Automating Nano-Optical Measurements

Supervised Learning Project (PH 303)

Spandan Sachin Anaokar
210260055

Guide: *Prof. Anshuman Kumar*

3rd April 2023

Contents

1 Introduction

When we have to gather data for a huge matrix of points, it becomes difficult to do it manually. For each point, you have to move to get to that point and then you have to take readings. These steps are too tedious to do by hand. The objective of this project is to design a code that enables the recording of data from a spectroscope and another code that programs stage movement on the nano-scale. The final phase of this project involves the integration of both codes into a single one. Hence this would automate several complicated procedures and thus greatly increase the efficiency of research. Overall, the project aims to provide an efficient and automated method for data gathering in scientific research. It is expected that the developed code will be useful for researchers in various fields whereby they can save a lot of effort and time.

1.1 Spectrometer

Photoluminescence (PL) is an optical method that is nondestructive and contactless. When a sample is irradiated with light, photoexcitation occurs, whereby the light is absorbed and imparts excess energy into the atoms of the material. This excess energy can be dissipated in two ways: through the emission of light or through luminescence. The emitted light can be analyzed spectrally, spatially, and temporally to obtain valuable information about the electronic properties of the material.

A PL spectrometer is a tool used to analyze the emissions from a sample during the photoexcitation process. The spectrometer converts the emitted light into a set of data that shows the intensity of the light at different wavelengths or frequencies. This data can be used to generate a graph of intensity versus wavelength or frequency, which provides insights into the electronic structure of the material.

(a) CCS 175

(b) Andor Kymera 328i

Figure 1: Spectrometer

In this project, we primarily worked with Thorlabs CCS and Andor Kymera spectrometers. However, we have also developed a model that can extract data from samples using any

spectrometer, as long as we can map the initialization and wavelength scanning functions of the spectrometer.

1.2 Stage - Nanopositioning Device

By its original definition, a nanopositioning device is a mechanism capable of repeatedly delivering motion in increments as small as one nanometer. The ones we use are mostly piezo stages. A piezo stage, or piezoelectric stage, is a surface that is controlled by a piezo motor. The piezoelectric actuator is composed of a piezo motor plus mechanical elements which allow the movement of the stage. To control the movement of the stage, modern piezo stages are fitted with a controller and position feedback device and manipulated via a computer interface.

Piezo motors can move in tiny increments in rapid-fire bursts, and then become very stable. Also, after a voltage is applied to move the piezo stage, it remains in that position even when power is lost. Thus Piezo stages are very advantageous as compared to other stages in short-distance movements.

In the project, stages are mainly used as a way to move the sample by nanoscale distances. The stage we use here is the Thorlabs Nanomax 300 stage. It can move in X, Y, & Z axes. The movement can be controlled by a piezo-controller which has 3 knobs used to set up the voltages in the XYZ directions. This directly corresponds to the motion of the stage in these directions.

(a) Thorlabs Nanomax Stage

(b) Thorlabs Stage Controller

Figure 2: Piezo Stage

2 Phase I: Thorlabs CCS

The purpose of this phase of the project is to develop a **Python Code** which would control the spectrometer, collect data, and store or display it in a presentable format. Currently, we can use an application *ThorSpectra* which has features that enable us to read data from the spectrometer and to save the data in a file.

Purpose: Designing a Code that would perform these tasks on running it and give the same output.

2.1 Capture Data and Store it

Referring to a Github Repository available online, we used it to make a primary code that takes in data from a connected spectrometer and stores it. For this code, we just need to install the drivers needed for communication with the device through USB that are installed automatically if we install the ThorSpectra application.

```
# -*- coding: utf-8 -*-
"""
Example of C Libraries for CCS Spectrometers in Python with CTypes

"""
import os                                       #Necessary imports
import time
import matplotlib.pyplot as plt
import numpy as np
from numpy import savetxt
from ctypes import *
import timeit

os.chdir(r"C:\Program Files\IVI Foundation\VISA\Win64\Bin")
lib = cdll.LoadLibrary("TLCCS_64.dll")

ccs_handle=c_int(0)

#documentation: C:\Program Files\IVI Foundation\VISA\Win64\TLCCS\Manual

#Start Scan- Resource name will need to be adjusted
#windows device manager -> NI-VISA USB Device -> Spectrometer ->
↪  Properties -> Details -> Device Instance ID
lib.tlccs_init(b"USB0::0x1313::0x8087::M00796613::RAW", 1, 1,
↪  byref(ccs_handle))

#set integration time in  seconds, ranging from 1e-5 to 6e1
integration_time=c_double(10.0e-3)
lib.tlccs_setIntegrationTime(ccs_handle, integration_time)

