3 years public engineering school

Internship Report

AUTOMATION OF OPTICAL SETUPS USING PYTHON

31st May 2023,

Arthur Kramm,
2nd year student
arthur.kramm@ecole.ensicaen.fr

home supervisor: Philippe LEFEBVRE
host supervisor: Richard HILDNER
lab supervisor: Erik WOERING

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

university of groningen

TABLE OF CONTENT

TABLE OF FIGURES / ANNEXES

1. introduction

1.1. Acknowledgements

First, I would like to thank Professor Richard HILDNER who kindly accepted me in his research group during this internship providing me a great experience in a research environment. I would also like to thank Erik WOERING who mentored me during these 16 weeks and taught me a lot about the set-up I'd have to automate as he built it himself during his first doctorate year. He gave me a lot of advice and tips about the measurements and the pictures we are supposed to obtain. I also want to thank Foppe DE HAAN who helped me about many technical aspects as he is the research group technician. Moreover, I'd like to thank everyone in the research group as well as everyone in the other research group with who we shared the working environment. Another great help during my project was Maria DUMA, a master student who helped me by providing a lot of data she made during her experiments. I would also particularly like to thank PhD student Ioannis TOULOUPAS who helped me discover the city and the Dutch culture. I would also like to mention Anne-Marie CHASLE, another French student at the ENSICAEN with whom I spent some time discovering the country.

1.2. Host organisation

The University of Groningen or UG for short is an international and multidisciplinary university located in the city of Groningen. Groningen is the largest place as well as the economic and cultural center of the northern part of the Netherlands. UG, founded in 1614, is a public research university of more than 34,000 students, among them are more than 4,000 PhD students and around 8,000 international students. UG is ranked 75th in the Times Higher Education World University Ranking in 2023. Comparatively, Normandy University is ranked 801-1000th.

The department I was working in was the Zernike Institute for Advanced Materials or ZIAM for short. It is a state-of-the-art institute of the Groningen university. Founded in 1970, ZIAM is an interdisciplinary research institute that aims to develop new nanotechnologies and nanomaterials by connecting all the resources brought by the different groups. It consists of physicists, chemists, and biologists working together to bring their expertise to each other to improve the efficiency of the research groups.

The research group I joined is named Optical Spectroscopy of Functional Nano systems or OSFN for short, it is headed by Professor Richard HILDNER. The research in this group focuses on organic functional materials and supramolecular (nano)systems that are relevant for nanophotonic or molecular electronic applications. Extensively, the researchers in the group want to learn about the optical and electronic properties of these materials. This group aims at understanding how energy flows through such system, how light propagates within nanostructures and how energy flow and / or light propagation could be actively controlled.

1.3. Setup introduction

The main set-up I have been working on during my internship could be described as performing spectroscopy on single molecules at different temperatures, ranging from room temperature to 77 K. This experiment used to be operated manually and was a very repetitive work that could take a few hours with human actions. My goal was to automate it to take multiple pictures while not needing an operator to be present in the room.

1.3.1. Physics / Principle

First, it's important to define what single molecules are and why it is interesting to do spectroscopy on. The goal of single molecule measurement is to obtain the spectrum of a single polymer instead of a cluster of molecules. This way we will study the precise state a molecule resides in, instead of measuring the average of thousands of molecules stacked on top of each other, all with random positions and different interactions.

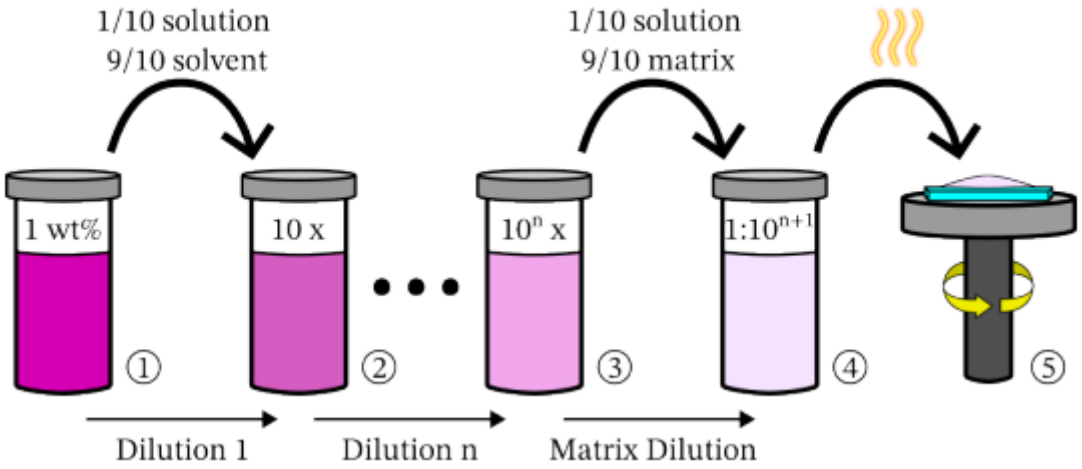


FIGURE 1 SAMPLE PREPARATION (CREDIT ERIK WOERING)

Single molecule samples are obtained by diluting a solution of the optical active polymer by a dilution factor of around 10^4 by successive 1/10 dilutions (steps 1 through 3 in figure 1) in what is considered a good solvent, what means he dissolves well without. It is then diluted with solution containing a polymer matrix instead of a standard solvent clustering (step 3 to 4 in figure 1). The polymer matrix will separate the optically active polymer strands from one another to avoid any interactions between them. The sample is then often centrifuged or heated up to remove/evaporate most of the solvents.

The core of this setup is a cryostat: a vacuum insulated sample chamber which uses a liquid cryogen as a coolant, here liquid nitrogen (~77 Kelvin). The sample we were studying was cooled down for two main reasons.

- limiting photobleaching due to the laser damaging the sample, i.e., preventing the molecules from "burning out" (undergoing a photo-induced chemical reaction).

- enabling the sample to be studied in a more "inert" state, without heat variations, allowing us to observe the molecule's state even more precisely.

