DYNAMIC SYSTEM DEVELOPMENT FOR REAL-TIME LIGHT SPECTRA ACQUISITION FOR OPTICAL BIOSENSOR APPLICATIONS IN PYTHON

Christian Isaac Hurtado-Esquivel[1], Erika Maricruz Gallardo Pinal[1], Luis Alfonso Villa-Vargas[2], Miguel Ángel Alemán-Arce[2], Verónica Iraís Solís-Tinoco[2,3]*, Marco Antonio Ramírez-Salinas[2]

[1] Unidad Profesional Interdisciplinaria en Ingeniería y Tecnologías Avanzadas UPIITA-IPN. Ingeniería Biónica. Ciudad de México, C.P. 07340.

[2] Centro de Investigación en Computación del Instituto Politécnico Nacional, Laboratorio de Microtecnología y Sistemas Embebidos. Ciudad de México, 07738, México.

[3] Instituto de Ciencias Aplicadas y Tecnología, Universidad Nacional Autónoma de México, Ciudad de México, 04510, México.

* irais.solis@cic.ipn.mx

Abstract

This work presents the development of a dynamic real-time light spectrum acquisition software system utilizing Python, a high-level programming language, focusing on optical biosensor applications. This software provides a versatile tool for acquiring, processing, and visualizing white light spectra obtained by connecting to an Ocean Optics spectrometer.

Dynamic term denotes the system's ability to adjust acquisition parameters in real-time, both in software and hardware, in response to user inputs. This adaptability is achieved via thread-based concurrent programming, guaranteeing precision sampling time (T_S) of up to 99.98 %. By configuring a 500 ms period and an integration time equivalent to 10 ms, offset levels are minimized and noise in captured spectra is reduced: SNR reaches up to 38.54 dB, RMSE is minimized to 27.83, and a maximum R-Squared of 0.9998 is attained. To improve signal quality, a zero-phase second-order Butterworth filter is applied, effectively suppressing noise above the normalized 0.04 rad/sample cut-off. Each captured spectrum is presented by a graphical user interface, with optimized axis boundaries, facilitating real-time analysis. Thus, this work provides an accessible methodology for scientists and technicians to develop a real-time data acquisition program, free from limitations and technological dependencies associated with commercial devices.

Keywords: optical biosensors, python programming, data acquisition, threads, spectrometer

1. Introduction

The implementation of sensors enables to comprehend, monitor, and regulate the environment. Sensors convert physical phenomena into other physical quantities, mainly electrical signals for qualification and quantification purposes. Research advances have driven an interdisciplinary path to emerging biosensors [1, 2]. The Union of Pure and Applied Chemistry (IUPAC) defines a biosensor as a device that uses specific biochemical reactions to detect compounds usually by electrical, thermal, or optical signals [3]. Biosensors are analytical devices designed to interpret biological events mediated by two key components: biological or biomimetic compounds and transducers. The initial stage of a biosensor involves incorporating an element sensitive to an analyte, while the transducer is responsible for detecting the interaction between the analyte and the biorecognition component. When the biological recognition reaction occurs, a series of physicochemical changes takes place, which are detected by the transducer. The transducer transforms this stimulus into a quantifiable output signal [4]. Some transducers recognize the reactions through optical properties changes, facilitating direct, real-time, and label-free detection of biological and chemical substances. Electrical components that contain these transducers are classified as optical biosensors, constituting the central theme of discussion in this work [5].

In the realm of optical biosensors, light is the energy source used to interact with samples. When light falls on an analyte, a portion of it may be absorbed by the sample's constituents. Absorbance is a measure that quantifies the ability of a medium to absorb incident light, while transmittance refers to the medium capacity to transmit energy. Furthermore, some of the incident light may be reflected off the surface of the sample, a phenomenon known as reflectance. Reflectance measures the proportion of incident light that is redirected from the sample's surface. Optical sensors discern and quantify these changes in absorbance, transmission, and reflectance to analyze samples for the presence of certain components, such as biomolecules or specific chemical compounds [4]. The high specificity of these sensors, their sensitivity, compactness and cost-effectiveness enable applications spanning healthcare, environmental monitoring and biotechnology [5].

A spectrum represents the amplitudes of a system's wave components, and light can be seen as this. The measurement of interactions between light and matter, reactions and measurements of radiation intensity and wavelength, is called spectrometry. Spectrometers are scientific-technical field instruments that are used for the analysis of different forms of matter in the form of a signal [6]. Spectral data analysis is crucial for many fields, where accurate measurements are fundamental. The demand for real-time signal analysis requires faster execution of analytical operations to process all signal components within the designated frequency spectrum. Therefore, it is imperative that signal display occurs quickly and that all computational procedures persist uninterrupted, adapting to variations in the input signal [7].

Ocean Optics spectrometers provide various data acquisition methods, which have their own spectroscopic application. It is a software that has a graphical interface that provides fast and stable data acquisition and processing [8]. However, the license comes with a hefty price tag, making it unaffordable. Moreover, it includes additional advanced features that enable users to engage with the spectrometer data, incurring an additional expense. Then, this program has limitations and technological dependencies associated with commercial devices.

Another software that is capable of interfacing with these spectrometers is MATLAB. This is a platform that provides the Instrument Control Toolbox, which allows the connection with Ocean Optics spectrometers. A wide range of tasks can be carried out in MATLAB, such as acquiring a spectrum, adjusting integration time, applying dark current and non-linear spectral corrections, and viewing all connected devices [9]. Despite being a powerful tool, it is important to be aware of its drawbacks. One significant disadvantage is the high cost associated with it. Moreover, licenses need to be renewed annually, which can be quite expensive for businesses and organizations relying on MATLAB regularly.

Therefore, to overcome these challenges, it is essential to understand how to develop software that allows efficient and seamless access to spectrometer data. This would unlock new possibilities for researching and analyzing spectroscopic data, eliminating the need to purchase costly licenses or additional software packages.

