# Report 2 - Spectrograph

April 12, 2019

## 1 Phys 581 Winter 2019

## 2 Report #2: Live spectrograph data

### 2.1 Alexander Hickey, 10169582

Note that the contents of this notebook were created and tested in a 64-bit distribution of Windows 10, using Python 3.6.8.

```
In [1]: import sys
        sys.version
```

```
Out[1]: '3.6.8 |Anaconda, Inc.| (default, Feb 21 2019, 18:30:04) [MSC v.1916 64 bit (AMD64)]'
```

```
In [2]: #Import useful libraries
        from IPython.display import Image
```

#### 2.1.1 Introduction

In assignment 6, we explored the construction of a graphical user interface for an Ocean Optics USB2000+ spectrograph. A simple GUI was developed using the Tkinter library in Python, and allowed the user to collect data from the spectrograph with the click of a button, and display this data on screen in a window. Additionally, the user was able to control the integration time of the spectrograph by manually entering the time in a text box. A full discussion on how to interface the spectrograph to a Windows 10 machine can be found in Assignment 6.

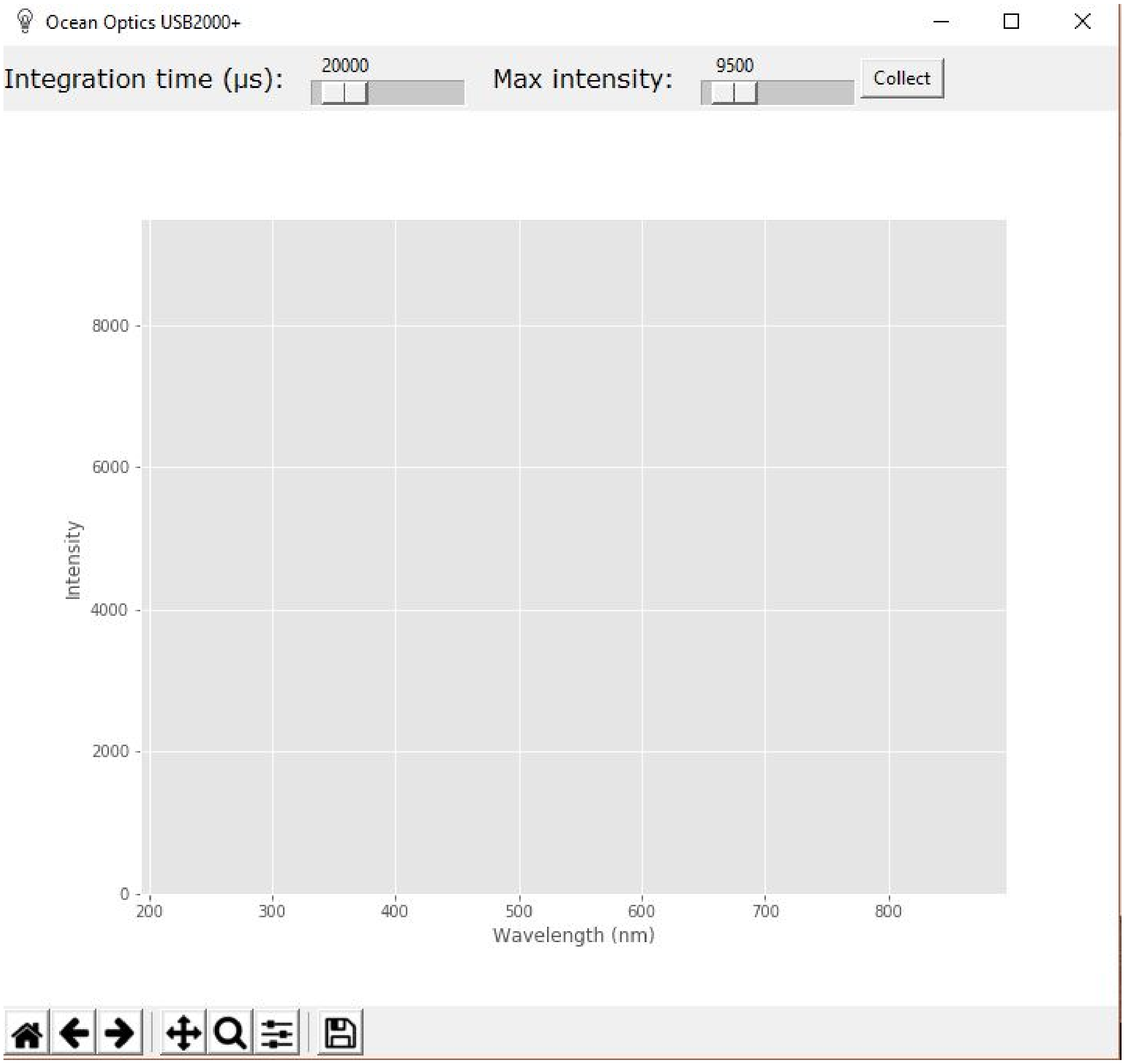
While the previous GUI was functional for simple visualization of the data collected by the spectrograph, the user was limited to displaying a single collection at the press of a button. This report presents improvements on the GUI, the most significant one being the implementation of live plotting the data collected by the spectroscope. Additionally, the integration time interface was changed from an entry box to a slider, which defaults to a reasonable setting, and prevents the user from entering an invalid value. Finally, a "max intensity" slider was added, which allows the user to manually adjust the range of the vertical axis on the plot.