The original setup looks like this:

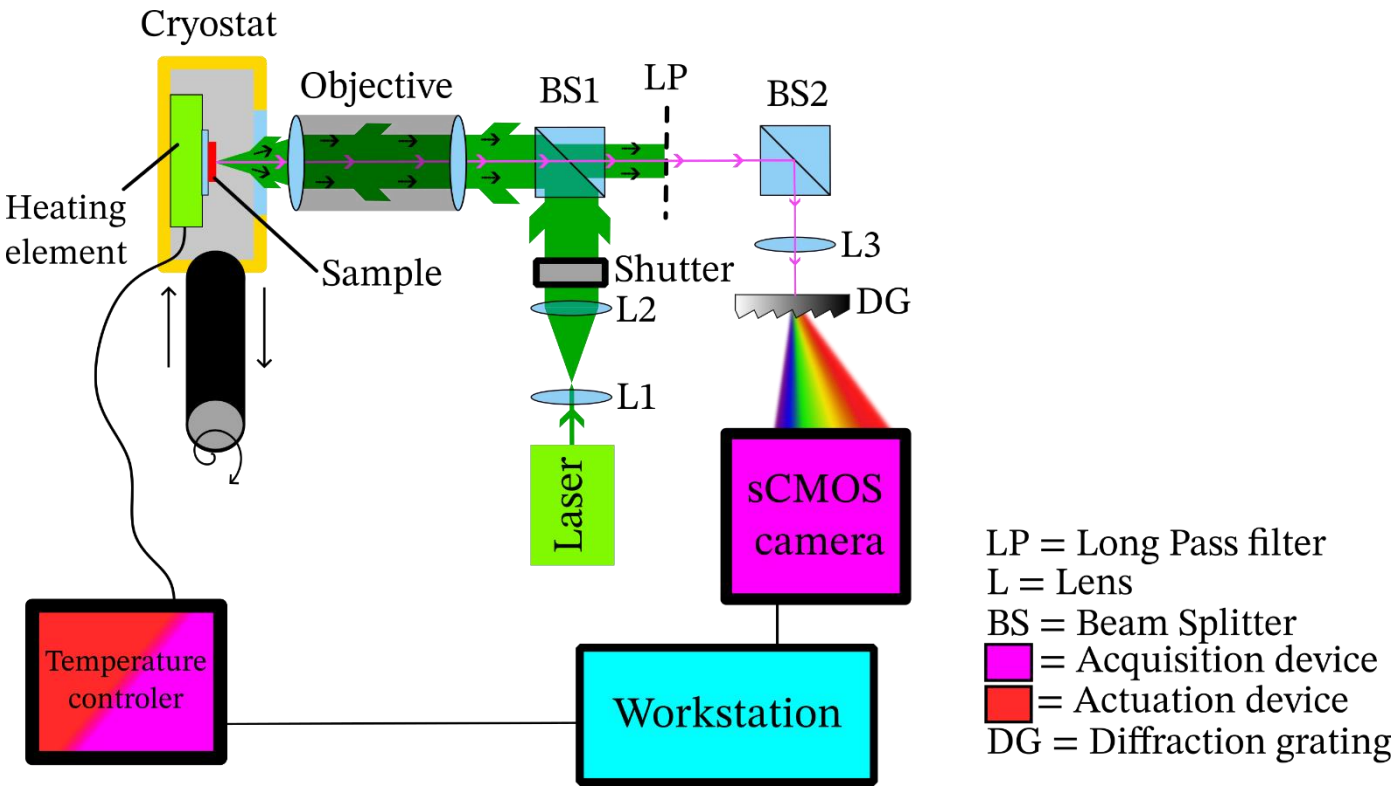


FIGURE 2 SINGLE MOLECULE CRYO SETUP WITHOUT AUTOMATION

The way the spectroscopy works on this setup is the following:

a graphical display of the original, fully manual setup is displayed in figure 2.

First, a laser beam is pointed at the sample we want to study. Because of the nature of the sample, the molecules on the sample will interfere with the photons hitting the sample, emitting other photons of lower wavelengths. This light coming from the sample is then filtered by a low-pass filter (LP on the figure), removing any light wavelength higher than a value just under the wavelength of the monochromatic laser. This way, the light that still contained photons from the laser has now these photons “filtered” which we did not want in our measurement. Finally, a diffraction grating (DG on the figure) will diffract the remaining light, that is, splitting the light based on its wavelength and creating a spectrum, just before it hits the camera.

It is important to mention that the element being moved by the operator was the cryostat, precisely moved by a micrometer screw. Moving the cryostat containing the sample would allow the laser to hit other molecules, present on other parts of the sample, making new measurement for every molecule detected. The screw will then be replaced by stepper motor to be part of the automation process.

The camera obtains a .tif image containing a spectrum, where the position of the pixels on the y axis of the picture correspond to different wavelengths and the brightness of the pixels correspond to the intensity. A background measurement will have to be done to subtract the background value of the image and a calibration picture with only the laser wavelength is required to calibrate the pixels on the x-axis to specific wavelength values. All these steps necessary to convert an image to an emission spectrum are done by using different Python scripts written by Erik Woering.

The set-up also contains numerous lenses (L1-L3 on the figure) whose purpose is to modify the laser beam to reach precise dimensions and positions. These lenses must be aligned every day before doing any measurement.

A manual shutter is also used in the set-up to prevent light from hitting the sample when no measurement is being made.

This setup contains various devices eligible for remote control:

- Thorlabs KST101, a stepper motor controller.
- Andor Zyla 4.2 Plus CMOS camera, a 2048x2048 16bits grayscale camera.
- TCSPC PicoHarp 300 USB, a picosecond event timer from PicoQuant (which was implemented during the automation of the original setup)
- 5V controlled shutter controlled by a USB DAQ from National Instruments.
- Cryostat from Oxford Instruments with controlled temperature by a mercury ITC

2. Automation

The heart of the set-up to be automated was the actuator (stepper motor) and the camera. With these 2 elements being able to be remotely controlled, most measurement could already be done that's why I spent most of the project working on a way to control them.

The main script "automation_routine.py" is available at the end of this report as figure 5 and 6 in the annex.

