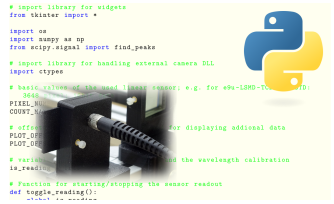
Spektralkalibrierung eines Spektrometers mit einer Glimmlampe

Dieses Applikationsbeispiel beschreibt den Prozess der spektralen Kalibrierung eines Spektrometers am Beispiel des Czerny-Turner-Spektrometers CTS-150. Dieses Verfahren kann jedoch auch für andere Spektrometertypen mit ähnlichem Aufbau verwendet werden. Es verwendet ein kurzes Python-Skript namens *EasyCalibrate*, das eine Erweiterung unseres Skripts *EasyDisplay* ist. Das Skript stellt die Sensordaten grafisch dar und nutzt die manuelle Identifizierung charakteristischer Peaks im Spektrum einer Glimmlampe für den Kalibrierungsprozess.



1 Physikalischer Hintergrund

Ein Spektrometer verwendet ein optisches Beugungselement (z. B. ein Gitter oder Prisma), um die verschiedenen Wellenlängen in einem Lichtsignal zu trennen. Zur Detektion der getrennten Wellenlängen wird häufig eine Zeilenkamera verwendet. Abhängig von der jeweiligen optischen Anordnung entspricht jede Pixelposition einem bestimmten Wellenlängenbereich.

Um ein Spektrometer zu kalibrieren, muss man die Korrelation zwischen Pixelposition und Wellenlänge des Signals kennen. Eine einfache Kalibrierungsmethode, die dennoch für die meisten grundlegenden Anwendungen geeignet ist, besteht darin, ein Signal von einer bekannten Lichtquelle mit schmalen Emissionslinien zu verwenden. Solche Linien mit bekannten und festen Wellenlängen werden z. B. von Glimmlampen emittiert.

2 Anforderungen

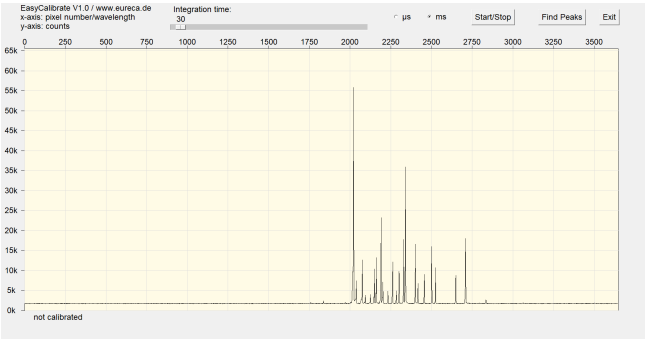
Sie benötigen ein Czerny-Turner-Spektrometer CTS-150, eine Glimmlampe mit bekannten Peaks sowie einen Computer mit dem Skript *EasyCalibrate*. Einzelheiten zum Herunterladen und Installieren des Python-Skripts auf Ihrem Computer finden Sie in der Dokumentation für das Skript *EasyDisplay*: https://www.eureca.de/4950-0-EasyDisplay.html.

Glimmlampen sind sehr billige Bauteile, die z. B. noch als Statuslampen in elektronischen Geräten oder Schaltsteckern verwendet werden. Das Spektrum dieser Bauteile zeigt ein charakteristisches Peakmuster im Wellenlängenbereich zwischen 580 nm und 710 nm. In diesem Bereich sind etwa 22 Peaks mit unterschiedlichen Stärken sichtbar.

Wir bieten geeignete Kalibriermodule an, die auch als Bausatz selbst aufgebaut werden können. Details zur Gewinnung des Spektrums einer Glimmlampe finden Sie in unserem Applikationsbeispiel *Spektroskopie an Glimmlampen* (https://www.eureca.de/4911-0-Spektroskopie-an-Glimmlampen.html).

3 Kalibrierungsprozess

Stellen Sie sicher, dass die Zeilenkamera des Czerny-Turner-Spektrometers korrekt an den Computer angeschlossen ist, und starten Sie das Skript *EasyCalibrate*. Schalten Sie die Glimmlampe ein, lassen Sie sie eine Minute lang warmlaufen und bringen Sie den Lichtleitereingang des Spektrometers so nah wie möglich an die Lampe heran. Wenn Sie ein Beleuchtungs- oder Kalibrierungsmodul verwenden, können Sie den Lichtleiter einfach an den entsprechenden Lichtquellenausgang anschließen. Sie sollten nun die charakteristischen Spitzen des Neonspektrums auf Ihrem Computerbildschirm sehen (Abb. 1; alle Abbildungen sind am Ende des Dokuments vergrößert dargestellt).