#### 2.1.2 The GUI

This section includes several screenshots of the GUI, including the added sliders. These sliders were implemented using the Scale object in tkinter. The following image shows the GUI at startup, before any data has been collected.

```
In [3]: Image(filename="open.JPG",width = 400)
```

```
Out[3]:
```



Ocean Optics USB2000+

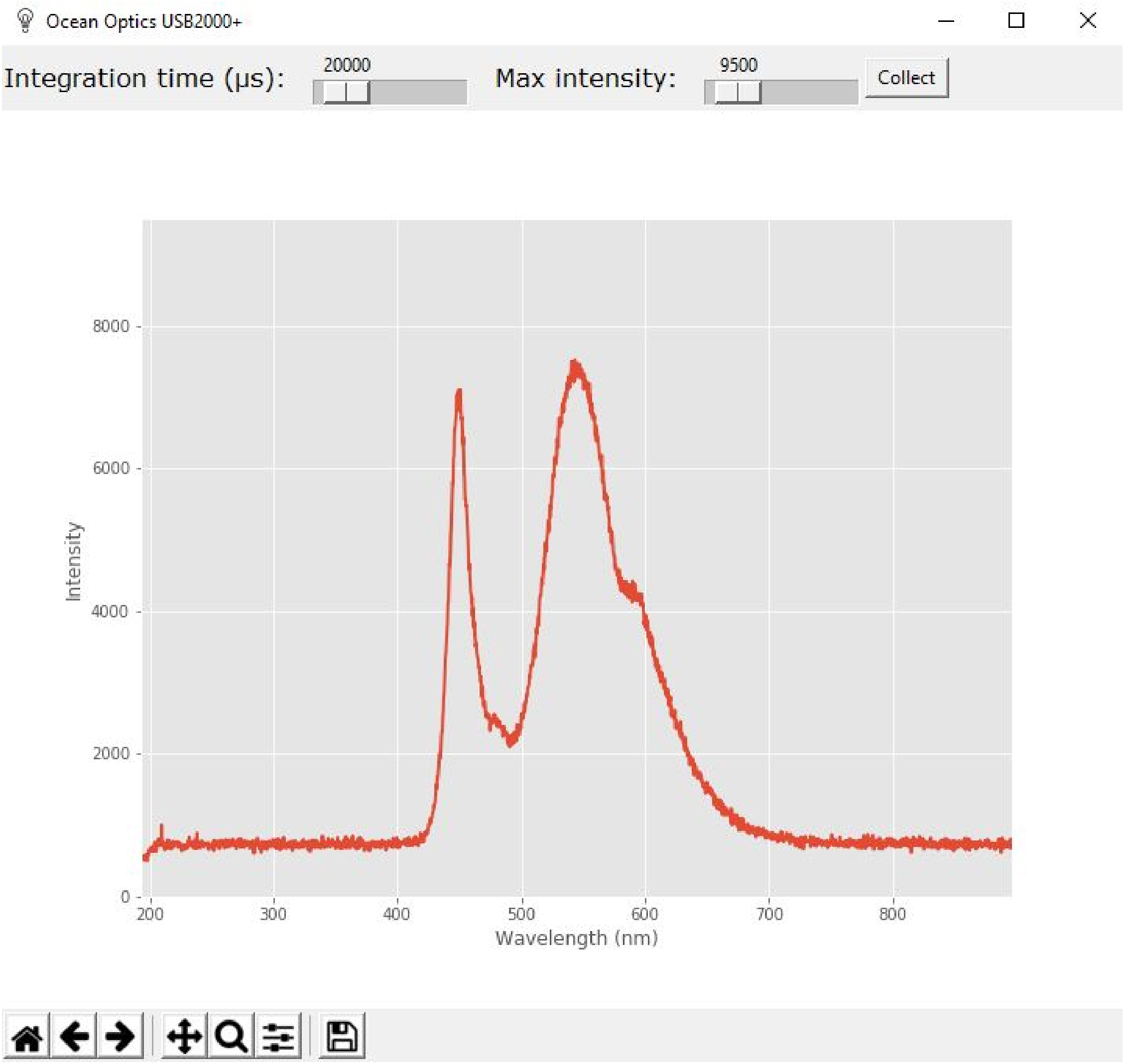
Integration time (µs): 20000

Max intensity: 9500

Upon pressing the "Collect" button, the program interfaces with the spectrograph and begins data collection at the desired integration time. The "Collect" button changes to read "Pause", and upon pressing this button, the plot freezes at the current frame, which can subsequently be manipulated and saved using the matplotlib toolbar. The following image shows the measured spectrum of my computer screen, at the default integration time.

```
In [4]: Image(filename="default.JPG",width = 400)
```

```
Out[4]:
```