On the other hand, Python is an open-source high-level programming language which has become indispensable in science and technology. Cross-platform portability, wide variety of packages and component integration are the main characteristics that allow the user to do multiple tasks successfully. Its versatility makes it popular in software development, due to the simplicity in syntax and robustness in functionality [10].

Another feature of Python is the fact that it is an interpreted language. This means that the Python interpreter alternately reads and performs the calculations for each of the lines of code. It then repeats this process if there are no errors. This allows to test language functionality while programming [11, 12]. Part of Python's technological development focuses on communication protocol implementation for several purposes, taking advantage of libraries such as PySerial and PyUsb. These protocols generate data exchange and communication between devices, which guarantees efficient operation and joint operability between platforms. Thus, Python represents aids for developers, who can create reliable and scalable solutions for their communication needs across different domains.

In the field of spectrophotometry, Python emerges as a functional tool for spectral analysis purposes, specifically, to interact with hardware devices like spectrometers: Python-SeaBreeze. That is why it is important to understand the underlying protocols in this context. Python-SeaBreeze plays a critical role, bridging the gap between Python's analytical capabilities and the complexities of spectrometer communication. SeaBreeze library facilitates communication with Ocean Optics spectrometers, offering two distinct backends: cseabreeze, which wraps the SeaBreeze library, and pyseabreeze, which uses PyUsb for communication [13]. Attributable to Python's flexibility and scalability, it is possible to develop software for the acquisition, processing, and visualization of spectrometer data. Developers can focus more on problem-solving and less on the complexities of programming languages, resulting in faster development cycles and more efficient software delivery. Due to the versatility of the language and its wide applications in science and engineering, libraries and built-in functions were used to process signals, plot, and control sub-processes [14].

To execute Python programs or *scripts*, a code editor, or an integrated development environment, called an IDE, is required [15]. Likewise, it is possible to use a *shell* to write and run Python code. A *shell* is a command interpreter, which means a program that translates commands entered by the user into instructions for accessing operating system services [16]. It is possible to access the Python Shell once the language has been installed. In this work, Windows PowerShell served as the primary tool for executing a variety of installation processes due to the operating system [17].

In addition, although Python can establish parallel processing, its application is limited depending on the computer architecture [18]. When a processor has only one core, it is not possible to perform multiple tasks simultaneously. On the other hand, for multi-core processors, it is possible to create several processes running in separate memory spaces and avoiding deadlocks; a concept known as parallel programming [19, 20]. A process represents a running program which requires resources allocated during its execution. Each process operates within its own memory space and execution context. The operating system manages the processes, enabling them to run simultaneously [14, 21]. This limitation poses problems in efficiently utilizing multiple processors or cores, particularly in scenarios where computational tasks need to be executed concurrently.

For this reason, to enhance compatibility with single-core processors, a multi-threaded task execution flow, referred to as threads, was introduced. One or more threads can be executed within a single process and Python provides a few modules to create and manipulate these threads. However, due to the Global Interpreter Lock (GIL) in CPython (the standard implementation of Python), only one thread can execute Python code at a time. This means that even if threads are used, only one thread can be running at any given time [22, 23].

However, developing spectral acquisition software using Python presents notable challenges. First, the spectrometer's quality directly affects the precision and reliability of the obtained data. Additionally, successful integration requires handling each device's specific communication protocols, which can be complex due to the variety of interfaces and standards used by manufacturers. Another consideration is the libraries and requirements necessary to interact with the spectrometer hardware. This involves installing and setting up additional dependencies, which can complicate the development process.

In this work, we present a step-by-step methodology for developing a Python program to facilitate the real-time acquisition and control of spectra from sources such as halogen light sources, standard flashlights, and LED light sources. The spectrometer utilized is the Ocean Optics Flame-T model, which is widely employed for acquiring and measuring electromagnetic spectra from optical sensors.

First, the acquisition system is explained. It starts by presenting the general model of the designed software and continues with the creation of the environment that allows access to the SeaBreeze library functions. This is a process that only needs to be performed once. Subsequently, the multithreaded processing is discussed to make the system modifiable in its acquisition parameters and asynchronously. Likewise, it is detailed how the recognition is performed with the device, the reading of the spectra, and the adjustment to the signals to increase their quality by attenuating the noise present. To conclude the section, the strategy for displaying the spectra on the screen and storing them in the computer is mentioned. Finally, the spectra obtained are presented and analyzed. In this work, a LED light source was used, and 10 spectra captured every 5 seconds were analyzed.

This Python program could be used for real-time detection of electromagnetic spectra generated by optical biosensors. The acquisition of spectrometer data can be applied in measuring optical properties such as transmittance, reflectance, or absorbance. Furthermore, acquiring data from the spectrometer in real-time enables the calculation of sensorgrams generated during biofunctionalization experiments of biosensors, which may last for minutes or even hours. This Python program can be used for detecting changes in optical properties with the help of specific optical biosensors, depending on the type of study.