The final setup after automation and with the additional pieces of equipment looks like this:

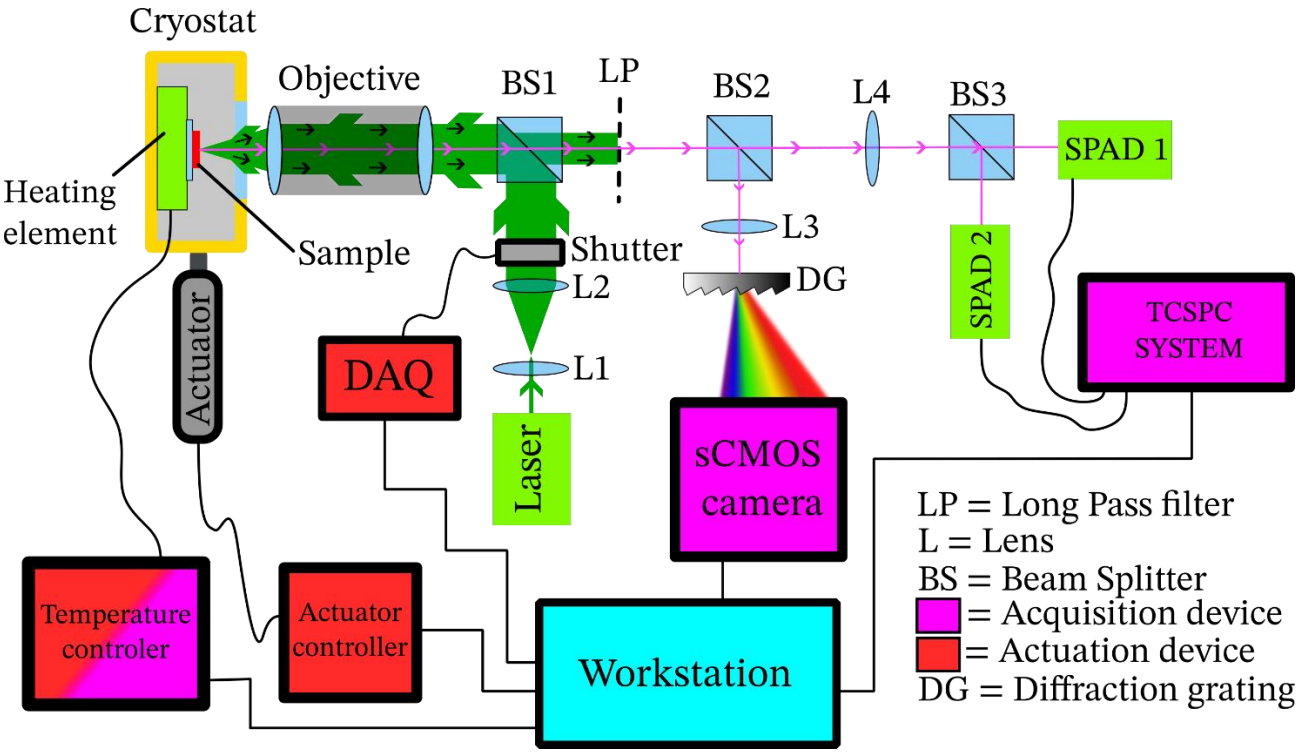


FIGURE 3 SINGLE MOLECULE CRYO SETUP AFTER AUTOMATION

2.1. Thorlabs actuator controller and Andor camera

To be able to move the cryostat before every measurement, the first device I started working on to automate was the Thorlabs KST101, a stepper controller coupled with the ZST213B, a 13mm travel stepper motorized actuator. Initially, the idea was to use YAQ, an interesting project for communicating with instrumental hardware using daemons, which are processes which run in the background of an application. This framework abstracts away the low-level hardware communication and allows to use similar syntax for similar functionalities hardware. Although YAQ contained a daemon exclusively for the KST101, I did not end up using this framework because the project I was working on was not using many different devices simultaneously, so the use of daemons would have been excessive. The use of background processes could have also caused issues as the different actions in the set-up had to be made in a sequential order, so problems like the stepper actuator moving with the camera capturing images simultaneously would have been likely to happen.

After, we came to the idea of using “PyLabLib”, a Python package for device control and experimental automation that is compatible with various devices for optical measurements. Besides the actuator, it is also compatible with the camera that is used in the lab, the Andor Zyla 4.2 Plus. I wrote a script called “pylablib_classes.py” which contains the wrapping of many functions from PyLabLib that were unclear or had single actions that was taking multiple lines of code which could be hard to read for the user. This way the user only has access to the main routine script that contains simple and self-explanatory functions and variables. It also contains default parameters for some functions so the user doesn’t have to always input the parameters which barely change (for example, the Andor Zyla camera is always held at -10°C).

All the functions and variables regarding the Andor Zyla camera are contained in the “AndorZyla” class and all the functions and variable regarding the stepper motor are contained in the “KST101” class. This way every hardware equipment can be used independently. This method of using classes is nice because it’s highly scalable. For example, with the addition of a second KST-101 and stepper motor as it will be discussed in the conclusion, the user would only need to create a new instance of the KST101 class.

The script “pylablib_classes.py” script is available at the end of this report as figure 7 in the annex.

2.2. Mercury ITC and the cryostat

To be able to control and get information from the Mercury ITC (the temperature controller), I used a hardware wrapper written by my supervisor, Erik. It contains functions to be able to communicate remotely with the ITC by sending requests serial communication requests, for example to obtain the temperature of the cryostat. The ITC is only able to monitor the temperature and heat up the cryostat: together with a constant flow of cryogen cooling down the sample, the ITC can keep a constant temperature using a proportional-integrable-derivative (PID) controller, for which we only used thew. The PID control is not altered from its industry values. After the measurements, a script is used to heat up the cryostat to room temperature quicker. This way, the whole experiment takes less time. I also implemented a logging of the temperatures as they could be useful to the user during the processing of the obtained data.

2.3. Shutter and DAQ

As the sample is still vulnerable to photobleaching (even at lower temperatures), it is imperative there is no light shining on the sample if no measurement is taken. This includes the moment the actuator is moving the cryostat. Hence, we needed to automate the shutter to block the laser light from shining onto the sample and open just at times of measurements. To control the 5 Volts remote controlled shutter, I had to use a DAQ (Data AcQuisition) device, a piece of hardware allowing for management of digital and analog Input/Outputs. The one I had at my disposal was a simple DAQ from National instrument, the NIDAQ USB-6000. Like for the other devices, a Python script was set up containing functions allowing for communication for opening and closing the shutter. This DAQ was working with what they call “tasks”. To clarify for the user, I also wrote wrapping functions inside the pylablib_classes.py script. All the functions and variables regarding the shutter are contained in the “Shutter” class. Besides the set-up I was working on, I was also able to integrate this module in another optical set-up of the research group.