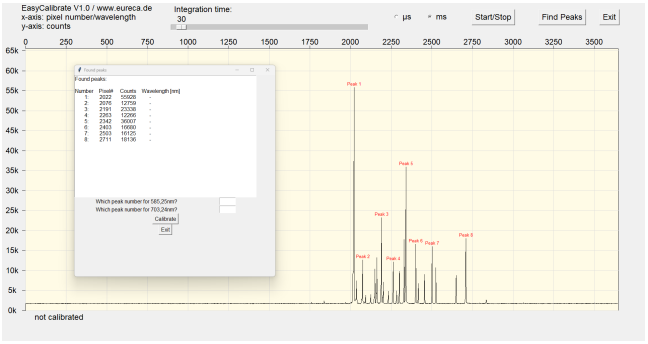


1: Spektrum einer Glimmlampe, mit unkalibriertem Spektrometer

Das Skript verwendet eine externe Textdatei mit dem Namen `EasyCalibrate.config`, um bereits erhaltene Kalibrierungsdaten zu speichern. Wenn Sie das Skript zum ersten Mal ausführen und diese Datei nicht vorhanden ist, wird

die x-Achse des Signalplots mit »not calibrated« gekennzeichnet. Wenn die Kalibrierungsdatendatei gefunden wurde, wird die x-Achse mit den entsprechenden Wellenlängen beschriftet, die aus den gespeicherten Kalibrierungswerten berechnet wurden.

Stellen Sie nun die Integrationszeit so ein, dass der größte Peak des Spektrums eine Höhe von ca. 55k counts hat. Mit einer solchen Integrationszeit wird das Sensorsignal nicht durch mögliche Sättigungseffekte, die bei höheren Intensitäten auftreten können, verfälscht und die Peaks können vom Skript zuverlässig gefunden werden.

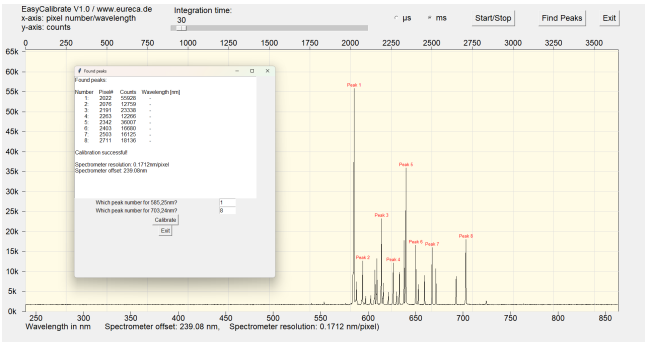


2: Spektrum mit gefundenen und nummerierten Peaks

Drücken Sie die Schaltfläche »Find Peaks« und es öffnet sich ein neues Fenster, in dem die gefundenen Peaks aufgelistet sind (Abb. 2). Wenn das Spektrometer noch nicht kalibriert ist, wird in dieser Liste nur die Peaknummer, die Pixelposition und die maximale Peakanzahl angezeigt. Die jeweilige Wellenlänge wird dann mit einem Bindestrich angezeigt: »-«.

Die Peak-Erkennung erfolgt durch das Skript mit Hilfe der Funktion `find_peaks` des SciPy-Moduls, die sehr leistungsfähig und gleichzeitig einfach zu bedienen ist. Neben der Auflistung der Peaks werden die entsprechenden Signale im Spektrum zu Referenzzwecken auch mit Nummern versehen.

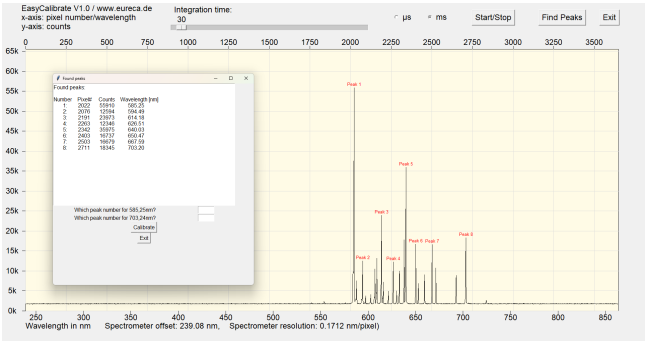
Für die Kalibrierung verwendet das Skript die Neon-Peaks bei 585,25 nm und 703,24 nm, die wir hier *Kalibrierungspeaks* nennen werden. Diese Kalibrierungspeaks sind recht einfach zu finden, da sie sowohl das erste als auch das letzte hohe Signal der Peakreihe sind. Abhängig von der Produktionscharge der Lampe und der verwendeten Versorgungsspannung kann die Höhe der einzelnen Peaks leicht variieren, aber die Kalibrierungspeaks sind normalerweise so dominant, dass sie kaum verwechselt werden können. Die Toleranzen in den Peakhöhen können jedoch die Reihenfolge beeinflussen, in der die einzelnen Peaks vom Skript gefunden und beschriftet werden.



3: Kalibriertes Spektrum

Wenn Sie eine andere Lichtquelle zur Kalibrierung des Spektrometers verwenden, müssen Sie eventuell die verwendeten Wellenlängen für die Kalibrierungspeaks im Skript ändern.

Unterhalb dieser Liste mit den gefundenen Peaks befinden sich zwei Eingabefelder. Hier kann die Peaknummer für die Kalibrierungspeaks eingegeben werden. Im gezeigten Beispiel sind dies die Nummern #1 und #8. Nach Betätigung der Schaltfläche »Calibrate« führt das Skript eine lineare Regression anhand der Positionen der Referenzpeaks durch und berechnet die Auflösung des Spektrometers sowie den Offset, also die Wellenlänge für das erste Sensorpixel (Abb. 3). Diese beiden Werte werden nun zur Kalibrierung der x-Achse verwendet und die entsprechenden Markierungen werden eingezeichnet.