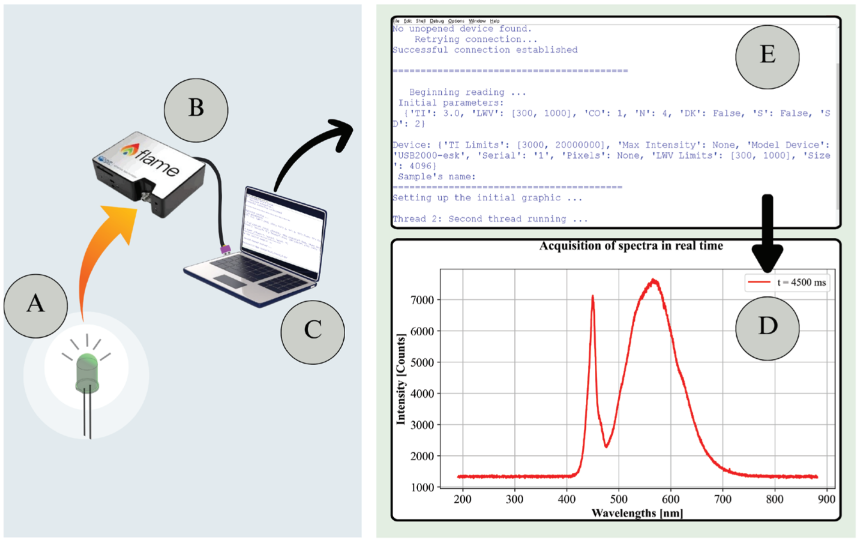
2. Methodology

This section elaborates on the design of the acquisition system. A representation of the connections used is depicted in Figure 1. Initially, an LED light source was positioned in proximity to the spectrometer detector (Figure 1A). Additionally, a Flame T spectrometer was employed (Figure 1B), connected to the computer via the USB communication protocol (Figure 1C). The dynamic real-time spectrum acquisition system was developed using the Python programming language. In this acquisition process, the utilization of a *shell* was imperative to establish communication with the software. It enabled user data input and provided information display regarding the various ongoing processes, as illustrated in Figure 1D. Ultimately, throughout the acquisition process, a printout of each spectrum was generated (Figure 1E).

For the development of this system, we utilized Python version 3.9.6 along with numerical data manipulation and visualization libraries such as *NumPy* and *Matplotlib*. Additionally, we incorporate the spectrum acquisition package *SeaBreeze*, as well as modules for time monitoring (*time*), thread management (*threading*), and interaction with the operating system to control program inputs and outputs (*sys*). The general block diagram of the proposed system is shown in Figure 2 and it is described in the following sections.

2.1. *SeaBreeze Execution Environment Set Up*

The SeaBreeze library facilitates communication between the spectrometer and the computer. Before using SeaBreeze functions, it is necessary to complete an installation process for dependencies and libraries. This configuration is a one-time process, establishing an environment capable of transmitting data, reading hardware parameters, and facilitating the development of new functions. Figure 3 illustrates the proposed sequence, with each sub-process detailed below.


B
flame
A
C
E
File Edit Shell Debug Options Window Help
Acquisition of spectra in real time
t = 4500 ms
D
Intensity [Counts]
Wavelengths [nm]

```
No unopened device found.
    Retrying connection...
Successful connection established

==========================================


   Beginning reading ...
 Initial parameters:
  {'TI': 3.0, 'LWV': [300, 1000], 'CO': 1, 'N': 4, 'DK': False, 'S': False, 'S
D': 2}

Device: {'TI Limits': [3000, 20000000], 'Max Intensity': None, 'Model Device':
'USB2000-esk', 'Serial': '1', 'Pixels': None, 'LWV Limits': [300, 1000], 'Size
': 4096}
 Sample's name:
==========================================
Setting up the initial graphic ...

Thread 2: Second thread running ...
```

Figure 1. Components of the proposed spectrum acquisition system: (A) representation of the LED light source, (B) example of Ocean Optics spectrometer, (C) computer compatible with the requirements of the proposed system detailed in section 2.2, (D) example of the history produced when running the system through a terminal, (E) example of wavelength distribution associated with the light source used; spectrum saved at instant 4.5 s after starting the run.

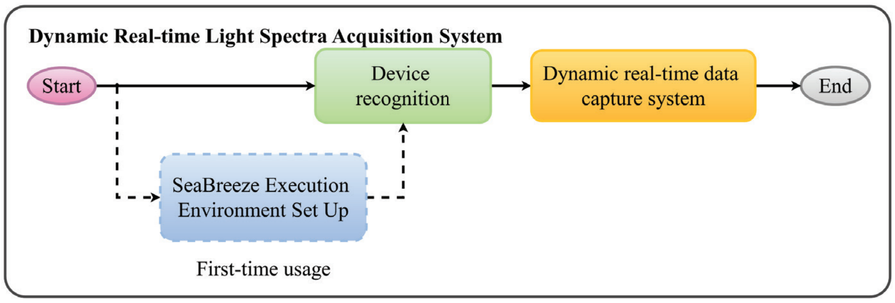

Dynamic Real-time Light Spectra Acquisition System
Start
Device recognition
Dynamic real-time data capture system
End
SeaBreeze Execution Environment Set Up
First-time usage

Figure 2. Block diagram illustrating the implementation and configuration of a dynamic real-time spectrum acquisition system.

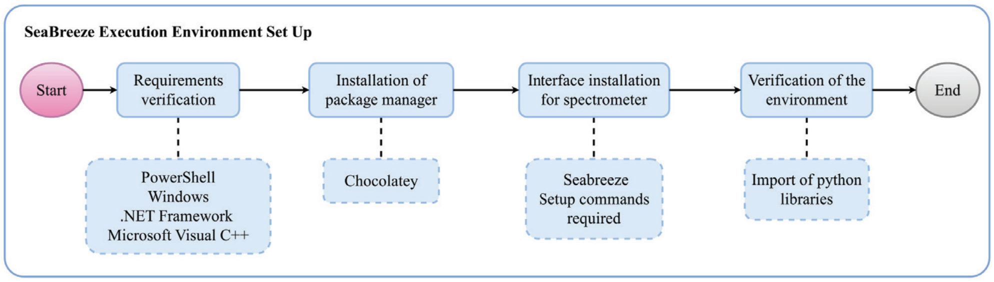

SeaBreeze Execution Environment Set Up
Start
Requirements verification
Installation of package manager
Interface installation for spectrometer
Verification of the environment
End
PowerShell Windows .NET Framework Microsoft Visual C++
Chocolatey
Seabreeze Setup commands required
Import of python libraries

Figure 3. Flowchart outlining the proper configuration of the environment required for utilizing the SeaBreeze library.

2.2. *Requirements Verification*

The initial requirements for establishing communication with the measurement hardware are summarized in Table 1.

Software	Version	More information
Windows	+8	-
Windows PowerShell	+3	PowerShell Installation
.NET Framework	+4.8	.NET Framework Download
Microsoft Visual C++	+4.8	Microsoft Visual Studio Download

Table 1. Software versions. Information revised in 2024.