2.4. Logging, features

Crucially, a logging system in my scripts was added using the logging python library. This will allow to keep track of all the actions performed by the script, as well as saving the parameters with which the measurements were made, such as stepper motor length increments, camera's exposure time, etc. This way, all information about the measurement would be saved for later use during the data analysis, a very important aspect which was requested by those working on the setup. The creation of all the necessary folders will be made at the beginning of the script if it doesn't already exist in the current working directory.

Additionnal pieces of equipment installed.

Besides spectra, we integrated novel devices for so called time-resolved measurements, besides obtaining spectra. Implementing the PicoHarp 300, a picosecond event timer used together with 2 Single-Photon Avalanche Diodes (SPADs) from Micro Photon Devices. The combination of these devices allows for 2 different types of information: first, we can measure the count rate for one SPAD, obtaining information about the number of photons going through our setup and simultaneously get more information of the duration of molecule-photon interaction in our sample. Initially, the idea was to use this information to locate points on the sample from which we obtain the highest intensity and create a spectrum from this position. However, as the count number variations would be too low, it was deemed unsuitable to use for molecule detection.

The other type of measurement the picosecond event timer would allow us to capture is the cross-correlation of both SPADs. This would allow us to determine the number of photons emitted simultaneously by the sample are made, which can further indicate our measurements are performed on single molecules. We have successfully integrated this method into the setup, but for the same reason as the previous measurement, we have not been able to see a measurement in action.

3. Image processing

As the setup is now automated in such a way that it takes measurements regardless of the incoming data, the next step in my work was to make a script which filters out unusable pictures. If the noise was low enough compared to the region of interest, the image would be copied in a new folder with all the other satisfactory images. Other parameters such as standard deviation, quantile values of pixel matrixes as well as image sharpness would be considered.

All these matrixes' manipulations and calculations will be made using NumPy arrays. Multiple OpenCV functions will be used for noise reduction and other applications.

Reasons for an image to be unusable could be that the sample became out of focus after moving too much in the perpendicular direction, or the sample had photo-bleached). In both cases, my image filtering script should remove these images from the usable pictures folder.

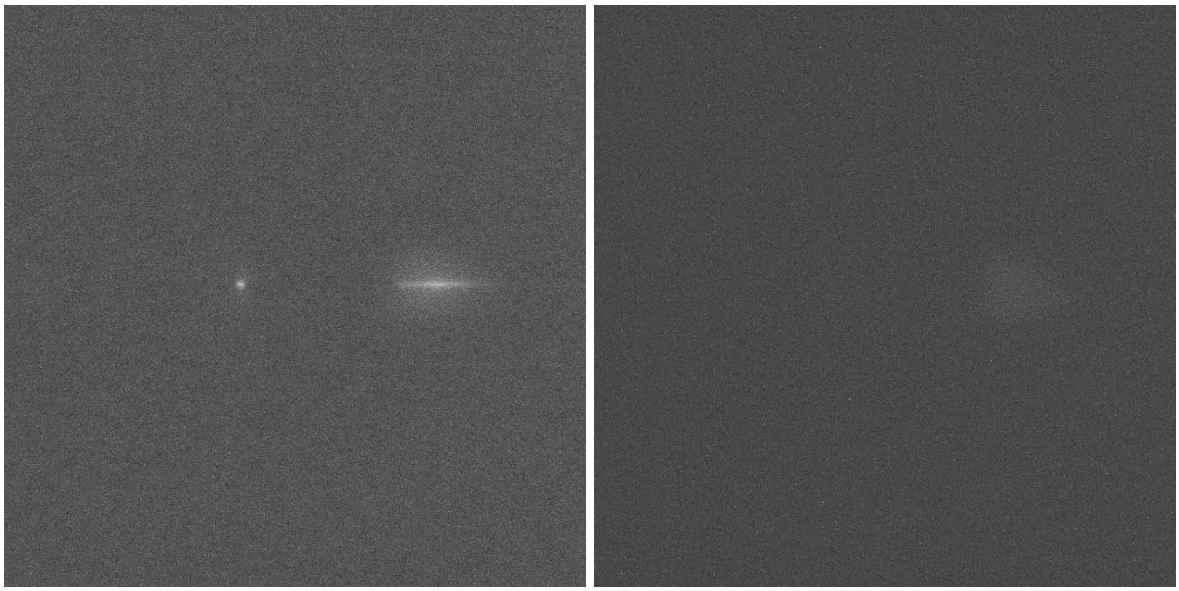
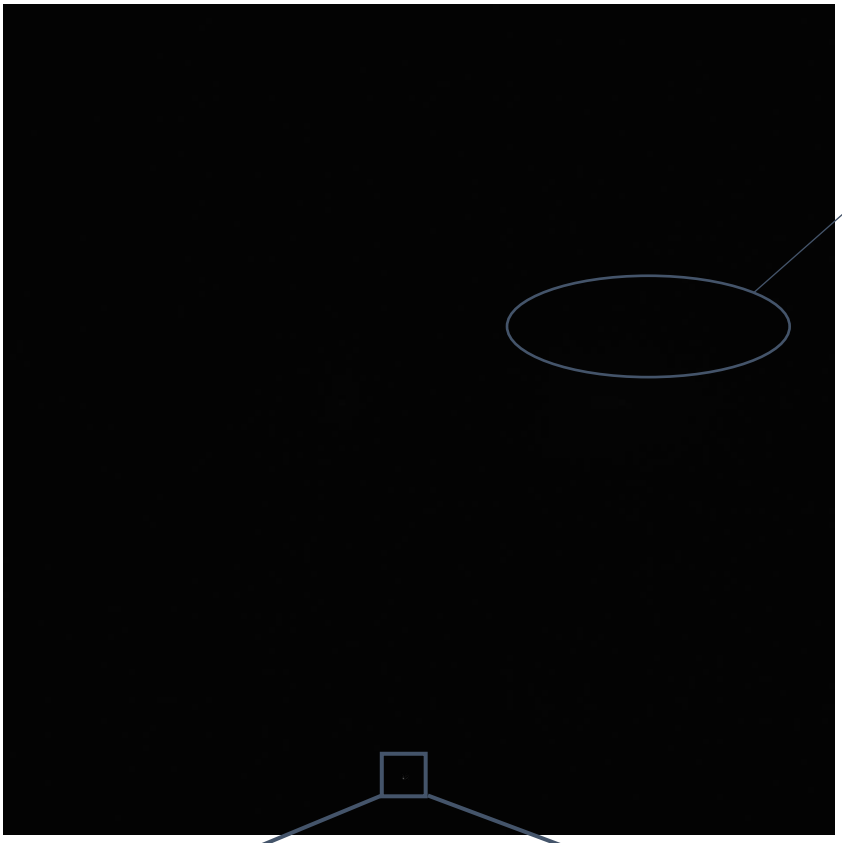