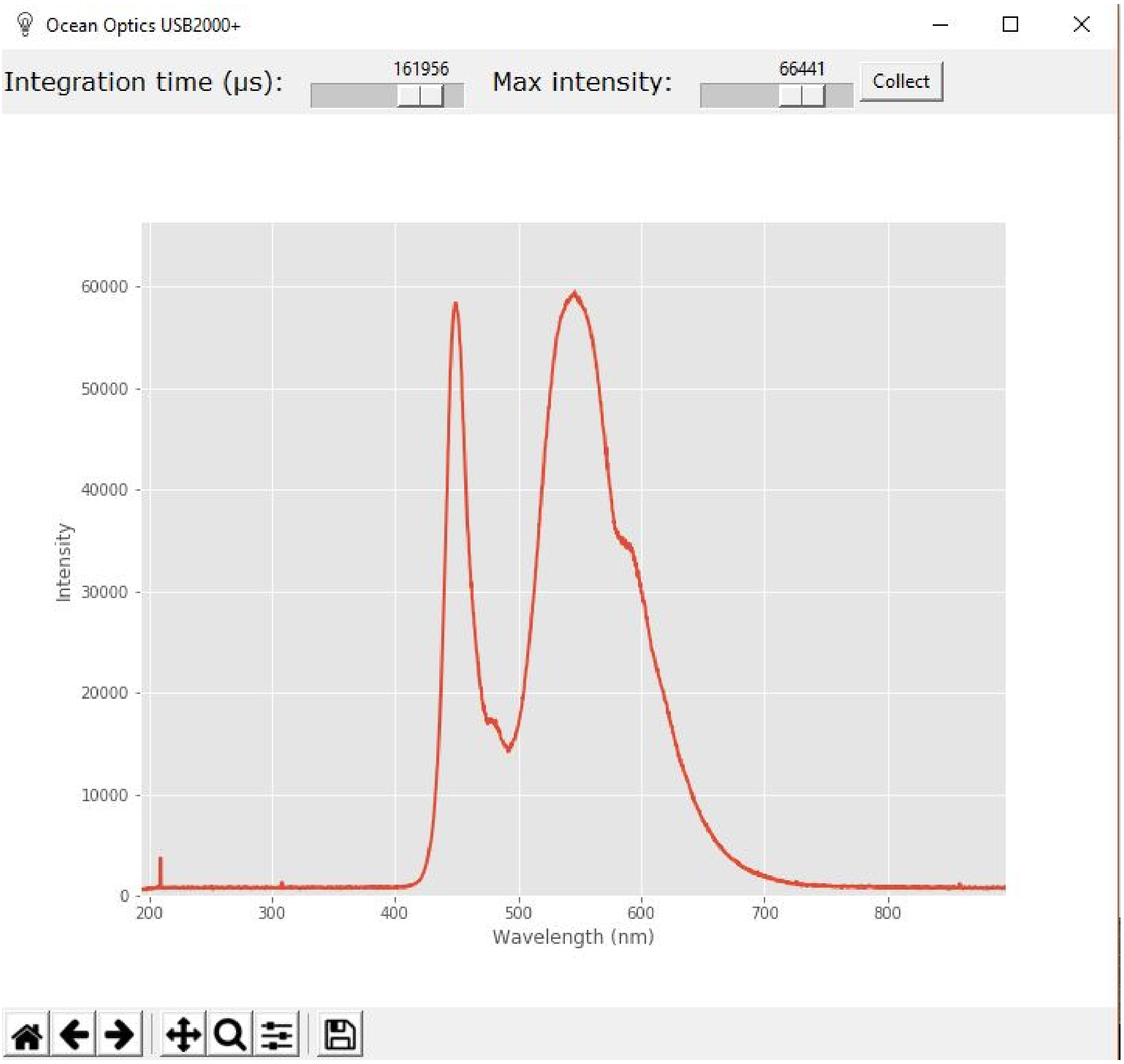
Integration time (μs): 20000

Max intensity: 9500

The following image shows the data collected, again from my computer screen, but for a much larger integration time. In this case, the intensity of the peaks increased greatly, so the vertical axis needed to be adjusted using the “Max intensity” slider.

```
In [5]: Image(filename="increased_integration.JPG",width = 400)
```

```
Out[5]:
```