To begin the installation process on Windows, a minimum of Windows 8 or Windows Server 2012 version is required. Operating systems are generated through PowerShell, enabling subsequent installations. The version of PowerShell must be reviewed, which, according to current technologies, should be equal to or greater than 3.

```
Windows PowerShell

PS C:\Windows\system32> $PSVersionTable.PSVersion

Major  Minor  Build  Revision
-----  -----  -----  --------
5      1      19041  4170
```

In this example, based on the Major and Minor attributes, the Windows PowerShell version is 5.1. Configuring the execution policy is necessary. This adjustment allows the execution of scripts, which are essentially programming code designed to automate certain processes, but only those scripts signed by a trusted publisher [24, 25].

The first step is to run PowerShell as an administrator and grant permissions to access as a superuser. Next, the security level is verified by executing the following line:

```
Windows PowerShell

PS C:\Windows\system32> Get-ExecutionPolicy
```

If the result in the terminal is Restricted, the level must be changed to *Allsigned*. The command to set up the execution policy is as follows:

```
Windows PowerShell

PS C:\Windows\system32> Set-ExecutionPolicy AllSigned
```

On non-Windows computers running PowerShell 6.0, the default execution policy is Unrestricted and cannot be changed [26]. The following prerequisite is .NET Framework version 4.8 or higher. This framework facilitates the creation and running of various applications, including desktop apps, web services, and more [27, 28]. Finally, Microsoft Visual C++ compilation tools are necessary with a version greater than or equal to 4.8. These tools are essential for compiling and debugging C++ code, which is particularly valuable for low-level device communication and hardware control. The SeaBreeze library primarily operates via C/C++, even though the interface is in Python, making these tools crucial for its functionality [29, 30].

2.3. *Package Manager Installation*

Once the requirements have been fulfilled, the *Chocolatey* package manager was chosen to facilitate the installation process of the SeaBreeze library, following [13, 31] recommendation. The following batch of commands is executed in Windows PowerShell.

```
Windows PowerShell
PS C:\Windows\system32> Set-ExecutionPolicy Bypass -Scope Process -Force;

    PS C:\Windows\system32> [System.Net.ServicePointManager]::SecurityProtocol =
  [System.Net.ServicePointManager]::SecurityProtocol -bor 3072;

    PS    C:\Windows\system32>    iex    ((New-Object    System.Net.WebClient).
  DownloadString(‘https://chocolatey.org/install.ps1’));
```

Note: using semicolon *(;)* at the end of each command is a method used to distinguish the beginning and end of long instructions. To verify the correct package manager installation, the next command is used:

```
Windows PowerShell
PS C:\Windows\system32> choco
Chocolatey v2.2.2
```

This example displays *Chocolatey* version 2.2.2 installed.

2.4. *Spectrometer Interface Installation*

After installing *Chocolatey*, basic operation requirements are obtained by executing the following code in the terminal.

```
Windows PowerShell
PS C:\Windows\system32> choco install visualcpp-build-tools
PS C:\Windows\system32> choco install windows-sdk-10.1
PS C:\Windows\system32> choco install windowsdriverkit10
```

These three packages are essential for compiling and executing SeaBreeze code, facilitating hardware manipulation using Python commands executed in C++. They collectively streamline the compilation and execution processes. Table 2 provides further details on these commands.

Commands	**Actions**
choco install visualcpp-build-tools	Installs Microsoft Visual C++ compilation tools.
choco install windows-sdk-10.1	Installs Windows 10.1 Software Development Kit (SDK) for building applications.
choco install windowsdriverkit10	Installs Windows 10 Driver Development Kit (WDK), which includes building and deploying driver tools (programs that support hardware-operating system interaction), [13].

Table 2. Requirements installation using Chocolatey.

The subsequent task is SeaBreeze library installation using the commands below:

```
   Windows PowerShell
   PS  C:\Windows\system32>  git  clone  https://github.com/ap--/
  python-seabreeze.git python-seabreeze;
PS C:\Windows\system32> cd python-seabreeze;
And then,
PS C:\Windows\system32> python -m pip install .;
Or
   PS C:\Windows\system32> python -m pip install --no-binary :all:
  seabreeze;
```

Each line of code is described below, including the actions it performs.

- **Copy a directory:** The command *git clone https://github.com/ap--/python-seabreeze.git python-seabreeze*, makes a copy of the GitHub repository (according to the specified path) into a new directory on the computer named "python-seabreeze".
- **Change directory:** The *cd python-seabreeze* command indicates that you are moving to another path, in this case, to the "python-seabreeze" directory, which is the one you just cloned. The *cd* (change directory) statement is who performs the directory change action [32].
- **Installation:** The instruction [*python -m pip install .*] installs the SeaBreeze library in the current directory, which is specified with the dot (.).
- **Second way of installation:** If the binary versions are not available, it is possible to install using *python -m pip install --no-binary :all: seabreeze*. This installation is done from source code instead of using precompiled versions, also called binary.

Finally, it was necessary to install the drivers responsible for establishing correct communication with the hardware. This was achieved by executing the following command.

```
Windows PowerShell
PS C:\Windows\system32> seabreeze_os_setup
```

After pressing the *Enter key*, a confirmation menu appeared on the screen. To proceed with the installation, the affirmative option was selected, initiating the installation process. Once the previous operation has been completed, it is possible to establish the correct connection between the spectrometer and the computer [13]. To verify all previous installations, it is suggested to use the commands described in section 2.5.

2.5. *Verification of the environment*

As the last step in setting up the environment, it was necessary to follow the following commands, which check that the installation processes were executed without errors.

2.6. *Open Windows PowerShell and activate the Python environment*

To activate the Python virtual environment, the command *"py"* was entered into the shell, followed by pressing the Enter key:

```
Windows PowerShell

PS C:\Users\USERNAME> py

Python 3.9.6 (tags/v3.9.6:db3ff76, Jun 28 2021, 15:26:21)
[MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more
information.
>>>
```

This results in displaying the current version of Python being used. Now, *Windows PowerShell* can be used as the language interpreter, recognizing all the language's commands. It is important to note that the ">>>" symbols now appear, indicating a change in shell behavior. With this change, the Python interpreter has been initiated, enabling the writing and execution of Python code in real-time, establishing a text input and output environment. This is referred to as the Terminal [33].