4: Liste der gefundenen Peaks mit kalkulierten Wellenlängen

Nach der Kalibrierung des Spektrums können Sie jederzeit erneut die Schaltfläche »Find Peaks« drücken, um neue Peaks im aktuellen Sensorsignal zu finden. Zu den gefundenen Peaks werden nun auch die berechneten Wellenlängen aufgelistet. Diese Funktion kann also auch zur Messung unbekannter Peaks verwendet werden (Abb. 4).

4 *EasyCalibrate* Python-Code

```
# EasyCalibrate.py V1.1
#
# Python display tool for line scan cameras by EURECA Messtechnik GmbH
# - detects the camera via the EasyAccess DLL
# - reads out the camera and displays the recorded sensor data
# - integration time can be adjusted between 1µs and 1000ms
#
# For details please refer to: www.eureca.de

# import library for widgets
from tkinter import *

import os
import numpy as np
from scipy.signal import find_peaks

# import library for handling external camera DLL
import ctypes

# basic values of the used linear sensor; e.g. for e9u-LSMD-TCD1304-STD:
   3648 pixel; 16bit data
PIXEL_NUMBER = int(3648)
COUNT_MAX = int(65536)

# offsets for the plot window; needed for displaying addional data
PLOT_OFFSETX = 70
PLOT_OFFSETY = 50

# variable to test the camera status and the wavelength calibration
is_reading = True

# Function for starting/stopping the sensor readout
def toggle_reading():
    global is_reading
    is_reading = not is_reading

def Plot_Sensordata_Wavelength():

  # axis marking for wavelength
  wavelength_start = spectrometer_offset
  wavelength_end = spectrometer_offset + 3648*spectrometer_resolution

  plot.delete("wavelength")

  for x in range(100,1200,50):
    if (x > wavelength_start) and (x < wavelength_end):
     # print(x)
     # print(wavelength_to_pixel(x))

      plot.create_line(PLOT_OFFSETX +
         wavelength_to_pixel(x)*plot_width/PIXEL_NUMBER, plot_height +
         PLOT_OFFSETY, PLOT_OFFSETX + \
                      wavelength_to_pixel(x)*plot_width/PIXEL_NUMBER,
                         plot_height + PLOT_OFFSETY + 10,
```

```
                            tags="wavelength")
        plot.create_text(PLOT_OFFSETX +
            wavelength_to_pixel(x)*plot_width/PIXEL_NUMBER, plot_height +
            PLOT_OFFSETY + 20, font=("Arial Bold",
            18),text=str(x),fill='black', \
                        tags="wavelength")

    spectrometer_offset_output = "Spectrometer offset: " +
        str(spectrometer_offset) + str(" nm")
    spectrometer_resolution_output = "Spectrometer resolution: " +
        str(spectrometer_resolution) + str(" nm/pixel")
    wavelength_output ="Wavelength in nm      " + spectrometer_offset_output
        + ",    " + spectrometer_resolution_output + ")"
    plot.create_text(PLOT_OFFSETX, plot_height + PLOT_OFFSETY + 45,
        font=("Arial Bold",18),text=wavelength_output,fill='black',
        tags="wavelength", anchor="w")

# calibrate the spectrometer using the two choosen peaks from the neon
    spectrum
def calibrate_sensor():

    global spectrometer_offset, spectrometer_resolution

    # get the wavelengths of the two choosen peaks
    wavelength_1 = peaks[int(peak_num_entry.get())-1]
    wavelength_2 = peaks[int(peak_num_entry2.get())-1]

    # calculate the spectrometer resolution in nm/pixel
    spectrometer_resolution = round((703.24 - 585.25) /  (wavelength_2 -
        wavelength_1),4)
    # calculate the offset of the spectrometer, which is defined as the
        wavelength at the first pixel
    spectrometer_offset = round(585.25 - wavelength_1 *
        spectrometer_resolution,2)

    output_text.insert(END, "\nCalibration successful!\n\n")
    output_text.insert(END, "Spectrometer resolution: " +
        str(spectrometer_resolution) + "nm/pixel")
    output_text.insert(END, "\nSpectrometer offset: " +
        str(spectrometer_offset) +"nm")

    is_calibrated = True
    Plot_Sensordata_Wavelength()

    # writing configuration file
    file = open("EasyCalibrate_config.txt", "w")
    file.write("spectrometer_offset " + str(spectrometer_offset) + "\n")
    file.write("spectrometer_resolution " + str(spectrometer_resolution))
    file.close()

def wavelength_to_pixel(wavelength):

  global spectrometer_offset, spectrometer_resolution