```

```
def main():
    #start scan
    lib.tlccs_startScan(ccs_handle)

    wavelengths=(c_double*3648)()

    lib.tlccs_getWavelengthData(ccs_handle, 0, byref(wavelengths),
    ↪  c_void_p(None), c_void_p(None))

    #retrieve data
    data_array=(c_double*3648)()
    lib.tlccs_getScanData(ccs_handle, byref(data_array))

    #plot data
    wavelengths_python = np.ndarray((3648, ), 'f', wavelengths, order='C')
    data_array_python  = np.ndarray((3648, ), 'f', data_array, order='C')
    #Change float f to double

    tempMatrix = np.array([wavelengths_python, data_array_python])
    tempMatrix = np.transpose(zipped)

    print(wavelengths_python)
    print(data_array_python)

    savetxt('..\Thorlab CSS\Single data.txt', tempMatrix, delimiter='\t')
    ↪  # Add location where data is to be saved
    plt.plot(wavelengths, data_array)       #Plot the data

main()
#close
lib.tlccs_close (ccs_handle)
```

Here we can see that the code is quite accurate. The data recorded from the software and the code are nearly the same.

2.2 Repeatedly Capture and Store Data after fixed Time interval

This above snippet of code only takes a single reading. We need to modify the code in such a way that the data is captured repeatedly after fixed intervals. The following code snippet depicts this:

```
def main():
    timeslices = 5                      # Interaval between consecutive
    ↪  data readings
    numberOfReading = 2                 # No of readings to be taken
    lib.tlccs_setIntegrationTime(ccs_handle, integration_time)