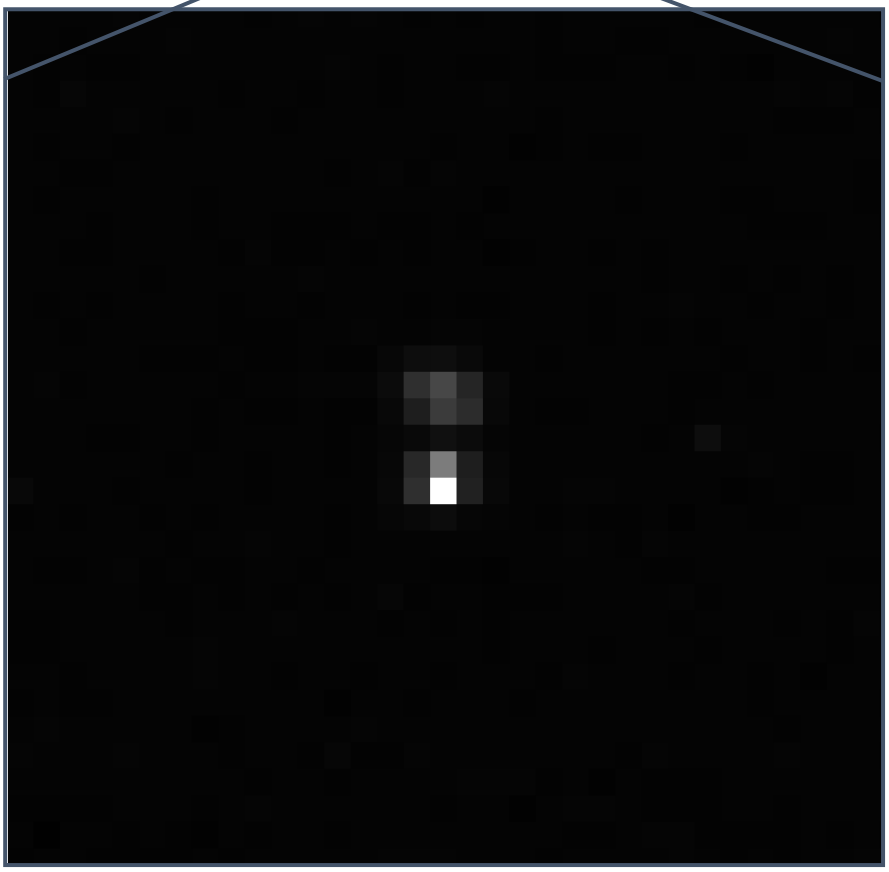
FIGURE 4 ON THE LEFT A USABLE PICTURE AND, ON THE RIGHT, AN UNUSABLE PICTURE

The issue I encountered was that some picture somehow had very bright value, making the whole picture look very dark even though there were still some data in the picture. The reason for such high values was discussed to be a result of cosmic rays which are high energy particle from space that occasionally hit the camera. We can also sometimes see this type of noise on the exact same pixel on none-consecutive pictures which results from dead hot pixels, which is more of a hardware issue/damage that comes from the camera aging or a manufacturing defect.

To resolve this issue, I implemented in my script a noise removal algorithm that changes all pixels with a value higher than a certain quantile of the image values to the average value of adjacent pixels (window size of the averaging can be changed).



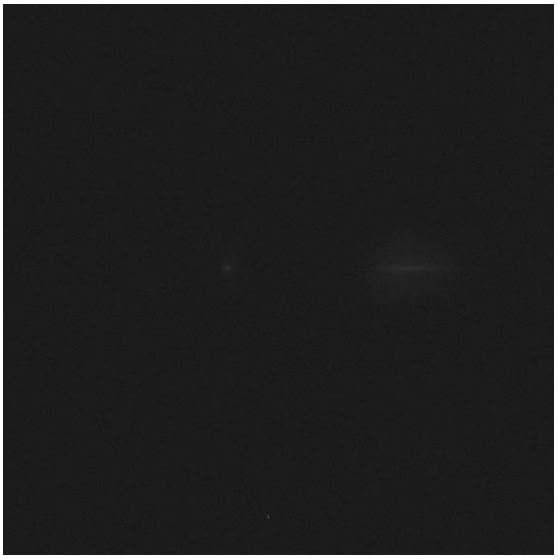
We can still slightly notice the presence of values we could use there (intensity of around 102-104). While the background values are around 98-101



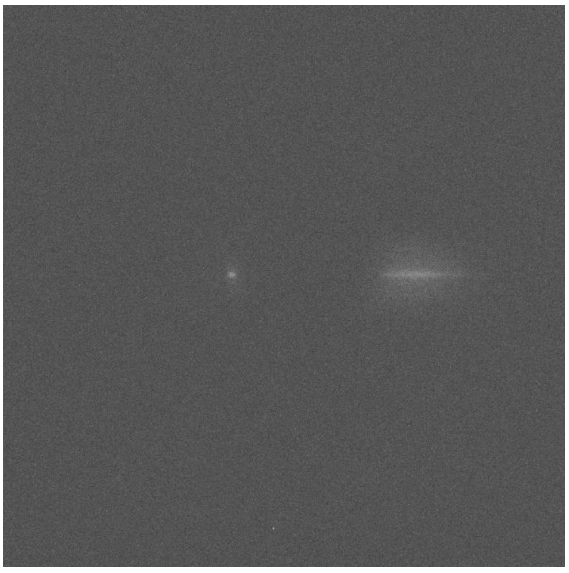
The brightest pixel value here is about 450 while the signal we are trying to measure is at about 102-105

The image would look like this in ImageJ, an image processing software able to open 16bits greyscale images.

After one cycle of the noise removal algorithm present in my python script, the image looks like this:



The usable data is becoming more and more visible. And finally, after 2 more processes through the algorithm loop, the image looks like this:



We can now clearly see that the region of interest of our picture is way clearer and brighter than before.

The bottom noisy pixels are only now at a value of 106 so they do not interfere with the usable data anymore.

The advantage of only doing a median blur on the very high pixel values instead of doing a median blur on the whole picture is that no usable data will be modified, while still improving the picture readability.