  # print(wavelength)
  pixel = (wavelength - spectrometer_offset) / spectrometer_resolution
```

```
    # print(pixel)
    return pixel

def close_peak_window():

    plot.delete("peaks")
    peak_window.destroy()

def find_and_show_peaks():

    global peaks, peak_num_entry, peak_num_entry2, output_text, peak_window

    # Receive sensor values
    sensor_data = [pointer[pixel] for pixel in range(PIXEL_NUMBER)]

    # Finding peaks with the find_peaks function of SciPy
    peaks, _ = find_peaks(sensor_data, height=10000, distance=50)

    # Create a new window for the output of the peaks found
    peak_window = Toplevel(master)
    peak_window.title("Found peaks")
    peak_window.geometry("600x600")

    # Text field for displaying the peaks found
    output_text = Text(peak_window, font=("Arial", 12), width=60, height=20)
    output_text.grid(row=0, column=0, columnspan=2)

    # Entry fields for peak numbers
    peak_num_label = Label(peak_window, text="Which peak number for
       585,25nm?", font=("Arial", 12))
    peak_num_label.grid(row=1, column=0)
    peak_num_entry = Entry(peak_window, font=("Arial", 12), width=5)
    peak_num_entry.grid(row=1, column=1)

    peak_num_label2 = Label(peak_window, text="Which peak number for
       703,24nm?", font=("Arial", 12))
    peak_num_label2.grid(row=2, column=0)
    peak_num_entry2 = Entry(peak_window, font=("Arial", 12), width=5)
    peak_num_entry2.grid(row=2, column=1)

    # Calibration button
    calibrate = Button(peak_window, text="Calibrate", font=("Arial", 12),
       command=calibrate_sensor)
    calibrate.grid(row=3, column=0, columnspan=2)

    # Button for closing the window
    exit_button = Button(peak_window, text="Exit", font=("Arial", 12),
       command=close_peak_window)
    exit_button.grid(row=4, column=0, columnspan=2)

    # Output of the peaks found in the text field
    output_text.insert(END, "Found peaks:\n\nNumber    Pixel#     Counts
       Wavelength [nm]\n")
    for idx, peak_index in enumerate(peaks):
        wavelength = spectrometer_offset + peak_index *
```

```
            spectrometer_resolution
        if wavelength > 0:
          output_text.insert(END, f"{idx+1:>8}:       {peak_index:>5}
             {sensor_data[peak_index]:>5}          {wavelength:6.2f}\n")
        else:
          output_text.insert(END, f"{idx+1:>8}:       {peak_index:>5}
             {sensor_data[peak_index]:>5}          -\n")

    # Update plot and number peaks consecutively
    for idx, peak_index in enumerate(peaks):
        counts = sensor_data[peak_index]
        x = int(peak_index / PIXEL_NUMBER * plot_width)
        y = int(counts / COUNT_MAX * plot_height)

        # Update plotting of the peak marker above the PeakPlot and number
            peaks consecutively
        plot.create_text(x + PLOT_OFFSETX, plot_height + PLOT_OFFSETY - y -
            10,
                         font=("Arial", 10), text=f"Peak {idx+1}",
                            fill="red", tags="peaks")

print("EasyCalibration V1.1\nSearching for camera: ")

# open external DLL
libe9u = ctypes.WinDLL('./libe9u_LSMD_x64.dll')

# define argument and return types for the used functions
libe9u.e9u_LSMD_search_for_camera.argtype = ctypes.c_uint
libe9u.e9u_LSMD_search_for_camera.restype = ctypes.c_int

libe9u.e9u_LSMD_start_camera_async.argtype = ctypes.c_uint
libe9u.e9u_LSMD_start_camera_async.restype = ctypes.c_int

libe9u.e9u_LSMD_set_times_us.argtypes = (ctypes.c_uint, ctypes.c_uint,
    ctypes.c_uint)
libe9u.e9u_LSMD_set_times_us.restype = ctypes.c_int

libe9u.e9u_LSMD_get_next_frame.argtype = ctypes.c_uint
libe9u.e9u_LSMD_get_next_frame.restype = ctypes.c_int

libe9u.e9u_LSMD_get_pixel_pointer.argtypes = (ctypes.c_uint, ctypes.c_uint)
libe9u.e9u_LSMD_get_pixel_pointer.restype = ctypes.POINTER(ctypes.c_uint16)

# Seach for a suitable camera on all USB ports and quit with returning the
    error code, if no camera is found
i_status = libe9u.e9u_LSMD_search_for_camera(0)
if i_status != 0:
  print("No camera found! Error Code: " + str(i_status))
  exit(1)

print("Starting camera: ", end='')
libe9u.e9u_LSMD_start_camera_async(0)

# getting the pointer to the array containing the sensor data
pointer = libe9u.e9u_LSMD_get_pixel_pointer(0, 0)
```