```

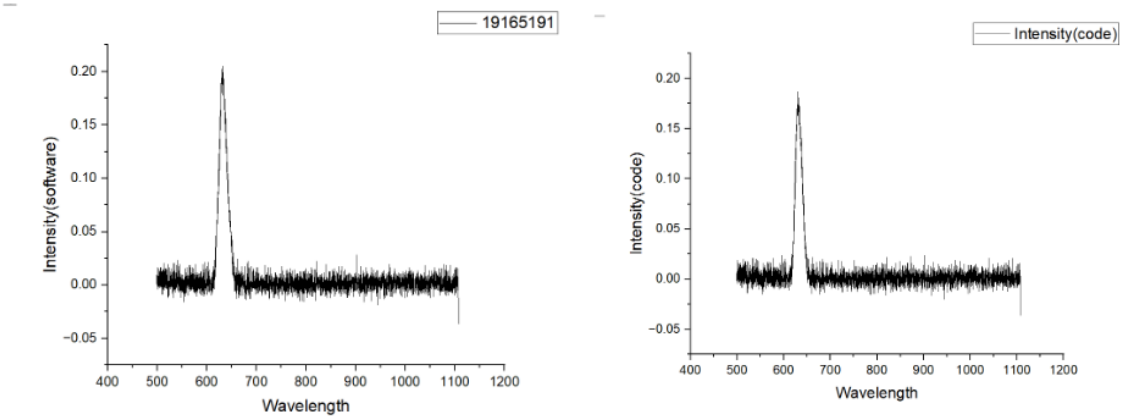


Figure 3: Data Comparison for software and code

```
    for count in range(numberOfReading):
        startTime = timeit.default_timer()
        #denotes the time when loop starts

        #start scan
        lib.tlccs_startScan(ccs_handle)

        wavelengths=(c_double*3648)()

        lib.tlccs_getWavelengthData(ccs_handle, 0, byref(wavelengths),
        ↪  c_void_p(None), c_void_p(None))

        #retrieve data
        data_array=(c_double*3648)()
        lib.tlccs_getScanData(ccs_handle, byref(data_array))

        #plot data
        wavelengths_python = np.ndarray((3648, ), 'f', wavelengths,
        ↪  order='C')
        data_array_python  = np.ndarray((3648, ), 'f', data_array,
        ↪  order='C')
        #Change float f to double

        if(count==0):   #For first data also include the wavelength list
            resMatrix = pd.DataFrame(data=wavelengths_python,
            ↪  columns=["Wavelengths"])
            resMatrix["t = "+str(count*timeslices)] = data_array_python
        else:           # For new data just add them
            resMatrix["t = "+str(count*timeslices)] = data_array_python
```

```
31
32          print(resMatrix)
33          stopTime = timeit.default_timer()    # denotes time for loop to end
34          time.sleep(timeslices-(stopTime-startTime)) #wait for the
            ↪  timeslice amount of time to be over
35
36      #Save data. Here resMatrix contains the data
37      np.savetxt('...\Thorlab CCS\Timesliced data.txt', resMatrix,
        ↪  delimiter='\t') # Add location where data is to be saved
```

2.3 Live changing Plot by continuously Capturing Data

Now we make a Matplot graph of intensity vs wavelength and continuously update it making it a Live Plot. This gives us the feeling of a live video we are watching for a particular point

```
1  def main():
2      #start scan
3      time_show = 10  #duration of the plot in seconds
4      start = timeit.default_timer()  #set start time and then continuously
       ↪  update stop time
5      stop = start
6      while (stop-start)<=time_show:
7          lib.tlccs_startScan(ccs_handle)
8
9          wavelengths=(c_double*3648)()
10
11         lib.tlccs_getWavelengthData(ccs_handle, 0, byref(wavelengths),
           ↪  c_void_p(None), c_void_p(None))
12
13         #retrieve data
14         data_array=(c_double*3648)()
15         lib.tlccs_getScanData(ccs_handle, byref(data_array))
16
17         #plot data
18         plt.plot(wavelengths, data_array)
19         plt.title("Time: " + str(round(stop-start,2)) + " s")
20         plt.xlabel("Wavelength [nm]")
21         plt.ylabel("Intensity [a.u.]")
22         plt.grid(True)
23         plt.pause(0.01)
24         plt.cla()
25
26         stop = timeit.default_timer()
```

2.4 ThorCam

This is another type of input reader where the data is not collected in the form of an intensity matrix but rather we get an image just like a photo. However, the scientific camera contains extremely powerful components thus allowing us to view objects of extremely small dimensions. We have an application *ThorCam* which enables us to click pictures and even start a video.

We have thus designed a code to perform the same tasks through Python.

```
def generate_video(self):
    image_folder = '.'
    video_name = 'result2.mp4' #file name.
    os.chdir(".../ThorCam")
    # Change above path to where you want to video to be saved
    pil_images = list(self._image_queue.queue)
    images = [numpy.asarray(i) for i in pil_images]
    fourcc = cv2.VideoWriter_fourcc(*'DIVX')
    print(len(images))
    # Array images should only consider
    # The image files ignoring others if any

    # Setting the frame width, height width
    # The width, and height of the first image
    height, width, layers = images[0].shape

    video = cv2.VideoWriter(video_name, fourcc, 24, (width, height))

    # Appending the images to the video one by one
    for image in images:
        video.write(image)

    # Deallocating memories taken for window creation
    cv2.destroyAllWindows()
    video.release()  # releasing the video generated
```

3 Phase II: Thorlabs Nanomax

The nanomax contains of a stage and a controller. We set voltage values on the controller which correspond to different distances that the stage moves by. In this case, we don't have an application to move the stage. We have to do it manually by moving the knob on the controller.

Purpose: Design a code that would take in voltage values and the stage would move by that difference for a set of multiple points covering an area.

3.1 Move the stage to each position in a 2D matrix

The software of the Nanomax can be used to move the stage by the corresponding distance. Our purpose is to use a code to be able to do the same. For this, we require a *MDT_COMMAND_LIB.py* file which contains a set of functions that can be used, and a *MDT_COMMAND_LIB_x64.dll* file that contains required components to connect to the Nanomax stage. Our code shall use them to achieve our goals.