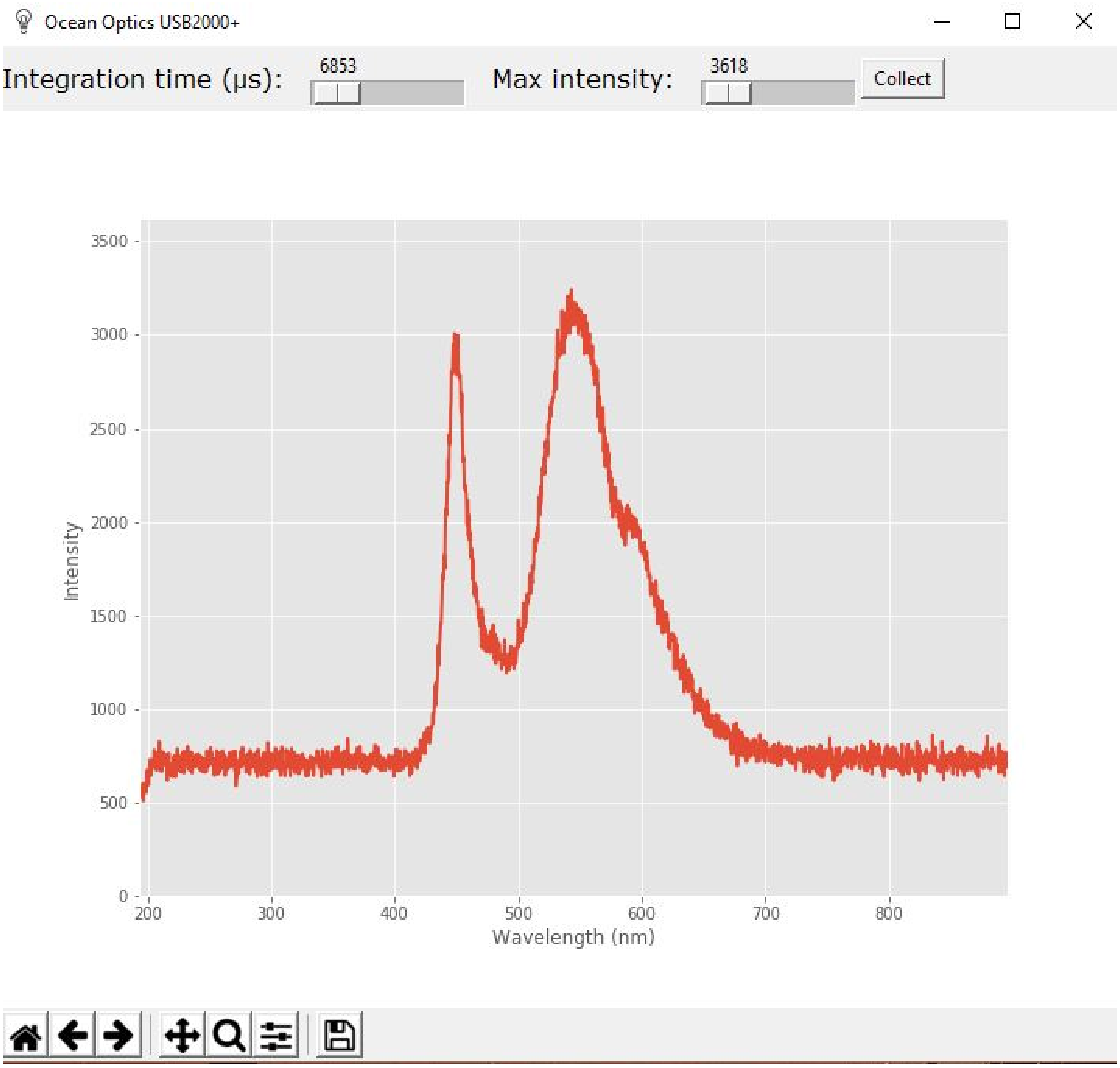
Integration time (µs): 161956

Max intensity: 66441

Finally, the next image shows the data, once again collected from my computer screen, but for a much lower integration time.

```
In [6]: Image(filename="decreased_integration.JPG",width = 400)
```

```
Out[6]:
```



Integration time (µs): 6853

Max intensity: 3618

It is worth noting that the two toolbars work during both the live plotting mode, and paused mode of the application. The full source code for this application can be found in the included SpectrographGUI_Live.py file.

### 2.1.3 Conclusion

This report presented improvements on the GUI developed in Assignment 6, used to ease the data collection upon interfacing with an Ocean Optics USB2000+ spectrograph. The improvements to this application include the live plotting functionality, as well as the implementation of sliders to control the spectrograph integration time, and the vertical scale of the plot.

With additional time, further functionality could be introduced, such as the ability to save the data file corresponding to the on screen plot while paused. It would also be interesting to spend some time improving the aesthetic of the application, to be more visually appealing to the user.