2.7. *Import SeaBreeze libraries*

These instructions are necessary to establish the connection to the spectrometer.

```
Python

>>> import seabreeze
>>> seabreeze.use('cseabreeze')
>>> import seatease.cseatease as emu_Spectrometer
```

After successful import, it is recommended to connect the device to the computer and execute the following command block.

2.8. *Connection verification*

With these sentences, it was possible to verify the connection to the spectrometer. The result on the screen depends on the model of the device you are working with.

```
Python
>>> spec = Spectrometer.from_first_available()
>>> print(spec)
<Spectrometer USB4000:USB4H10178>
```

In case the physical device is not available, it is still possible to perform the verification by means of a spectrometer simulation. The instructions for this were:

```
Python
>>> emu_spec = emu_Spectrometer.SeaTeaseAPI().list_devices()[0]
>>> print(emu_spec)
<SeaTeaseDevice: 1>
>>> type(emu_spec)
<class 'seatease.cseatease.SeaTeaseDevice'>
```

The *print()* instruction is used to display the list of available compatible devices on the screen. In the case of the simulation, it only returned *<SeaTeaseDevice: 1>*. Finally, the *type()* instruction displays the type of variable being introduced. In this case, it corresponds to an object of class *'seatease.cseatease.SeaTeaseDevice'*, which is correct and indicates that the virtual device recognition process is correct. The spectrometer model utilized in this work was the *Ocean Optics Flame-T model*. Table 3 displays some parameters of this device.

Specification	FLAME-T
Integration Time	3.8 *ms* to 10 *seconds*
Dynamic Range of system[1]	3.4 x 10^6
Scan rate (max)[2]	260 Hz

Table 3. Optical and spectroscopic specifications.

1. Dynamic range of the system is the range of the detectable light level and can be thought of as the maximum detectable light level at the minimum integration time divided by the minimum detectable light level at the maximum integration time.

2. Scan rate is dependent on the operating computer and not the spectrometer. These figures assume a non-real-time operating system [51].

Before concluding the section, the code statements used in the processes described above are presented in Table 4, along with their representation in pseudocode form, which is used throughout this work. The process of installing the required libraries, packages, and software was performed only once on the computer. Once the device recognition test was successful, we proceeded to design the acquisition system, which is detailed in sections 2.9 and 2.10.

Code	Pseudocode	Action
`Spectrometer.from_first_available()`	spec ← Spectrometer_connection( )	Connection to the first available spectrometer.
`emu_Spectrometer.SeaTeaseAPI().list_devices()[0]`	emu_Spec ← emu_Spectrometer_connection( )	Spectrometer simulation.
`spec._wavelengths`	range_wv ← spec_wavelengths	Reads the accepted wavelength range.
`spec.integration_time_micros_limits`	range_ti ← spec_integration_time_micros_limits	Reads the accepted integration time interval (*us*).

Table 4. Pseudocode representation of the functions to connect to the spectrometer.

2.9. *Device Recognition*

According to Figure 2, the proposed acquisition system begins with the recognition of the spectrometer. To initiate the readings, it was necessary to create a PY file and import the corresponding libraries into the environment. The following libraries were imported: the *SeaBreeze* library, facilitating the connection with the spectrometer; *NumPy*, for manipulating measurements as matrix data; *Matplotlib* for graph visualization; and *time* for controlling the flow of time in data sampling. Additionally, *threading* and *signal* were utilized to create and manage actions in two processing threads: one dedicated to data capture (Main Thread) and the second to review and update parameters modifying the hardware acquisition behavior according to user requirements. Finally, *sys* was used to successfully complete the script execution.

Subsequently, a connection request loop was executed. This involves attempting to connect to the spectrometer, and if any issues arise, the system continues attempting to connect. If a certain number of attempts are exceeded, it indicates an unsuccessful connection and prompts the user to either continue in simulation mode or terminate the program. This functionality can be beneficial, particularly when the device is not owned, and the user wishes to learn how to manipulate spectrogram data available with the SeaBreeze library. Another scenario is when the computer requires additional time to recognize the external device.

Upon a successful connection, the system creates an object of the *Spectrometer* class, which is defined by the model and serial number of the hardware. In this work, the model obtained was *<Spectrometer USB4000:USB4H10178>*. This information is available in the attributes of the spec variable, specified in the pseudocode of Table 4 and Table 5. With this entity, it became feasible to access the captured data and configure spectrometer parameters for the reading, such as: integration time T_I, number of samples to be averaged N, and the desired wavelength limits $wv_{L_{min}}$ and $wv_{L_{max}}$ according to the desired analysis.

Details regarding these parameters and their modifications are presented in section 2.13. The logic used is shown in Table 5 by means of pseudocode.

Following successful linkage with the spectrometer, a synchronous data acquisition system was implemented. This system enables the acquisition of spectral data according to a sampling time T_S, determined by two factors: the user's choice and the integration time T_I required during experimental tests. The process is applicable for both available modes, namely using the Hardware or Virtual mode. This system is explained in detail in section 2.10.

2.10. *Dynamic Real-Time Data Capture System*

This subsection elaborates on the strategies devised to initiate the real-time reading process. This system possesses the characteristic of accommodating the user's external requests and adjusting the reading data accordingly. This is achieved while the system acquires signals from the spectrometer, avoiding any undesired pauses. Consequently, the system is termed dynamic. Figure 4 illustrates the block diagram of this system.

To ensure that the system can manage external user requests, two processing threads have been implemented. These threads enable achieving a concurrency effect, allowing two different processes to run seemingly simultaneously. The components comprising this system are described below.