We will move to each point in a 2D matrix in a snake-like path. At each point, we run the *read()* function which should contain any operation you need to perform.

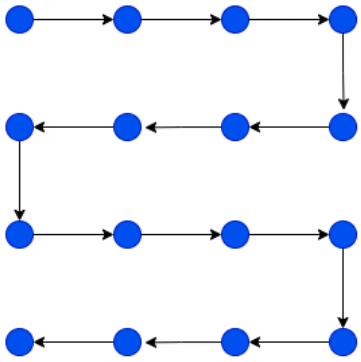


Figure 4: Path through which data is collected

```
#Primary code containing important functions

print("Enter no. of rows and cols of matrix: ")     #Input the size of the
↪  matrix of readings we want to move
n = int(input("Cols:"))
m = int(input("Rows:"))

devs = mdtListDevices()
#print(devs)
hdl = CommonFunc(serialNumber="1908086985-03") #Fill serial number of your
↪  model
```

```
initializeVoltage(0,0,0,hdl)

direct = 1                  #Determines if you are going left to right or
↪  opposite
col = 0                     #Column number
matrix = []
t = read(hdl)
for row in range(m):         #row number
    setY(row/10, hdl)        #Move stage to correct row
    temp = []
    if(col==-1):             #If we cross end go back to edge
        col=0
    elif(col==n):
        col=n-1
    while(0<=col<n):
        setX(col/10, hdl)    #Move stage to correct column
        t = read(hdl)
        temp.append([row/10,col/10, t])
        col+=direct
        time.sleep(1)
    direct = -direct         #reverse direction
    if(direct>0):            #append row data to matrix
        matrix.append(list(reversed(temp)))
    else:
        matrix.append(temp)
[print(matrix[row]) for row in range(len(matrix))]
print(read(hdl))
```

The following picture denotes how the data is stored. The first column is a list of wavelengths. The following columns each have an intensity value for a point moving in order (0,0), (1,0) ... corresponding to given wavelength. Note that the first line contains information about the size of the grid. This is used while extracting the data from the txt file.

