
                            pass
                        qda_data_x, qda_data_y = ai.integrate1d(img_diff, data_points, unit = 'q_A^-1')
                        qda_combined_xy = np.vstack((qda_data_x, qda_data_y)).T
                        np.savetxt(qda_file_path, qda_combined_xy)
                    else:
                        qda_file_path = dest_subdirs_array[i] + '/' + ge2_file_paths[i][-13:-4] + '.qda'
                        if(os.path.exists(qda_file_path)):
                            pass
                        else:
                            print 'Reducing to .qda file from: ', ge2_file_paths[i]
                            qda_data_x, qda_data_y = ai.integrate1d(img_diff, data_points, unit = 'q_A^-1')
                            qda_combined_xy = np.vstack((qda_data_x, qda_data_y)).T
                            np.savetxt(qda_file_path, qda_combined_xy)
                elif(file_type == 3):
                    if(overwrite_files == 'y' or overwrite_files == '1'):
                        print 'Overwriting reduced .chi file from: ', ge2_file_paths[i]

                        tth_file_path = dest_subdirs_array[i] + '/' + ge2_file_paths[i][-13:-4] + '.chi'
                        try:
                            os.remove(tth_file_path)
                        except:
                            pass
                        tth_data_x, tth_data_y = ai.integrate1d(img_diff, data_points, unit = '2th_deg')
                        tth_combined_xy = np.vstack((tth_data_x, tth_data_y)).T
                        np.savetxt(tth_file_path, tth_combined_xy)
                        print 'Overwriting reduced .qda file from: ', ge2_file_paths[i]

                        qda_file_path = dest_subdirs_array[i] + '/' + ge2_file_paths[i][-13:-4] + '.qda'
                        try:
                            os.remove(qda_file_path)
                        except:
                            pass
                        qda_data_x, qda_data_y = ai.integrate1d(img_diff, data_points, unit = 'q_A^-1')
                        qda_combined_xy = np.vstack((qda_data_x, qda_data_y)).T
                        np.savetxt(qda_file_path, qda_combined_xy)
                    else:
```

```python
                    tth_file_path = dest_subdirs_array[i] + '/' + ge2_file_paths[i][-13:-4] + '.chi'
                    qda_file_path = dest_subdirs_array[i] + '/' + ge2_file_paths[i][-13:-4] + '.qda'
                    if(os.path.exists(tth_file_path)):
                        pass
                    else:
                        print 'Reducing to .chi file from: ', ge2_file_paths[i]
                        tth_data_x, tth_data_y = ai.integrate1d(img_diff, data_points, unit = '2th_deg')
                        tth_combined_xy = np.vstack((tth_data_x, tth_data_y)).T
                        np.savetxt(tth_file_path, tth_combined_xy)
                    if(os.path.exists(qda_file_path)):
                        pass
                    else:
                        print 'Reducing to .qda file from: ', ge2_file_paths[i]
                        qda_data_x, qda_data_y = ai.integrate1d(img_diff, data_points, unit = 'q_A^-1')
                        qda_combined_xy = np.vstack((qda_data_x, qda_data_y)).T
                        np.savetxt(qda_file_path, qda_combined_xy)
            else:
                print('Could not create the reduced data files.')

    #USER INPUT
    print('========== Directories and Paths ==========')
    root_dir_path = raw_input('Enter the path to the raw data directory: ')
    darkfield_file_within = raw_input('Is the darkfield file within the raw data directory (y/n): ')
    darkfield_file_path = ''
    darkfield_scan_number = 0
    if(darkfield_file_within == 'y' or darkfield_file_within == '1'):
        darkfield_scan_number = int(input('Enter the dark field scan number: '))
        for src, dirs, files in os.walk(root_dir_path + str(darkfield_scan_number) + '/ff/'):
            for file in files:
                if(file.endswith('.ge2')):
                    darkfield_file_path += os.path.join(src, file)
    else:
        darkfield_file_path += raw_input('Enter the path to the darkfield file: ')
    start_scan_from = int(input('Start from scan number (type 0 to read all scans): '))
    destination_dir_path = ''
    default_or_other = raw_input('Save processed files in defualt directory? (y/n): ')
    default_or_other = default_or_other.lower()
    if(default_or_other == 'y' or default_or_other == '1'):
        destination_dir_path += generate_default_output(root_dir_path)
        print(destination_dir_path)
    else:
        destination_dir_path += raw_input('Enter the path to the destination directory: ')
    file_types = int(input('What files do you want to generate? (1:.chi / 2:.qda / 3:both): '))
    overwrite_files = raw_input('Do you want to overwrite the existing data reduction files? (y/n): ')
    display_scan = raw_input('Display a specific scan (type "n" to skip this step): ')
    print('========== Azimuthal Integration ==========')
    data_points = int(input('Enter the number of data points per scan: '))
    fit2d_x = float(input('Enter the horizontal pixel position for the main beam in pixel number: '))
```