An example script of image processing is available at the end of this report as figures 8 and 9 in the annex.

4. Bibliography

https://yaq.fyi/introduction/

https://pylablib.readthedocs.io/en/latest/

https://nidaqmx-python.readthedocs.io/en/latest/

https://www.thorlabs.com/thorproduct.cfm?partnumber=KST101

https://www.picoquant.com/products/category/tcspc-and-time-tagging-modules/picoharp-300-stand-alone-tcspc-module-with-usb-interface

https://github.com/bennomeier/MercuryITC

5. Conclusion

To conclude, I've managed to fulfill my original goals which was the automation of the spectroscopy set-up, allowing the control and communication with different pieces of equipment present in the set-up. I'm satisfied with the work I've done and the scripts I wrote. I've been confirmed that some of them will keep being used in the lab by future students. My supervisor, Erik, even estimated that some measurement that used to take a week to be obtained can now be made in a day, so it feels very rewarding to know that my work was useful and not some sort of artificial work as I heard that this sometimes happens during internships. After this main work was considered done, I started working on side tasks such as the image processing of obtained data as well as implementing on other existing setups some functionalities I've built while working on the main setup. This whole internship was a great experience and taught me a lot about various aspects of work as well as life in general, discovering a new culture, living and working with a lot of internationals, therefore even learning about other cultures than the Dutch one. I've discovered the interesting and fulfilling workplace that is a university research group. I would highly recommend this country and more specifically this research group for other students as this project could be highly improved in terms of features and types of measurements: - temperature dependent measurements, - automation of the cryostat position in 2 dimensions, making automatic focus of the set-up on the sample, allowing to capture more spectra without user intervention (around 4-5 times more spectra), - incoming intensity dependent measurements.

6. Summary

6.1. Résumé de votre rapport en 10-15 lignes :

Le but de mon stage était d'automatiser un setup optique présent dans le laboratoire. J'ai pour cela réalisé environ 200 lignes de wrapper et déclarations de fonctions ainsi que 500 lignes de script principal d'automatisation. Un gros travail de documentation a été nécessaire sur les différents appareils utilisés dans le laboratoire afin de trouver les kits de développement nécessaires pour l'automatisation sans utiliser les logiciels constructeurs. Un autre travail m'a ensuite été confié, celui d'effectuer du traitement d'image sur l'importante quantité de données obtenue après fonctionnement du script d'automatisation. J'ai pour cela principalement utilisé OpenCV, une librairie de traitement d'image « open source ». J'ai également utilisé NumPy, librairie implémentant les tableaux en python. Permettant différentes opérations sur les images, pixel par pixel. De nombreux scripts seront écrit dans ce but pour un total d'environ 1000 lignes.

6.2. Summary of your report in 10-15 lines:

The main goal of my internship was the automation of an optical setup using python. For this goal I wrote around 200 lines of wrapper and functions declarations as well as around 500 lines of main script. A big work of documentation and research about the hardware and devices was necessary to learn about the different communication protocols as well as the development kits mandatory for use in external scripts instead of the proprietary software. Then, I was given another mission, which was to manage all the data (mainly pictures) that were created during an automated measurement. For this purpose, I mainly used OpenCV, an Open-source Computer Vision library. I also used NumPy, a library implementing arrays in python, allowing for pixels-by-pixels operations. Multiple image processing scripts were written for this purpose for a total of around 1000 lines.

7. Annexes

File - C:\Users\arthu\Desktop\stage\code\FINAL\Arthur\automation_routine.py

```python
import os
import time
from pylablib_wrapper import *
import cv2
import logging
from datetime import datetime


__name__ = 'automation_routine'
__version__ = 1.0
__author__ = 'Arthur Kramm, Erik Woering'

"""This script is used to automate the action of moving a stepper motor a certain distance multiple times,
while taking pictures everytime
(note that log files with parameters and execution traces are created local to the python script, while picture,
temperature logs and histograms are written in cwd)"""

debug = False    # debug mode will display more information in the console, display live feed,
# and the motor will go back to initial position after measurements

# saving the starting time of the program to monitor for how long it ran
start_time = time.time()


# # SETTINGS
# Current Working Directory
cwd = "D://Data//Erik//EW109//A//rt//Automatic"
#todo: check if exists before starting.


# Exposure Time of picture in s
exposure_time = 10


# # OD filters total value, will be part of the pictures names
# OD = 2.0
# if OD.is_integer():
#     OD = int(OD)



# Temperature of CCD in C
ccd_temp = -10

# Step size of motor in mm
step = 0.01


# Amount of measurements
measurement_amount = 50

########################################################################################################################

cam = AndorZyla(exposure_time, ccd_temp)
motor = KST101()
shutter = Shutter()

#motor.move_absolute(-1.700000)


# creating a new cwd folder if it does not exist already
if not os.path.exists(cwd):
    os.makedirs(cwd)
# creating a log folder where the script is located if it does not exist already
if not os.path.exists("logs"):
    os.makedirs("logs")


# creating a new 'scan' sub folder with a number that does not exist already
for j in range(100):
    subfolder_name = "scan" + str(j).zfill(2)
    cwd2 = cwd + '//' + subfolder_name
    if not os.path.exists(cwd2):
        # print(cwd2 + " does not exist already")
        break
    # print("file : " + cwd + "//Scan" + str(j).zfill(2) + " does already exist")
os.mkdir(cwd2)


now = datetime.now()
```