1. Device Recognition

Algorithm 1: Device Recognition

```
Data: $attempts$
Result: $features, spec$
$count \leftarrow 1$;
while $count \leq attempts$ do
    try
        $hard_{Spec} \leftarrow Spectrometer\_connection()$;
        $status \leftarrow hard_{Spec}$;
        $break$;
    catch
        $count \leftarrow count + 1$;
        $status \leftarrow None$;
end
if $status = None$ then
    $r_{user} \leftarrow Input()$;
    if $r_{user} = 'E'$ then
        $Exit()$ ;
    else
        $emu_{Spec} \leftarrow emu\_Spectrometer\_connection()$;
        $spec \leftarrow emu_{Spec}$;
        $s_{type} \leftarrow 'Virtual'$;
    end
else
    $spec \leftarrow status$;
    $s_{type} \leftarrow 'Hardware'$;
end
if $s_{type} = 'Hardware'$ then
    $range_{wv} \leftarrow spec\_wavelengths$;
    $range_{ti} \leftarrow spec\_integration\_time\_micros\_limits$;
else
    $range_{wv} \leftarrow [300, 1000]$;
    $range_{ti} \leftarrow [3000, 10000000]$;
end
$features \leftarrow [range_{wv}, range_{ti}, s_{type}]$
```

Table 5. Pseudocode of the Device Recognition process.

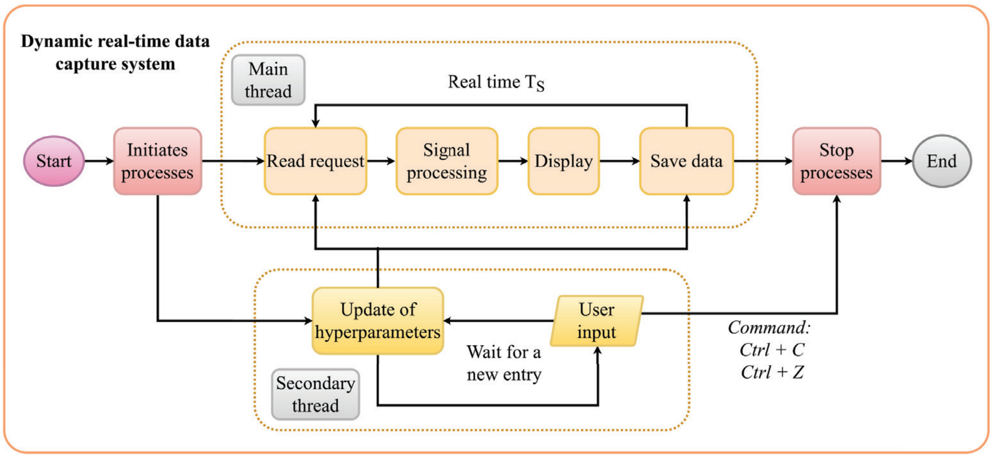


Figure 4. Flowchart diagram of the spectrometer acquisition system.

2.11. *Initiates Processes*

In this block, both threads or sub-processes are initialized: the Main thread and the secondary thread. In the first thread, all the logic related to real-time acquisition was established, and a reading request period of (T_S) was set. Within this interval, the functions related to signal acquisition and processing are executed. It is important to note that this time represents the speed of spectrum reading and should not be confused with the integration time (T_I), which is a parameter of the spectrometer.

To enhance the quality of recorded data, it is necessary to adjust the configuration of the device's hyperparameters. A methodology was implemented to execute these changes, possibly in an aperiodic manner, as indicated by the user. This process is carried out in the second thread and does not affect the acquisition process of the main thread.

The hyperparameters mentioned above are:

- Integration time T_I.
- Limits of the wavelengths of interest wv_L.
- Number of spectra to average N_S to attenuate the present noise.
- Instruction to consider the Background Spectrum (B_S) to eliminate it. This signal level is captured by the device when the sample to be analyzed is absent [34].

Finally, the possibility of terminating the program at any time by using the *Ctrl* + *C* or *Ctrl* + *Z* instructions was established. The logic used is illustrated in Table 6 through pseudocode.

2. Dynamic Real-Time Data Capture System

2.1. Initialization process

```
Algorithm 2: Initialization Processes
  Data: $N$,$T_{S_0}$,$T_{I_0}$,$N_{S_0}$
  Result: $hyperparameters$
1 $attempts \leftarrow N$;
2 $T_S \leftarrow T_{S_0}$;
3 $T_I \leftarrow T_{I_0}$;
4 $N_S \leftarrow N_{S_0}$;
5 $B_S \leftarrow True$;
6 $features \leftarrow device\_recognition(attempts)$;
7 $hyperparameters \leftarrow [features, T_S, T_I, N_S B_S]$;
8 $Activate : key\_board_{interrupt}$;
9 $Start : Second\_thread$;
```

Table 6. Pseudocode of the initialization process.

Next, sections 2.12 and 2.17 detail the methodology employed in the main processing thread as well as in the secondary thread.

2.12. *Main Thread*

This block corresponds to the process of acquisition, digital treatment, and display of the captured signal. Four main actions were established sequentially: (i) data reading, (ii) application of filtering and adjustment techniques, (iii) screen display, and (iv) the option of storing the readings in the computer. Each task is described in detail below.

2.13. *Read Request*

Every T_S seconds, a new read request is made with the hyperparameters allowed by the device. These were initialized in the Initiates processes block in section 2.11. In this work, the accepted hyperparameters corresponding to the device model are presented in Table 7.

Hyperparameters	Value
T_I	3.8 *ms* to 10 *seconds*
wv_L	[191.0969 – 881.4172] *nm*
N_S	Natural number
B_S	True or False
T_S	[500 - 5000] *ms*

Table 7. Options for hyperparameter values employed in the proposed methodology.

The data reading was conducted using the SeaBreeze library, resulting in the acquisition of the spectrum. It was feasible to identify the vector storing the wavelengths present and the count vector. These data are contingent on the nature of the light source, as well as the integration time T_I configured. Table 8 presents the main statements used and their corresponding pseudocode.