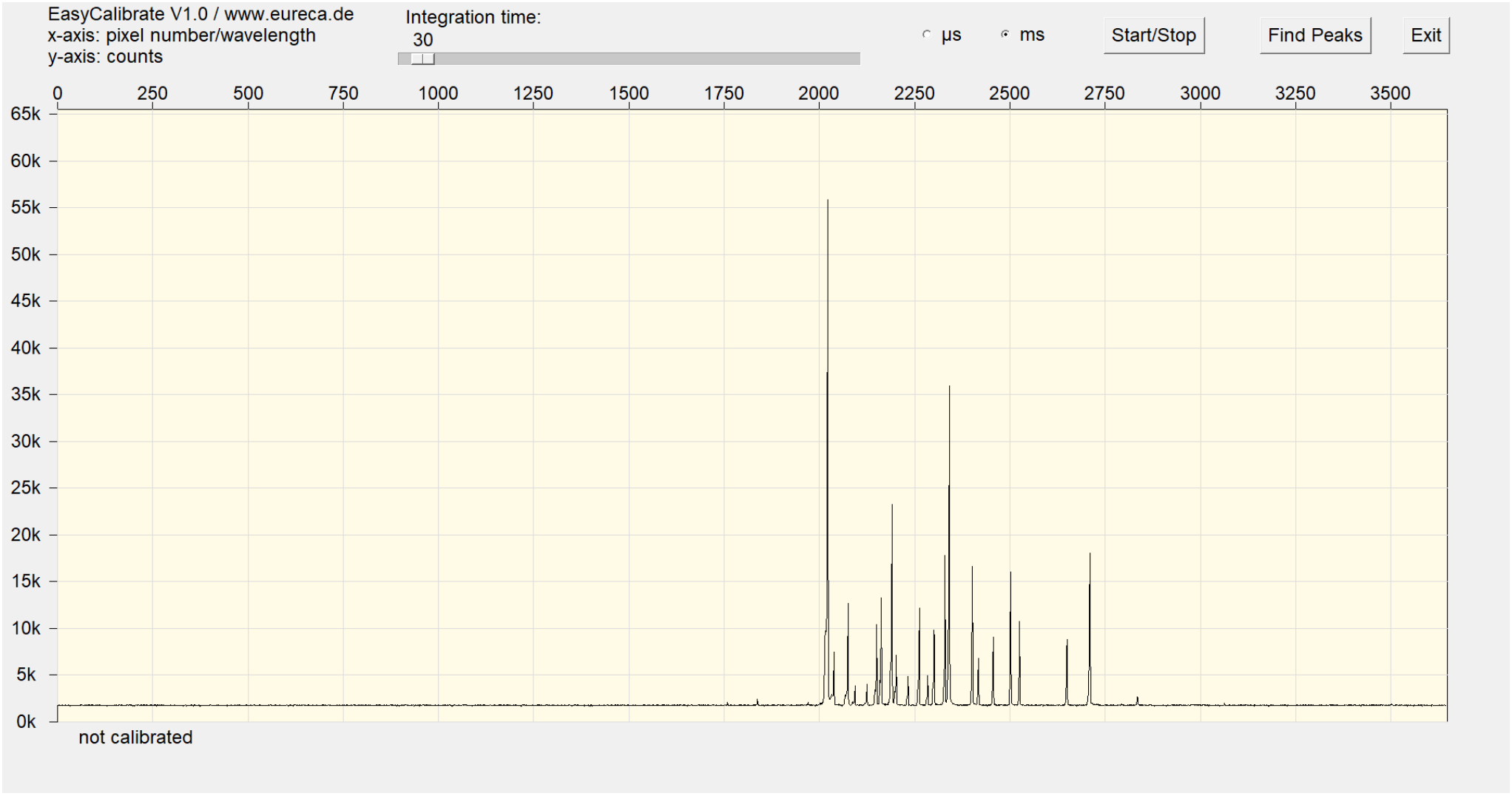
```
# defining the master window for graphical output
master = Tk()

# getting the size of the master window
screen_width = master.winfo_screenwidth()
screen_height = master.winfo_screenheight()

# setting the size of the display window to cover nearly the complete screen
master.geometry(str(screen_width - 50) + "x" + str(screen_height - 100) +
    "+10+20")

# defining the dimensions for the plot area
plot_width = screen_width - 150
plot_height = screen_height - 300

# defining and packing the control/output widgets
statusline = Frame(master)
statusline.pack(side='top')
plot = Canvas(master)
plot.pack(side ='bottom', fill=BOTH, expand=YES)

# output label for program name and version number
output_peak_width = Label(statusline, text="EasyCalibrate V1.1 /
    www.eureca.de\nx-axis: pixel number/wavelength\ny-axis: counts",
    justify=LEFT, font=("Arial Bold", 18))
output_peak_width.pack(side='left', padx=0)

# defining slider for integration time and setting it to 10
faktor = IntVar()
slider_exp_time = Scale(statusline, from_=1, to=1000, length=plot_width/3,
    orient=HORIZONTAL, label="Integration time:", font=("Arial Bold", 18))
slider_exp_time.pack(side='left', padx=50)
slider_exp_time.set(100)

# defining two radio buttons for switching the integration time between µs
    and ms
Radiobutton_us = Radiobutton(statusline, text="µs", font=("Arial Bold",
    18), variable=faktor, value=1)
Radiobutton_us.pack(side='left', padx=20)
Radiobutton_us.invoke()
Radiobutton_ms = Radiobutton(statusline, text="ms", font=("Arial Bold",
    18), variable=faktor, value=1000)
Radiobutton_ms.pack(side='left', padx=20)

start_stop_button = Button(statusline, text="Start/Stop", font=("Arial
    Bold", 18), command=toggle_reading)
start_stop_button.pack(side='left', padx=50)

find_peaks_button = Button(statusline, text="Find Peaks", font=("Arial
    Bold", 18), command=find_and_show_peaks)
find_peaks_button.pack(side='left', padx=20)

# defining the exit button with the closing function
def close_window():
    master.destroy()
exit_button = Button(statusline, text="Exit", font=("Arial Bold", 18),
```

```
    command=close_window)
exit_button.pack(side='left', padx=20)

# drawing a rectangular frame for the sensor data
Sensor_Plot = plot.create_rectangle(PLOT_OFFSETX, PLOT_OFFSETY, plot_width
    + PLOT_OFFSETX, plot_height + PLOT_OFFSETY, fill="#fffbe6")

# x-axis marking with pixel number
for x in range(15):
  plot.create_line(PLOT_OFFSETX + x*250*plot_width/PIXEL_NUMBER,
     PLOT_OFFSETY, PLOT_OFFSETX + x*250*plot_width/PIXEL_NUMBER,
     PLOT_OFFSETY - 10)
  plot.create_text(PLOT_OFFSETX + x*250*plot_width/PIXEL_NUMBER,
     PLOT_OFFSETY - 20,font=("Arial Bold", 18),text=int(x*250),fill='black')

  plot.create_line(PLOT_OFFSETX + x*250*plot_width/PIXEL_NUMBER,
     PLOT_OFFSETY, PLOT_OFFSETX + x*250*plot_width/PIXEL_NUMBER,
     PLOT_OFFSETY + plot_height, fill="#dddddd")

# y-axis marking with count number
for y in range(14):
  plot.create_line(PLOT_OFFSETX-10, PLOT_OFFSETY + plot_height -
     y*5000*plot_height/COUNT_MAX,
                   PLOT_OFFSETX, PLOT_OFFSETY + plot_height -
                      y*5000*plot_height/COUNT_MAX)
  text_output = str(y*5) + "k"
  plot.create_text(PLOT_OFFSETX-40, plot_height + PLOT_OFFSETY -
     y*5000*plot_height/COUNT_MAX,font=("Arial Bold",
     18),text=text_output,fill='black')

  plot.create_line(PLOT_OFFSETX + plot_width, PLOT_OFFSETY + plot_height -
     y*5000*plot_height/COUNT_MAX,
                   PLOT_OFFSETX, PLOT_OFFSETY + plot_height -
                      y*5000*plot_height/COUNT_MAX, fill="#dddddd")

spectrometer_offset = 0.0
spectrometer_resolution = 0.0

# check if there is a cofiguration file in the same directory
if os.path.exists("EasyCalibrate_config.txt"):
  # reading configuration file, values indicated here will overwrite the
     default values
  file = open("EasyCalibrate_config.txt", "r")
  for line in file:
    config_variable = line.split()[0]
    config_value = line.split()[1]
    # print(str(config_variable) + " = " + str(config_value))
    # if config_variable == "spectrometer_name":
    #  spectrometer_name = config_value
    if config_variable == "spectrometer_offset":
      spectrometer_offset = float(config_value)
    if config_variable == "spectrometer_resolution":
      spectrometer_resolution = float(config_value)

  file.close()
```