FIGURE 5 AUTOMATION ROUTINE SCRIPT PT.1

```
logging.basicConfig(
        filename=os.path.join('logs', 'automation_%s.log' % now.strftime('%Y-%m-%d_%H-%M-%S')),
        format='%(asctime)s %(message)s',
        filemode='w'
)
global logger
logger = logging.getLogger()

logger.warning("saving directory: "+cwd)
logger.warning("exposure time: "+str(exposure_time))
logger.warning("ccd camera temperature: "+str(ccd_temp))
logger.warning("Step size of motor in mm: "+str(step))
logger.warning("measurement amount: "+str(measurement_amount))
# logger.warning("OD filters total value: "+str(OD))

try:

    print('cam temperature (in °C):' + str(cam.get_temp()))

    logger.warning("camera starting to cool down")

    # we wait until the camera is .5 degrees over the target temperature because camera sensor has high uncertainty
    while cam.get_temp() > ccd_temp+0.5:
        print("cam temp still too high: " + str(cam.get_temp()))
        time.sleep(1)
    print("cam reached optimal temp")
    logger.warning("camera temperature (in °C): " + str(cam.get_temp()))

    if debug:
        cv2.namedWindow("image", cv2.WINDOW_NORMAL)
        cv2.resizeWindow("image", 512, 512)    # creating a window of certain size and position to display the images
        cv2.moveWindow("image", 100, 100)

    print('---CHECKLIST---')
    input('Check if saving location is correct: %s - press any key to continue.' % cwd)
    # input('Check if OD filters value is correct: %3f - press any key to continue.' % OD)
    # input('Check if the mercury ITC is in remote control mode - press any key to continue.')
    input('Check if Andor Solis is closed - press any key to continue.')
    input('Check if shutter is in automatic mode - press any key to start.')

    i = 0
    while i < measurement_amount:
        i += 1
        last_position = motor.position()
        motor.move_relative(step)   # initiate a move
        motor.wait()                # waits for the move to end
        if last_position == motor.position():  # if the motor is twice at the same position, break out of the loop
            break
        if debug:
            print('cam temp : ' + str(cam.get_temp()))

        img = cam.capture()  # takes a picture
        shutter.close()
        shutter.open()

        save_str = cwd2 + '//' + str(i).zfill(4) + subfolder_name + '.tif'
        print('[%4d/%4d] : saving file %s at position %9f mm' % (i, measurement_amount, save_str, motor.position()))
        logger.warning('%4d : saving file %s at position %9f mm'
                       % (i, save_str, motor.position()) + " camera temp: " + str(cam.get_temp()))
        cv2.imwrite(save_str, img)
        if debug:
            img[:, :] = (img[:, :] - img.min()) / (img.max() - img.min()) * 65535
            cv2.imshow("image", img)            # display the calibrated image if the debug mode is on
            cv2.waitKey(1)


except KeyboardInterrupt:
    if debug:
        cv2.destroyAllWindows()
    print('Interrupt key pressed, end of the program')
    print('disconnecting the motor...')
    motor.disconnect()
    print('disconnecting the camera...')
    cam.disconnect()
    print('disconnecting the shutter...')
    shutter.close()
    shutter.disconnect()
    #f.close()
    logger.warning('Script ended after %s seconds.' % (time.time() - start_time))
    exit(1)
```

FIGURE 6 AUTOMATION ROUTINE SCRIPT PT.2

File - C:\Users\arthu\PycharmProjects\pythonProject\pylablib_classes.py

```python
"""
Arthur KRAMM
05/2023
OSFN, Groningen University

Classes for using a thorlabs stepper motor together with an Andor camera and a 5V controlled shutter
"""

import sys
import pylablib
from pylablib.devices import Thorlabs
from pylablib.devices import Andor
import nidaqmx


class KST101:
    def __init__(self, sn="26005187", steps_per_mm=2000000):
        self.sn = sn
        self.steps_per_mm = steps_per_mm
        try:
            self.stage = Thorlabs.KinesisMotor(sn)  # connecting to the kst101 by specifying its serial number
            self.move_absolute(-1000)  # motor going to the lowest position possible
            self.wait()  # waiting for the motor to reach its low limit
            print("stepper motor initialized and reached position (mm): ", self.position())
        finally:
            sys.exit("ERROR: No Thorlabs stepper motor detected")

    def move_absolute(self, value=-1000):
        self.stage.move_to(value*self.steps_per_mm)

    def move_relative(self, value=0.01):
        self.stage.move_by(value*self.steps_per_mm)

    def wait(self):
        self.stage.wait_move()

    def position(self):
        return self.stage.get_position()/self.steps_per_mm

    def disconnect(self):
        self.move_absolute(-1000)  # motor going to the lowest position possible
        self.wait()  # waiting for the motor to reach its low limit
        print("stepper reached position (mm): ", self.position())


class AndorZyla:
    def __init__(self, exposure_time=0.1, temperature=-10):
        pylablib.par["devices/dlls/andor_sdk3"] = "dll"  # specifying location of andor .dll files
        self.exposure_time = exposure_time
        self.temperature = temperature
        if Andor.get_cameras_number_SDK3() == 1:  # if an andor camera is connected
            self.camera = Andor.AndorSDK3Camera()
            self.camera.set_attribute_value('ExposureTime', exposure_time)  # setting the exposure time
            self.camera.set_temperature(temperature)  # setting the temperature
        else:
            sys.exit("ERROR: No Andor camera detected")

    def capture(self):
        return self.camera.snap()

    def disconnect(self):
        self.camera.close()


class Shutter:
    def __init__(self, output_pin="Dev1/ao0"):
        try:
            self.task = nidaqmx.Task()
            self.task.ao_channels.add_ao_voltage_chan(output_pin)
        finally:
            sys.exit("ERROR: Not able to control the shutter")

    def open(self):
        self.task.write(5, auto_start=True)

    def close(self):
        self.task.write(0, auto_start=True)

    def disconnect(self):
        self.task.close()
```