Code	Pseudocode	Action
`spec.integration_time_micros(TI)`	set_integration_time(TI)	Sets the TI for spectrum reading.
`spec.wavelengths()`	$x \leftarrow$ read_wv()	Read vector wavelengths.
`spec.intensities()`	$y \leftarrow$ read_int()	Reads vector of intensities.
`spec.f.spectrometer.set_integration_time_micros(Ts)`	vset_integration_time(TI)	Sets the TI for simulation.
`spec.f.spectrometer.get_wavelengths()`	$x \leftarrow$ read_wv_virtual()	Read vector of virtual wavelengths.
`spec.f.spectrometer.get_intensities()`	$y \leftarrow$ read_int_virtual()	Read vector of virtual intensities.

Table 8. Main instructions and their representation in pseudocode.

Finally, with the equivalences provided in Table 8, the logic employed in the Read Request block is presented; refer to Table 9.

2.14. *Signal Processing*

This block was responsible for reducing the signal offset and attenuating the noise present in the data. Since this work primarily focused on capturing absorbance spectra, the signal to be processed indicates the behavior of the light captured solely by the spectrometer. This spectrum can be primarily represented as a composition of three signals: the real physical signal ($y_p(t)$), the noise mainly stemming from the effect of ambient illumination ($y_l(t)$), and the noise caused by electromagnetic effects added during the digital conversion processes, $y_e[n]$. Therefore, the actual physical signal tends to be contaminated.

The above can be described by equation (1).

$$\begin{gathered} y(t) = y_p(t) + y_l(t),\ t \geq 0, \\ \therefore y[n] = y_p[n] + y_l[n] + y_e[n],\ n \geq 0. \end{gathered} \tag{1}$$

Where $y[n]$ corresponds to the contaminated digital signal, which can be plotted on the computer. The digital signals $y_l[n]$ and $y_e[n]$ represent the ambient and electronic noise, respectively. The objective of this processing block was to decrease the effects of noise and ambient light (the latter causing an offset). Therefore, it is possible to reduce the signal offset by subtracting the minimum value present in the reading.

This is expressed in equation (2).

$$y_{adj} = y - min\{y\} \tag{2}$$

where y_{adj} represents a correction of the signal for the offset effect caused by a base noise signal (*Background Spectrum*). This adjustment aligns the values to estimate the maximum readout count and determine its corresponding wavelength. However, this process does not eliminate the noise present. Therefore, it was proposed to use a low-pass discrete Fourier domain filter with a 2nd order Butterworth response. The Butterworth response was chosen because this type of filter maintains a linear phase in the passband [35]. Additionally, due to the nature of the analyzed signal, it is necessary to preserve the correspondence between the wavelengths and their counts. Therefore, the zero-phase Butterworth filter was utilized. This type of filter has the advantage of avoiding a lag with respect to the original signal. Additionally, the filtering process is performed twice: once in the forward direction and the second in the reverse direction in the signal, enhancing the noise attenuation effect [36]. This approach ensures that critical points in absorbance spectra detection are not interfered with. Finally, the cut-off point at which the filter began to attenuate the noise was chosen experimentally, set at 0.1.

2.2. Read Request

```
Algorithm 3: Read Request
  Data: $hyperparameters$
  Result: $x\_select, y\_select$
  if $Valid\_hyperparameters = True$ then
      $Update : T_S$;
      $Update : T_I$;
      $Update : range_{wv}$;
      $Update : N_S$;
      $Update : B_S$;
  else
      $T_S^{new} \leftarrow T_S^{old}$;
      $T_I^{new} \leftarrow T_I^{old}$;
      $range_{wv}^{new} \leftarrow range_{wv}^{old}$;
      $N_S^{new} \leftarrow N_S^{old}$;
      $B_S^{new} \leftarrow B_S^{old}$;
  end
  if $s_{type} = 'Hardware'$ then
      $set\_integration\_time(T_I)$;
      $x = read_{wv}()$;
      $y = read_{wv}()$;
  else
      $vset\_integration\_time(T_I)$;
      $x = read\_wv\_virtual()$;
      $y = read\_int\_virtual()$;
  end
  $Update : hyperparameters$;
  $x\_select \leftarrow x[range_{wv}^{new}]$;
  $y\_select \leftarrow y[range_{wv}^{new}]$;
```

Table 9. Pseudocode of the Read Request block.

In this work, the FFT algorithm was implemented using the *fft.fft* function of the NumPy library for the choice of the cut-off point [37]. For the design and application of the filter, *signal.butter* and *signal.sosfiltfilt* from SciPy were utilized [38]. This filtered signal was displayed on the screen and stored in the computer in a CSV format file. Table 10 below presents the pseudocode that explains the previously detailed process.

2.3. Signal processing

```
Algorithm 4: Signal Processing
    Data: $x, y\_select, l_c$
    Result: $\hat{y}$
 1  if $First\_reading = True$ then
 2      $Order \leftarrow 2$;
 3      $\Delta_x \leftarrow (max\{x\} - min\{x\})/(length(x) - 1)$;
 4      $f_s \leftarrow 1/\Delta_x$;
 5      $\Phi_n \leftarrow l_c/(f_s/2)$;
 6  else
 7      $pass$;
 8  end
 9  $y\_adj \leftarrow y\_select - min\{y\_select\}$;
10  $sos_{coefficients} \leftarrow butter_{coeff}(order, \Phi_n, type = 'filtfilt')$;
11  $\hat{y} = filter(sos_{coefficients}, y\_adj)$;
```

Table 10. Pseudocode of the Signal Processing block.

2.15. *Display*

After signal processing, an interactive graph was created that updates with each reading. This means that the display automatically adjusts the limits of the coordinate axes according to the light input to the spectrometer to achieve correct visualization. This function updates according to the time T_S and accommodates changes made by the user, particularly those from the secondary thread. In this graph, the most important hyperparameters are the limits of the wavelengths to be displayed, according to the desired experimental process.