```
        fit2d_y = float(input('Enter the vertical pixel position for the main beam in pixel number: '))
        fit2d_dist = float(input('Enter the distance from sample to detector in milimeters: '))
        energy = float(input('Enter the energy of the X-Ray beam in KeV: '))

        #STATIC DATA & CALCULATIONS
        fit2d_dist = fit2d_dist / 1000
        detector = pyFAI.detectors.Detector(200e-6, 200e-6)
        detector.max_shape = (2048, 2048)
        wl = 12.3984 / energy * 1e-10
        p1 = (2048 - fit2d_y) * 200e-6
        p2 = fit2d_x * 200e-6
        rot1deg = 0
        rot2deg = 0
        rot3deg = 0
        ai = pyFAI.AzimuthalIntegrator(wavelength = wl, dist = fit2d_dist, detector = detector, poni1 = p1, poni2 = p2, rot1 = np.radians(rot1deg), rot2 = np.radians(rot2deg), rot3 = np.radians(rot3deg))

        raw_subdirs_list = get_subdirs(root_dir_path)
        scan_subdirs = build_raw_subdirs(raw_subdirs_list, darkfield_file_path, start_scan_from, darkfield_file_within, darkfield_scan_number)
        ge2_file_paths = build_scan_file_paths(root_dir_path, scan_subdirs)
        darkfield_file = build_darkfield_file(darkfield_file_path)
        subtract_scans = image_substraction(ge2_file_paths, darkfield_file)
        save_processed_scans(destination_dir_path, scan_subdirs, subtract_scans, ge2_file_paths, ai, overwrite_files, file_types)
        display_single_scan(ge2_file_paths, subtract_scans, display_scan)

        del raw_subdirs_list, scan_subdirs, ge2_file_paths, darkfield_file, subtract_scans
        gc.collect()
```

Source code: potential_v_time.py

```
#################################
#    potential_v_time.py
#    Python 2.x
#    Kurt Manrique-Nino
#################################



import os
import numpy as np
import matplotlib.pyplot as plt
import sys
from datetime import datetime


def acquisition_time(path):
    acq_time = None
    with open(path) as data:
        counter = 0
        for line in data:
            counter += 1
            if(counter == 14):
                acq_time = datetime.strptime(((line[25:len(line) - 2])), "%m/%d/%Y %H:%M:%S")
```

```python
                break
    return acq_time

def range_mpt_file(path):
    with open(path) as mpt:
        mpt_start = 0
        for line, x in enumerate(mpt):
            if(line == 1):
                splitLine = x.split()
                mpt_start += int(splitLine[len(splitLine)-1])
                break
    return mpt_start

def getDeltaTime(time1, time2):
    delta_t = time2 - time1
    return delta_t.total_seconds()

def raw_spec_times(path, acq_time, trim):
    scan_time = []
    delta_times = []
    with open(path) as spec_log:
        skip = False
        for i in spec_log:
            for j in i.split():
                if(j == '#D'):
                    scan_time.append(datetime.strptime(i[7:len(i)-1], "%b %d %H:%M:%S %Y"))
    for scan in range(0, len(scan_time)):
            delta_times.append(getDeltaTime(acq_time, scan_time[scan]))
    trimmed_delta_times = delta_times[trim:]
    return trimmed_delta_times