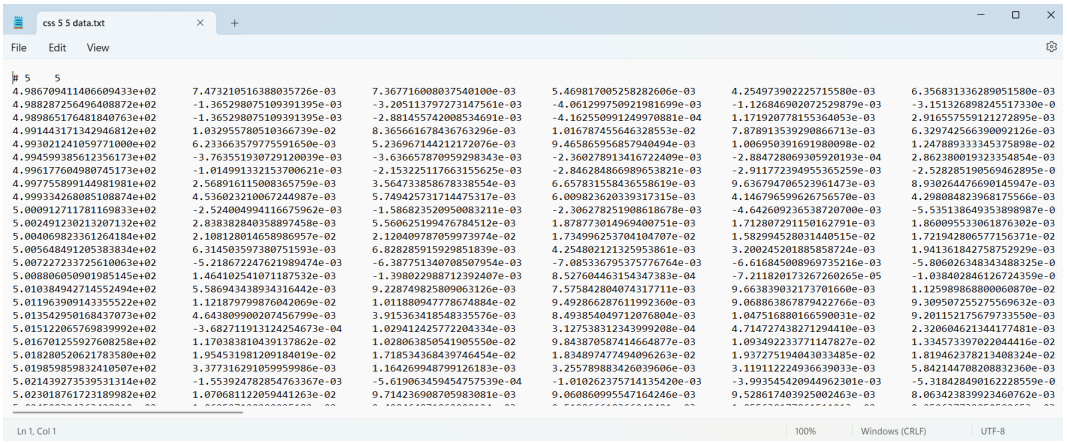


Figure 5: Data in txt file for 5×5

```
mdtIsOpen  1
www.thorlabs.comewton, NJ 07860ule
      Wavelengths          0,0          1,0          2,0          0,1          1,1          2,1          0,2          1,2          2,2
0     5.657128e-12  2.274849e+26 -4.585208e+26  7.374302e+18 -4.561318e-31  1.163722e+27 -5.348198e-30 -2.899604e+29  8.510173e+36 -6.017501e-27
1     3.986983e+00 -6.619728e-01 -7.878999e-01 -8.090690e-01 -8.233892e-01 -7.134864e-01  8.003075e-01  7.316747e-01 -9.046210e-01  4.905790e-01
2     5.366225e+32 -6.840398e-30  4.309977e-01  7.373825e+18  2.291066e-02 -2.890486e-24  1.664717e-22 -1.105881e-06  1.708550e+13  1.816935e+08
3     3.987137e+00 -7.687361e-01 -7.328677e-01  7.465690e-01  5.729746e-01  7.726681e-01 -6.554003e-01 -6.542513e-01  7.535673e-01 -8.002928e-01
4    -2.390302e+04  6.306020e+09 -1.231002e+11  1.682650e-03  9.168102e+05 -7.206897e-11 -4.618230e-32  6.036630e+26 -4.218911e+22  3.356317e+24
...            ...          ...          ...          ...          ...          ...          ...          ...          ...          ...
3643  4.277737e+00  8.320650e-01 -8.665801e-01 -6.641759e-01 -6.321576e-01  6.238809e-01 -7.936717e-01 -8.244788e-01  7.438658e-01  0.000000e+00
3644  2.548537e+17 -3.274040e-28 -6.788199e+01 -1.213342e-37  4.827749e+01  1.113417e-28 -5.150187e-15  4.294121e+05 -1.115001e-17  1.816935e+08
3645  4.277901e+00  7.554731e-01 -8.134760e-01  7.648621e-01  4.773043e-01  8.357212e-01 -6.979930e-01  8.871861e-01  8.145469e-01  8.002928e-01
3646 -1.036772e-12 -2.468724e-13  1.188239e-12  1.836122e+38  4.827749e+01  5.204876e+32  1.923913e-03  4.699461e+16  7.250552e+35  7.551882e-06
3647  4.278065e+00  7.277886e-01  7.961975e-01  7.847552e-01  4.773043e-01 -8.857778e-01 -8.263109e-01 -8.410604e-01 -7.317028e-01  8.763607e-01

[3648 rows x 10 columns]
PS C:\Users\spand>
```

Figure 6: Data stored as Python DataFrame

4 Phase III: Integration of Thorlabs CCS and Spectrometer

We now move to the final phase of the project. Until now whatever we did by using the code was something we could achieve by using the corresponding software. Now finally we are going to move out into unfamiliar territory by combining the two codes to be able to perform tasks that we couldn't do previously.

Purpose: Designing a code that moves over the entire region, takes readings, and stores them in one file.

4.1 Take in Data Points and store them

This code snippet is part of a larger code where the devices were initialized using classes and objects. This is very similar to that of the Thorlab Nanomax, however, this also deals with storing the data in a text file in a reversible way.

```
xLen = int(input("Cols:"))
yLen = int(input("Rows:"))
interval = 2
nanomax_obj = Nanomax(serialNumber="1908086985-03") #Detects automatically
↪  if MDT device present. If error occurs, manually fill in the value
ccs_obj = CCS(2)  #Give custom integration time else default 10.0e-3
nanomax_obj.initializeVoltage(0,0,0)
direction = 1   #means will move towards +X
xCor = 0
matrix = []
input("Click enter:")
wavelength = ccs_obj.getWavelength()
for yCor in range(yLen):
    nanomax_obj.setY(yCor*interval)
    rowResult = []
    if(xCor==-1):
        xCor=0
    elif(xCor==xLen):
        xCor=xLen-1
    while(0<=xCor<xLen):
        nanomax_obj.setX(xCor*interval)
        print([xCor*interval,yCor*interval])
        input()
        rowResult.append([xCor*interval,yCor*interval, ccs_obj.read()])
        xCor+=direction

    direction = -direction
    if(direction>0):
        matrix.append(list(reversed(rowResult)))
    else:
        matrix.append(rowResult)
```

```
nanomax_obj.initializeVoltage(0,0,0)


test = []
result = []

def store():
    result = pd.DataFrame({'Wavelengths':(wavelength)})
    for yCor in range(yLen):
        for xCor in range(xLen):
            result[str(xCor)+","+str(yCor)]=matrix[yCor][xCor][2]
    print(result)
    return result

result = store()
np.savetxt('...\Integrated\css test.txt',result, delimiter='\t',
    newline='\n', header=str(xLen)+"\t "+str(yLen)) #Add address where
    file is to be stored
```