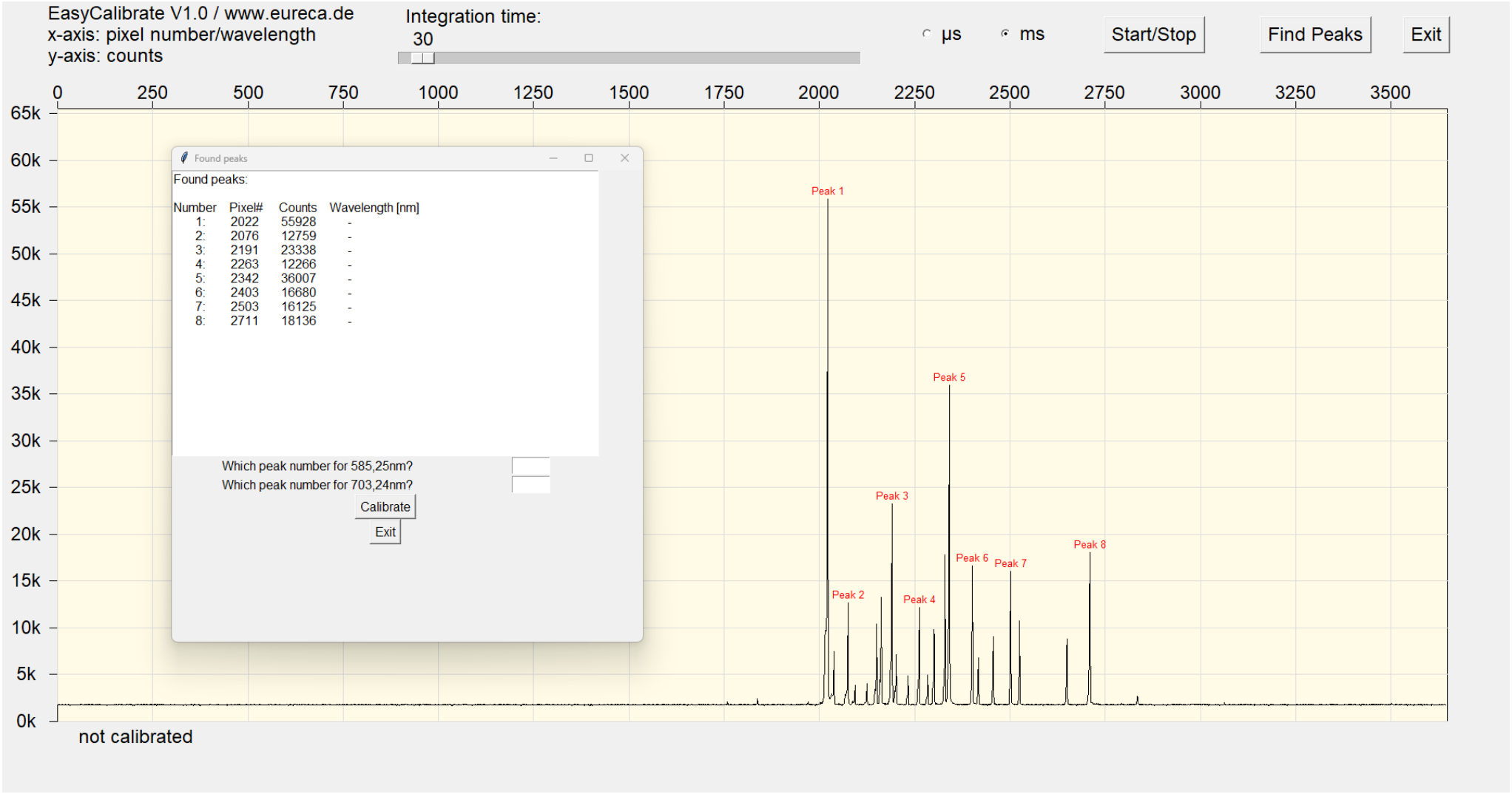
```
  # print the sensor wavelength at the lower side of the plot window
  Plot_Sensordata_Wavelength()
else:
  plot.create_text(PLOT_OFFSETX + 100, plot_height + PLOT_OFFSETY + 20,
     font=("Arial Bold", 18),text="not calibrated",fill='black',
     tags="wavelength")

def update_plot():
  if not master.winfo_exists():
    return

  if is_reading:
    # getting the integration time from the slider and using this value for
       the exposure time as well as for the frame time
    exposure_time = int(slider_exp_time.get()) * faktor.get()
    frame_time = exposure_time;

    # reading out the camera; for details refer to the EasyAccess
       documentation
    libe9u.e9u_LSMD_set_times_us (0, exposure_time, frame_time)
    libe9u.e9u_LSMD_get_next_frame (0)

    x_old = 0
    y_old = 0

    # deleting the old data points
    plot.delete("data")

    for pixel in range(0, PIXEL_NUMBER):

        counts = pointer[pixel]

        # scaling the sensor data to fit into the plot region
        x = int(pixel / PIXEL_NUMBER * plot_width)
        y = int(counts / COUNT_MAX * plot_height)

        # plotting the data point via connecting the current data with the
           last one
        plot.create_line(x_old + PLOT_OFFSETX,plot_height + PLOT_OFFSETY -
           y_old,x + PLOT_OFFSETX,plot_height + PLOT_OFFSETY - y,
           tags="data")

        x_old = x
        y_old = y

    plot.update()

  master.after(1, update_plot)

update_plot()

master.mainloop()
```