def parse_first_scan_par_file(path):
    scan = 0
    with open(path) as par_file:
        counter = 0
        for line in par_file:
            counter += 1
            if(counter == 2):
                temp = line.split()
                scan = int(temp[7])
    return scan


def get_subdirs(root):
    for src, dirs, files in os.walk(root):
        return dirs

def build_chi_paths(subdirs, root, start_from_scan):
    sort_subdirs = map(int, subdirs)
    sort_subdirs.sort()
    subdirs.sort()
    root_dir = root
    paths = []
    full_paths = []
    if(root_dir[len(root) - 1] != '/'):
        root_dir += '/'
    else:
        pass
    for sub in range(0, len(subdirs) + 1):
        paths.append('%s%d%s'%(root_dir, sub, '/'))
    trimmed_paths = paths[start_from_scan:]
    for path in trimmed_paths:
        for src, dirs, files in os.walk(path):
            for file in files:
                if(file.endswith('.chi')):
                    temp = os.path.join(src, file)
                    full_paths.append(temp)
    return full_paths

def parse_angle_chi_files(arrayPaths):
    angle_arr = np.genfromtxt(arrayPaths[0], delimiter = ' ', usecols = 0)
    return angle_arr

def parse_intensity_chi_files(arrayPaths):
    collection_intensity = []
    for path in arrayPaths:
        intensity_arr = []
        intensity = np.genfromtxt(path, delimiter = ' ', usecols = 1)
        for i in range(0, intensity.size):
            collection_intensity.append(intensity[i])
    return collection_intensity

def display_graphs(angle_arr, time_arr, intensity_arr, path, theRange, zi, zf):

    #plot graph data
    time = np.genfromtxt(path, delimiter = '\t', skip_header = theRange, usecols = 7)
    potential = np.genfromtxt(path, delimiter = '\t', skip_header = theRange, usecols = 11)
    #controur graph data
    x = np.array(angle_arr)
    y = np.array(time_arr)
    x1, y1 = np.meshgrid(x, y)
    z = np.array(intensity_arr).reshape(x1.shape)
    #subplots
    f, (g1, g2, g3) = plt.subplots(1, 3)
    fileName = path.split('/')
    f.suptitle(fileName[len(fileName) - 1])
    #subplot plot
    g1.plot(potential, time, 'b')
    g1.set_xlabel('Ewe/V')
    g1.set_ylabel('Time (seconds)')
    g1.set_ylim([y[0], y[len(y) - 1]])
    g1.grid()
    #subplot contour
    g2.contourf(x1, y1, z)
    g2.set_xlabel('Angle (degrees)')
    g2.set_ylabel('Time (seconds)')
    #subplot zoom-in
    if(zi < angle_arr[0] or zi > zf or zf > angle_arr[len(angle_arr) - 1]):
        print('Invalid values to generate zoomed contour graph')
    else:
        g3.contourf(x1, y1, z)
        g3.set_xlim([zi, zf])
        g3.set_xlabel('Angle (degrees)')
        g3.set_ylabel('Time (seconds)')
    plt.show()

root_directory = raw_input('Enter the path to the root folder path: ')
spec_log_path = raw_input('Enter the path to the SPEC log file: ')
par_file_path = raw_input('Enter the path to the .par file: ')
```

```
        mpt_file_path = raw_input('Enter the path to the .mpt file: ')
        print('-----Zoom contour graph by angle (degrees)-----')
        zoom_contour_from = int(input('From: '))
        zoom_contour_to = int(input('To: '))

        start_from_scan = parse_first_scan_par_file(par_file_path)
        raw_subdirs = get_subdirs(root_directory)
        chi_paths = build_chi_paths(raw_subdirs, root_directory, start_from_scan)
        acq_time = acquisition_time(mpt_file_path)
        angle_array = parse_angle_chi_files(chi_paths)
        intensity_array = parse_intensity_chi_files(chi_paths)
        time_array = raw_spec_times(spec_log_path, acq_time, start_from_scan)
        range_mpt = range_mpt_file(mpt_file_path)
        display_graphs(angle_array, time_array, intensity_array, mpt_file_path, range_mpt, zoom_contour_from, zoom_contour_to)
```