Along with this, we have stored the data as a Pandas Dataframe variable. However, we may want to reform the variable back from the .txt file. For this purpose, there is another code snippet.

```
def read():
    file_path = '...\Integrated\css test.txt'
    sizeArr = np.loadtxt(file_path, max_rows=1, comments='!', dtype= str)
    print(sizeArr)
    xLen = int(sizeArr[1])
    yLen = int(sizeArr[2])
    df = pd.DataFrame(np.loadtxt(file_path))
    df.columns = ['Wavelengths']+[str(i%xLen)+","+str(i//int(xLen)) for i
        in range(xLen*yLen)]
    print(df)
```

4.2 Data Processing and Plotting

In the end, it is not sufficient to end up just collecting data. It is important to be able to perform a basic analysis of the data too. In the following code, we have added several features.

4.2.1 Get 2D matrix of intensity plots

```
def plot2d(matrix, xLen, yLen):          #Matrix contains the data, while
    xLen and yLen gives size of the 2d region
    figure, axis = plt.subplots(xLen, yLen) #Divide the plots into
        subplots
```

```
    for xCor in range(xLen):
        for yCor in range(yLen):    #For each subplot assign a graph
            axis[xCor, yCor].plot(matrix['Wavelengths'],
            ↪  matrix[str(xCor)+','+str(yCor)])
            axis[xCor, yCor].set_title(str(xCor)+','+str(yCor))
            axis[xCor, yCor ].tick_params(left = False, right = False ,
            ↪  labelleft = False ,
                labelbottom = False, bottom = False)
    plt.show()
```