EURECA Messtechnik GmbH
Deutz-Kalker Str. 35
50679 Cologne, Germany

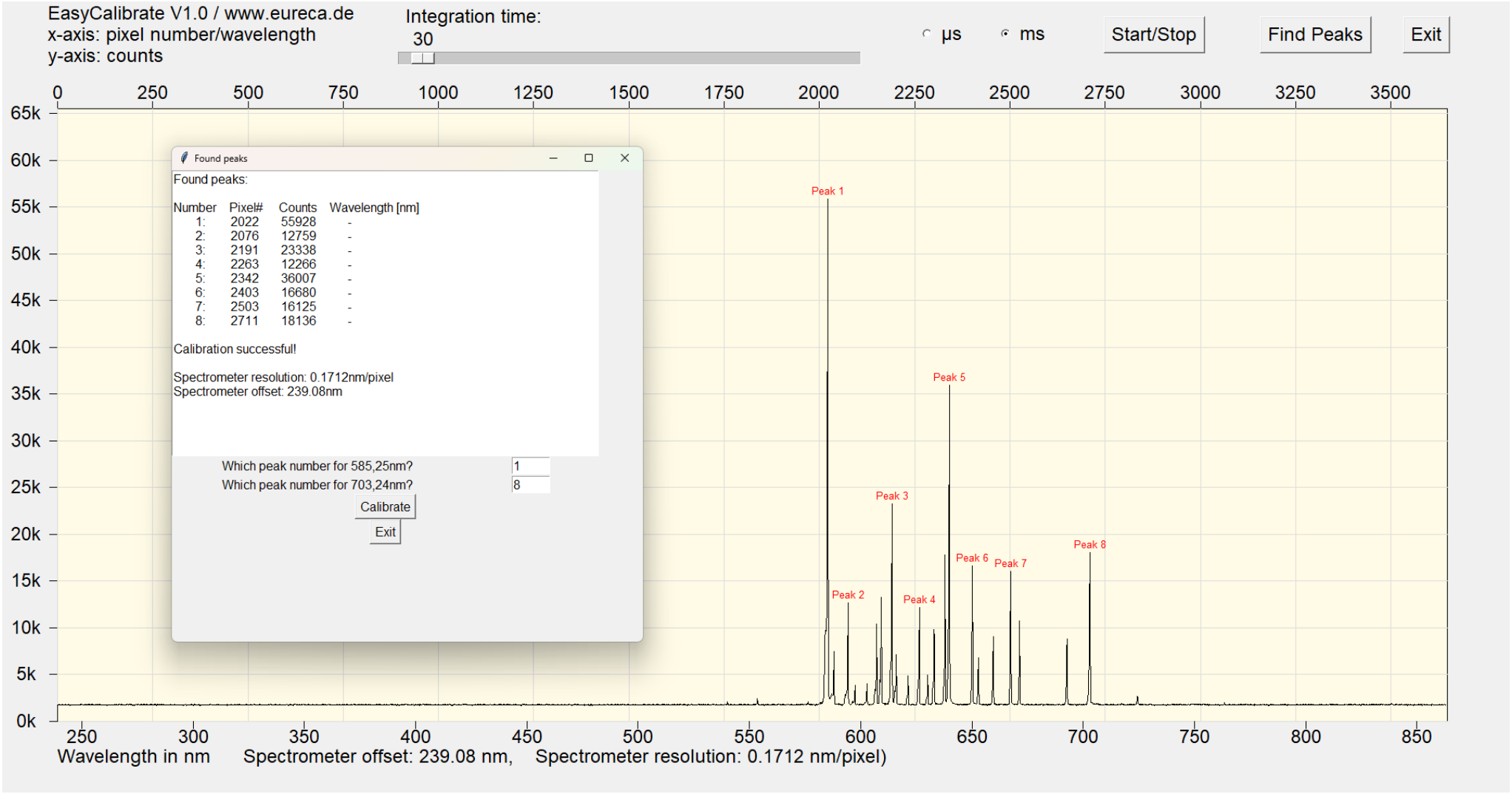
Tel: +49 (0) 221 / 952629 - 0
info@eureca.de
www.eureca.de