# Appendix: SpectrographGUI_Live.py

```
"""
@author: Alex Hickey

This program generates a graphical user interface for an Ocean Optics USB2000+
spectrograph. The interface includes a collection button, which will interface
with the spectrograph and plot the intensity versus wavelength and display
the output in real time. There is also a slider to manually set the integration
time, as well as the maximum intensity on the plot.
"""


#Import libraries
import tkinter as tk
from tkinter import ttk
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg, NavigationToolbar2Tk
from matplotlib import pyplot as plt
from matplotlib import style
import matplotlib.animation as animation
import seabreeze.spectrometers as sb

#Set font, plot style and collection range of spectrograph
FONT = ('Verdana', 12)
style.use('ggplot')
collec_range = (193,896) #Collection range in nanometers!
int_range = (0,9500) #Default intensity range




def get_data(integration_time = 20000):
    '''
    This function interfaces with the spectrograph and returns the
    intensity as a function of wavelength over a given integration time.

    Args:
        integration_time: Integration time in microsecons

    Return:
        wlength, intensity: arrays of wavelength and intensity of measurement

    '''

    #Enter loop to continuously update data
    while True:

        #Connect to spectrometer, set integration time
        spec = sb.Spectrometer.from_serial_number()
        spec.integration_time_micros(integration_time)

        #Retrieve wavelength and intensity arrays
        wlength = spec.wavelengths()[2:]
        intensity = spec.intensities()[2:]

        #Close connection
        spec.close()

        return wlength, intensity



class App(tk.Frame):
    ’’’
    Class corresponding to main application. Object is the homepage.
    ’’’

    def __init__(self, *args, **kwargs):

        #Initialize homepage
        tk.Frame.__init__(self, *args, **kwargs)
        cont = tk.Frame(self)
        cont.pack(side=’top’,fill=’both’)

        #Set application title
        self.winfo_toplevel().title(’Ocean Optics USB2000+’)

        #Attribute to track if animation is running
        self.running = False

        #Initialize animation attribute
        self.ani = None

        #Defines integration time label and slider
        lbl = ttk.Label(cont, text="Integration time ( s ):  ", font = FONT)
        lbl.pack(side=tk.LEFT)

        self.slide = tk.Scale(cont,orient=’horizontal’,from_= 1000, to= 200000)
        self.slide.set(20000)
        self.slide.pack(side=tk.LEFT)


        #Defines max intensity label and slider
        lbl2 = ttk.Label(cont, text="  Max intensity:  ", font = FONT)
        lbl2.pack(side=tk.LEFT)

        self.slide_max = tk.Scale(cont,orient=’horizontal’,from_= 1000,to_=90000)
        self.slide_max.set(int_range[1])
        self.slide_max.pack(side=tk.LEFT)

        #Defines data collection button
        self.btn = tk.Button(cont, text=’ Collect ’, command=self.on_click)
        self.btn.pack(side=tk.LEFT)

        #Defines the main plot
        self.fig = plt.Figure(figsize= (10,8))
        self.ax1 = self.fig.add_subplot(111)
        self.ax1.set_xlim(*collec_range)
        self.ax1.set_ylim(*int_range)
        self.ax1.set_xlabel(’Wavelength (nm)’,fontsize = FONT[1])
        self.ax1.set_ylabel(’Intensity’,fontsize = FONT[1])
        self.line, = self.ax1.plot([], [], lw=2)

        #Defines the canvas to insert matplotlib plot
        self.canvas = FigureCanvasTkAgg(self.fig, master=self)
        self.canvas.draw()
        self.canvas.get_tk_widget().pack()

        #Inserts the matplotlib toolbar
        self.toolbar = NavigationToolbar2Tk(self.canvas, self)
        self.toolbar.update()


    def on_click(self):
        ’’’
        Method to decide the action of the collect button
        ’’’

        #Start live animation on first click
        if self.ani is None:

           return self.start()

        #If paused, stop animation
        elif self.running:

            self.ani.event_source.stop()
            self.btn.config(text=’ Collect ’)

        #If resumed, start animation
        else:

            self.ani.event_source.start()
            self.btn.config(text=’ Pause ’)

        #Change status
        self.running = not self.running

    def start(self):
        ’’’
        Method to start/resume animation
        ’’’

        #Signal that animation is running
        self.running = True

        #Change button label
        self.btn.config(text=’ Pause ’)

        #Start animation
        self.ani = animation.FuncAnimation(self.fig,
                                           self.update_graph,
                                           interval=int(self.slide.get())/1000,
                                           repeat=False)
        self.ani._start()


    def update_graph(self, i):
        '''
        Method to update the spectrograph plot with current data.
        '''

        #Retrieve updated integration time
        int_time = int(self.slide.get())

        #Update plot
        self.line.set_data(*get_data(int_time))
        self.ax1.set_ylim(0,self.slide_max.get())

        return self.line,



def main():
    '''
    Compile and run main application.
    '''

    #Create tk object
    root = tk.Tk()

    #Set application icon
    root.iconbitmap(default = ’light_icon.ico’)

    #Compile and execute application
    App(root).pack()
    root.mainloop()


if __name__ == ’__main__’:
    main()
```