FIGURE 7 CLASS WRAPPER FOR PYLABLIB FUNCTIONS

```python
from glob import glob
import os.path
import numpy as np
import cv2
import PIL.Image
import time
from tifffile import imwrite

"This script searches for all .tif pictures inside a folder (also searches inside all sub-folders) "
"and then procedes to multiple image processing operations such as noise reduction. " \
"All images that contain enough contrast between two quantile values will then be saved " \
"inside the destination folder with the same name they had in the working directory"

# source folder of images
cwd = "C://Users//arthu//Desktop//image_proc//MD4//1to10k//scan01"

# destination folder for images
destination_dir = "C://Users//arthu//Desktop//image_proc//output"

destination_dir2 = "C://Users//arthu//Desktop//image_proc//output2"

# put to true to display real time image processing in opencv windows
display = True

start_time = time.time()

if display:
    cv2.namedWindow("image", cv2.WINDOW_NORMAL)
    cv2.resizeWindow("image", 768, 768)
    cv2.moveWindow("image", 100, 100)

    cv2.namedWindow("image2", cv2.WINDOW_NORMAL)
    cv2.resizeWindow("image2", 768, 768)
    cv2.moveWindow("image2", 868, 100)

all_tif_file_paths = glob(os.path.join(cwd, "**", "*.tif"), recursive=True)
j = 0
nb_saved = 0
f = 0
for i in all_tif_file_paths:
    save_bool = False
    j += 1
    print("reading image", i)
    image = PIL.Image.open(i)
    im = np.array(image)

    dim = (2048, 2048)

    nb_noise = 1000  # number of high pixel value to remove

    med = np.median(im)
    threshold = np.quantile(im, 1 - nb_noise / (dim[0] * dim[1])) + 5
    WINDOW_SIZE = 7
    array_cut = np.zeros((2048 - WINDOW_SIZE + 1, 2048 - WINDOW_SIZE + 1))
    array_cut[:, :] = im[int((WINDOW_SIZE - 1) / 2):2048 - int((WINDOW_SIZE - 1) / 2),
                      int((WINDOW_SIZE - 1) / 2):2048 - int((WINDOW_SIZE - 1) / 2)]

    if display:
        im_display = np.copy(im)
        im_display[:, :] = (im[:, :] - im.min()) / (im.max() - im.min()) * 65535
        cv2.imshow("image", im_display)
        new_name2 = i[i.rindex('\\') + len('\\'):]
        imwrite(destination_dir2 + "//" + str(f) + "_" + new_name2, im)
        cv2.waitKey(10)

    while array_cut.max() > threshold:
        # print(array_cut.max(), maxi)
        print('at least one very bright pixel detected to be removed on this picture')

        f += 1
        for k in range(dim[0]):
            for m in range(dim[1]):
                if im[k, m] > threshold and (WINDOW_SIZE - 1) / 2 - 1 < k < dim[0] - (WINDOW_SIZE - 1) / 2 \
                        and (WINDOW_SIZE - 1) / 2 - 1 < m < dim[1] - (WINDOW_SIZE - 1) / 2:
                    # print('noisy pixel at', k, m)
                    tot = 0
                    for o in range(WINDOW_SIZE):
                        for p in range(WINDOW_SIZE):
                            if p != (WINDOW_SIZE - 1) / 2 or o != (WINDOW_SIZE - 1) / 2:
                                tot = tot + im[k + o - int((WINDOW_SIZE - 1) / 2), m + p - int((WINDOW_SIZE - 1) / 2)]
```

FIGURE 8 EXAMPLE OF IMAGE PROCESSING SCRIPT PT.1

```
                            # print(o-2, p-2, im[k+o-2, m+p-2])

                    # print("changing pixel's value from ", im[k, m], "to ", int(tot/24))
                    # print("changing pixel's value in ", k, m)

                    im[k, m] = int(tot / (WINDOW_SIZE * WINDOW_SIZE - 1))


        if display:
            im_display = np.copy(im)
            im_display[:, :] = (im[:, :] - im.min()) / (im.max() - im.min()) * 65535
            cv2.imshow("image", im_display)
            new_name2 = i[i.rindex('\\') + len('\\'):]
            imwrite(destination_dir2 + "//" + str(f) + "_" + new_name2, im)
            cv2.waitKey(10)
        array_cut[:, :] = im[int((WINDOW_SIZE - 1) / 2):2048 - int((WINDOW_SIZE - 1) / 2),
                             int((WINDOW_SIZE - 1) / 2):2048 - int((WINDOW_SIZE - 1) / 2)]
    im_orig = np.copy(im)

    # dimension with which image processing will be made for faster run time. Of course saved images keeps orig
quality
    dim = (512, 512)
    im = cv2.resize(im, dim, interpolation=cv2.INTER_AREA)
    im = im.astype("uint8")

    blurred_im = cv2.fastNlMeansDenoising(im, None, 7, 9, 3)
    # print(difference.max(), difference.min(), np.quantile(difference, 0.1), difference.mean())

    im3 = np.copy(blurred_im)
    # print(np.quantile(im3, 0.9999), np.quantile(im3, 0.999), np.quantile(im3, 0.99), np.quantile(im3, 0.9), n
quantile(im3, 0.5), np.quantile(im3, 0.1), np.quantile(im3, 0.01), np.quantile(im3, 0.001))
    if np.quantile(im3, 0.99999) - np.quantile(im3, 0.99) >= 2:
        save_bool = True
        nb_saved += 1

    # print(difference.min(), difference.max())
    # print(im3.min(), im3.max())
    if save_bool:
        print("contrast high enough, saving the file in the output dir")
        new_name = i[i.rindex('\\') + len('\\'):]
        imwrite(destination_dir + "//" + new_name, im_orig)

    if display:
        im_orig[:, :] = (im_orig[:, :] - im_orig.min()) / (im_orig.max() - im_orig.min()) * 65535
        # im2[:, :] = (im2[:, :] - im2.min()) / (im2.max() - im2.min()) * 255
        im3[:, :] = (im3[:, :] - im3.min()) / (im3.max() - im3.min()) * 255
        # hot_pixels[:, :] = (hot_pixels[:, :] - hot_pixels.min()) / (hot_pixels.max() - hot_pixels.min()) * 65
        # print(im.max(), im.min(), " - ", blurred_im.max(), blurred_im.min(), " = ", difference.max(), differe
(), difference.mean())
        # difference[:, :] = (difference[:, :] - difference.min()) / (difference.max() - difference.min()) * 25
        # blurred_im[:, :] = (blurred_im[:, :] - blurred_im.min()) / (blurred_im.max() - blurred_im.min()) * 25
        # im[:, :] = im[:, :] * 1
        # im3[:, :] = im3[:, :] * 1
        # print(im)
        # print(im3)
        # print('max: ', im.max(), im.min())
        cv2.imshow("image", im_orig)
        cv2.imshow("image2", im3)
        cv2.waitKey(1000)
        # print("number of denoized pixels: ", count)
        edges = cv2.Canny(im3, 100, 110)

        # Display the edges
        cv2.imshow('Edge Image', edges)
        cv2.waitKey(10)
print('\n', "saved", nb_saved, "pictures out of the", j, "pictures read", '\n\n')
elapsed_time = time.time() - start_time
mlsec = repr(elapsed_time).split('.')[1][:3]
print('Execution time:', time.strftime("%H:%M:%S.{} ".format(mlsec), time.gmtime(elapsed_time)) + " (h:m:s.ms)"
print('for an average time of:', "{:.2f}".format(elapsed_time / j), " seconds per image treated")

```

FIGURE 9 EXAMPLE OF IMAGE PROCESSING SCRIPT PT.2

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

university of groningen

Ecole Publique d'ingénieures et d'ingénieurs en 3 ans

6 boulevard Maréchal Juin, CS 45053
14050 CAEN cedex 04