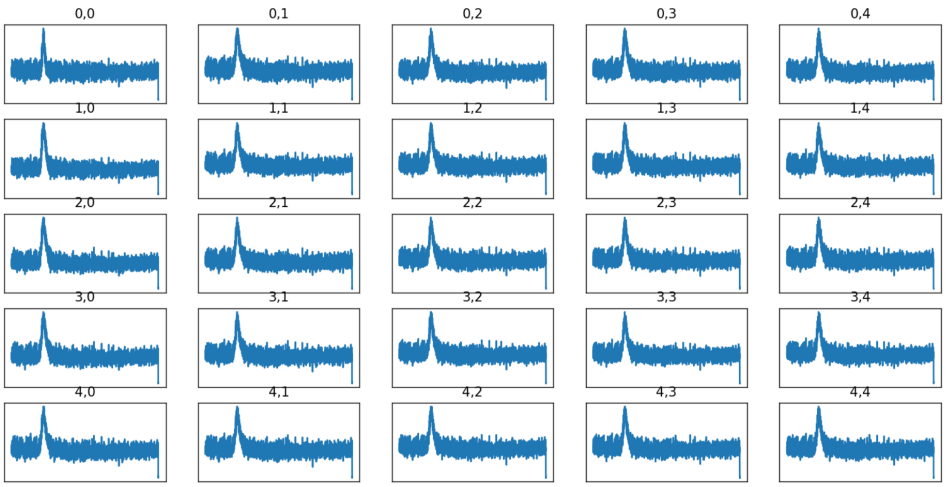


Figure 7: Intensity Plot of 2D matrix

4.2.2 Get a Color Plot of the max value of intensity

Clicking anywhere on the graph will open up a seperate graph showing intensity vs wavelength plot at that point.

```
def inten(matrix, xLen, yLen):          #Matrix contains the data, while
↪  xLen and yLen gives size of the 2d region
    maxIntensity = np.zeros((xLen, yLen))
    maxWave = np.zeros((xLen, yLen))
    for yCor in range(yLen):
        for xCor in range(xLen):
            maxwavelength = 0
            maxintensity = 0
            for i in range(len(matrix)):
                if(matrix[str(xCor)+','+str(yCor)][i]>maxintensity):
                    maxwavelength = matrix['Wavelengths'][i]
                    maxintensity = matrix[str(xCor)+','+str(yCor)][i]
```

```
            maxIntensity[yCor][xCor] = maxintensity
            maxWave[yCor][xCor]= maxwavelength
    print(maxIntensity, maxWave, sep='\n')
    return maxIntensity
#Above function gives the maximum intensity at all points

def colorplot(matrix, xLen, yLen):
    def on_click(event):
        if event.button is MouseButton.LEFT:
            xmouse, ymouse = event.xdata, event.ydata
            xmouse, ymouse = round(xmouse), round(ymouse)
            print(xmouse, ymouse)
            plt.figure(1)
            plt.cla()
            plt.plot(matrix['Wavelengths'],
            ↪  matrix[str(xmouse)+","+str(ymouse)])
            plt.xlabel("Wavelength [nm]")
            plt.ylabel("Intensity [a.u.]")
            plt.title(str(xmouse)+','+str(ymouse))
            plt.grid(True)
            plt.show()


    figure0 = plt.figure(0)
    figure0.canvas.callbacks.connect('button_press_event', on_click)

    x = [i for i in range(xLen)]
    y = [j for j in range(yLen)]
    z = inten(matrix, xLen, yLen)

    X, Y = np.meshgrid(x, y)
    #plt.pcolor(X, Y, z)
    plt.imshow(z,origin='lower',interpolation='bilinear')
    plt.colorbar()
    plt.show()
```

4.3 Click to Scan

This algorithm seeks to create a grid of the region where we want to measure the data. Now clicking on any point on the grid will result in the stage moving to that point and then the spectrometer will gather data regarding that point. The full code requires initializing the devices and several other functions which are attached in the appendix.

```
def scan(x, y):
    nanomax_obj.setX(x) #Move stage to mentioned coordinates
    nanomax_obj.setY(y)

```

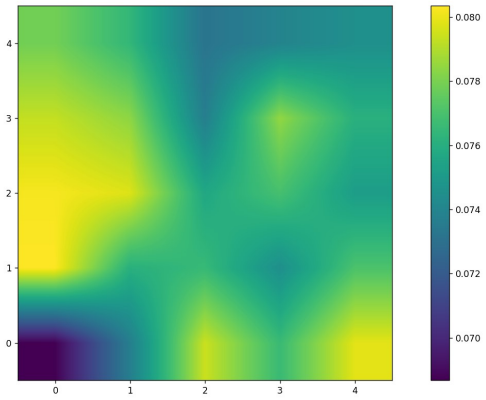
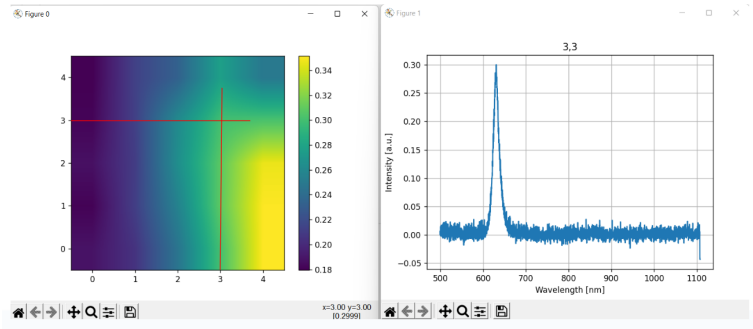