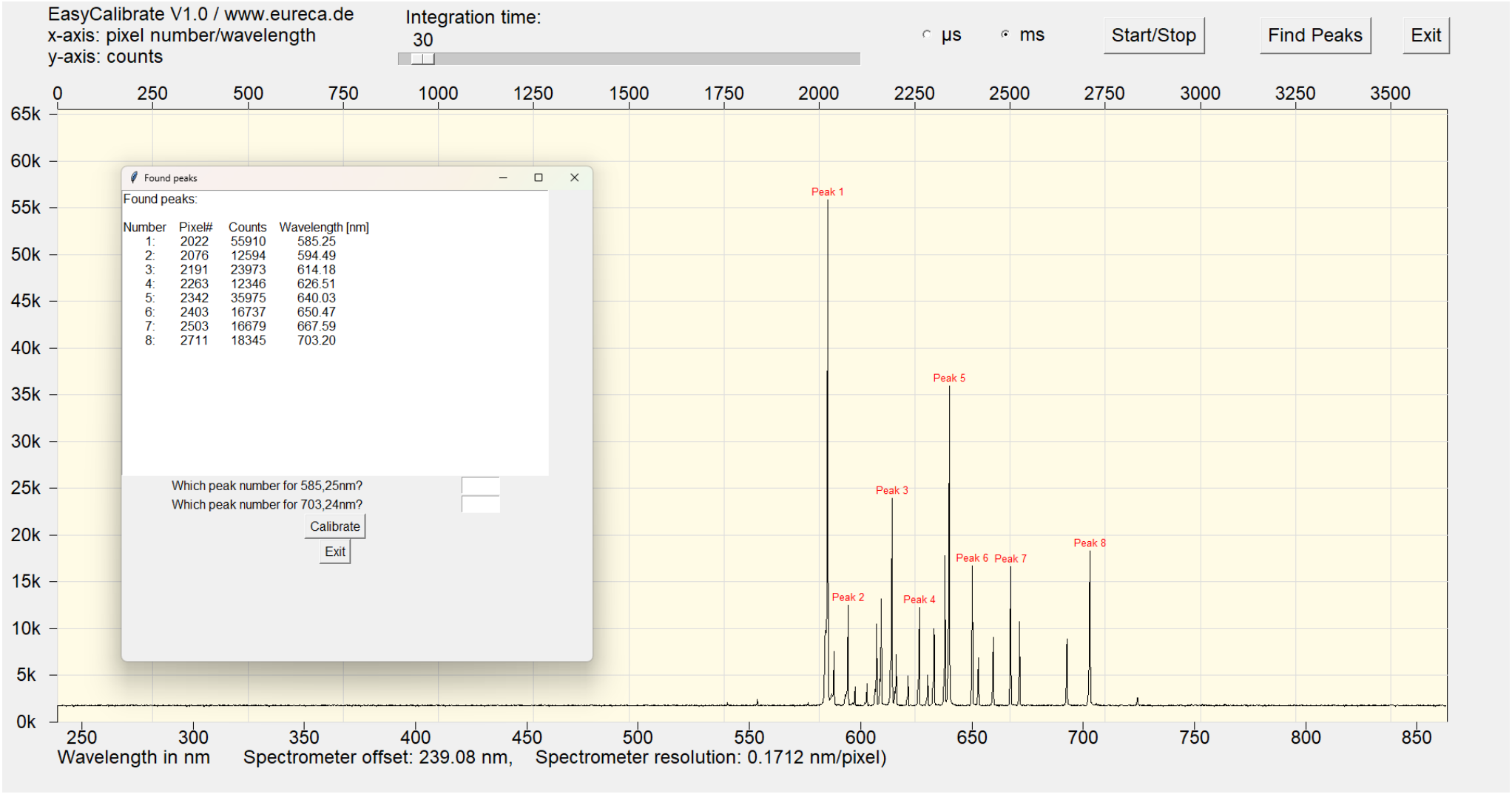
1: Spektrum einer Glimmlampe, mit unkalibriertem Spektrometer



2: Spektrum mit gefundenen und nummerierten Peaks



3: Kalibriertes Spektrum



4: Liste der gefundenen Peaks mit kalkulierten Wellenlängen