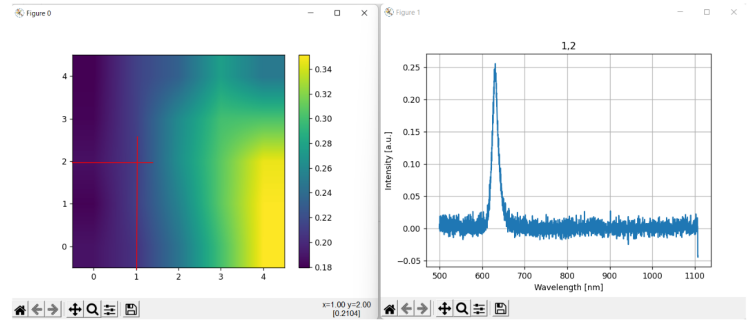
Figure 8: Color Plot

```
5
6       #start scan
7       lib.tlccs_startScan(ccs_handle)
8
9       wavelengths=(c_double*3648)()
10
11      lib.tlccs_getWavelengthData(ccs_handle, 0, byref(wavelengths),
        ↪  c_void_p(None), c_void_p(None))
12
13      #retrieve data
14      data_array=(c_double*3648)()
15
16      lib.tlccs_getScanData(ccs_handle, byref(data_array))
17
18      wavelengths_python = list(wavelengths)  #Change c type to Python data
        ↪  type
19      data_array_python  = list(data_array)   #Change c type to Python data
        ↪  type
20      temp_matrix = np.array([wavelengths_python, data_array_python])
21
22      temp_matrix = np.transpose(temp_matrix)
23      print(temp_matrix)
24      return wavelengths, data_array  #Return the data at that point
25
26  def grid(xLen, yLen):   #xLen and yLen refers to max size of area of
    ↪  interest
27      def on_click(event):    #On clicking the mouse the coordinates are
        ↪  taken
28          if event.button is MouseButton.LEFT:
29              xmouse, ymouse = event.xdata, event.ydata
```

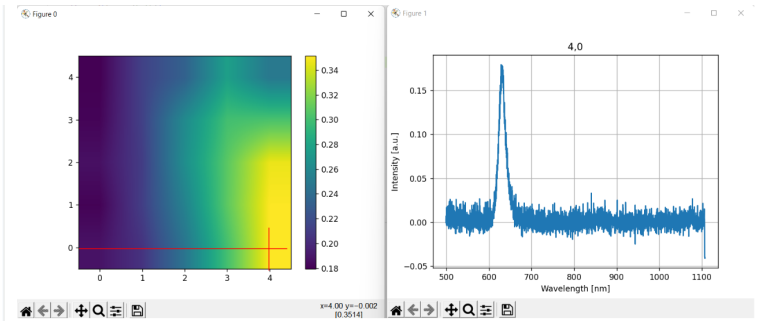
```
            #xmouse, ymouse = round(xmouse), round(ymouse)
            plt.cla()
            plt.xlim([0,xLen])
            plt.ylim([0,yLen])
            plt.grid(True, 'both')
            plt.plot(xmouse, ymouse, '+b')
            print(xmouse, ymouse)

            wavelen, data = scan(xmouse, ymouse)

            plt.figure(1) #Use data points to make another graph
            plt.cla()
            plt.plot(wavelen, data)
            plt.xlabel("Wavelength [nm]")
            plt.ylabel("Intensity [a.u.]")
            plt.title(str(xmouse)+','+str(ymouse))
            plt.grid(True)
            plt.show()

    print(np.zeros((xLen, yLen), int))
    figure0 = plt.figure(0)
    figure0.canvas.callbacks.connect('button_press_event', on_click)


    X, Y = np.meshgrid(xLen, yLen)  #Make the grid where you can click
    plt.plot()
    plt.xlim([0,xLen])
    plt.ylim([0,yLen])
    plt.grid(True, 'both')
    #plt.imshow(np.zeros((xLen, yLen)),
    ↪  origin='lower',interpolation='bilinear')
    plt.show()
```



(a)



(b)



(c)

Figure 9: Using Click to Scan at different points

5 Conclusion

In conclusion, this project aims to develop an efficient and automated algorithm for gathering data in scientific research. By designing a code that enables the recording of data from a spectrometer and at the same time commands the stage to move, we were able to automate an extremely tedious task. This provides an oversight into the potential that automation could have in increasing the efficiency of research. This project ends with developing several useful models for the tasks related to the integration of an input taker and input parameter changer. We can make changes to suit different spectrometers and different parameters. We hope that the project will be able to help researchers in different fields.

6 References

1. Thorlabs CCS175
2. Andor Kymera 328i
3. Piezo Stages
4. Thorlabs Nanomax
5. Github Reference for CCS Code
6. Nanomax SDK for Development

7 Appendix

The main of the project contains only important snippets of code. the rest of the code is necessary as well. A lot of the other part has been taken from Open Source Github Repositories and SDK(Software Development Kits) provided by the device makers.

This is the Link to the project's Github Repo:
https://github.com/Spandan2003/Automating-Nano-Optical-Measurements