Section 3.2 showcases the graphs obtained with the proposed methodology. Likewise, the structure of the code used is presented in Table 11.

2.4. Display

```
Algorithm 5: Display
    Data: $x\_select, \hat{y}$
    Result: $None$
 1  if $First\_reading = True$ then
 2      $Initialize : graph$;
 3      $Set : Title$;
 4      $Set : X\_Label$;
 5      $Set : Y\_Label$;
 6  else
 7      $pass$;
 8  end
 9  $X\_data \leftarrow x\_select$;
10  $Y\_data \leftarrow \hat{y}$;
11  $Set : X\_lim(range^{new}_{wv})$;
12  $Set : Y\_lim([0.98 \cdot min\{\hat{y}\}, 1.02 \cdot max\{\hat{y}\}])$;
13  $Graph\_Draw(X\_data, Y\_data)$;
14  $Graph\_uptade()$;
```

Table 11. Pseudocode of the Display block.

2.16. *Save Data*

Finally, the recorded and processed samples are stored in a CSV format file. After each save action, the acquisition loop restarts to initiate a new read request. As previously mentioned, the hyperparameter update and verification process were implemented in the secondary thread to avoid halting the acquisition loop and to concentrate the functions on signal processing. Section 2.17 provides details about this secondary thread.

2.17. *Secondary Thread*

This system was designed with the possibility of making changes in the hyperparameters at any time, allowing the updating of the spectrum capture process, as well as in the visualization of the graph. Due to the use of the *input( )* function, the program constantly waits for a change. Once a correct input is detected, the system updates the old hyperparameters used in the *Read Request* and *Display* blocks. Subsequently, the system returns to waiting for a new input. Each hyperparameter update is performed at the end of each reading and signal processing, avoiding errors during the acquisition process. If two previous consecutive inputs were the same, a workflow was designed to avoid unnecessary updates. This process continues until the user finishes the program from the terminal, having not only the secondary thread, but also the main thread.

2.5. Secondary thread

```
Algorithm 6: Secondary Thread
   Data: $hyperparameters^{old}$
   Result: $hyperparameters^{new}$
 1 $def\ \ hyper\_uptade\_Function(hyperparameters^{old}, hyperparameters^{new})\{$
 2 if $hyperparameters^{old} \neq hyperparameters^{new}$ then
 3     $Update : hyperparameters;$
 4 else
 5     $pass;$
 6 end
 7 $\}$
 8 while $True$ do
 9     try
10         $str\_hyper \leftarrow input();$
11         $data\_hyper \leftarrow str2float\_hyper\_Function(str\_hyper);$
12         $hyperparameters^{new} \leftarrow data\_hyper;$
13         $hyper\_uptade\_Function(hyperparameters^{old}, hyperparameters^{new})$
14     catch
15         $print('Invalid\ \ entry');$
16         $continue;$
17 end
```

Table 12. Pseudocode of the secondary thread.

2.18. *Stop Processes*

As the last block of the dynamic real-time data capture system, for this application, the use of the terminal was proposed as a method of program completion. Both threads were kept running until the *Ctrl* + *C*/*Z* command was entered in the Python Terminal while the script was running.

A process terminates when it completes the execution of its final statement and requests the operating system to terminate it via the *exit( ) system call*. At this point, the process can return a status value, typically an integer, to its parent process awaiting it (via the *wait( ) system call*). All process resources, including physical and virtual memory, open files, and I/O buffers, are deallocated, and reclaimed by the operating system [21].

In section 3, we present what was obtained during the execution of the proposed methodology for a cell phone light source, since being a low-quality signal, it allowed the design of a robust methodology. The quality metrics utilized to assess the feasibility of the signal processing stage are presented in section 2.19 below.

2.19. *Validation*

It is necessary to verify that the signal adjustment and filtering process does not diminish the quality of the captured information. Therefore, it is proposed to use three metrics to evaluate the processing block: the signal-to-noise ratio (SNR), the root mean square error (RMSE), the coefficient of determination (R-squared) and two indicators based on the amplitude of the signals (DA) and the Shannon entropy (DE) [39-41]. According to [42, 43], higher SNR values generate more efficient results, and the lower the RMSE, the more desirable the result. Likewise, it is desired that R-square be as close to 1 as possible; see equation (3).

$$SNR(dB) = 10 \cdot log\left(\frac{\sum_{i=0}^{N-1}(y_a[i])^2}{\sum_{i=0}^{N-1}\left(y_a[i]-\hat{y}[i]\right)^2}\right), \qquad (3)$$

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=0}^{N-1}\left(y_a[i]-\hat{y}[i]\right)^2} \ ,$$

$$R_{square} = 1 - \frac{\sum_{i=0}^{N-1} \left(y_a[i] - \hat{y}[i]\right)^2}{\sum_{i=0}^{N-1} \left(y_a[i] - \overline{y}_a\right)^2}.$$

Where $y_a[i]$ is the i-th value of the adjusted spectrum; the adjusted signal was taken to prevent the offset level from modifying the result of the metrics. The signal $\hat{y}[i]$ is the i-th data of the filtered signal. Finally, N is the length of the signals, in this work for the version of the spectrometer used, it was 3520 data. Since the low-pass filter tends to attenuate the signal at points higher than the cut-off point, in this work set at 0.1. A metric was proposed that indicates the variation of the amplitudes of the original and the filtered signal. This is especially useful in scenarios where the value associated with local maxima must be faithfully identified. The mathematical relationship is expressed in equation (4).

$$DA = \frac{|\Delta y - \Delta \hat{y}|}{\Delta y} \cdot 100\%. \qquad (4)$$

Where $\Delta y = y_{max} - y_{min}$, knowing that is the sample captured by the spectrometer. Similarly, $\Delta \hat{y} = \hat{y}_{max} - \hat{y}_{min}$ represents the maximum amplitude of the filtered signal. This metric helped to quantify the attenuation effect due to filter action.

Finally, a coefficient indicating the difference between the information preserved in the filtered signal by Shannon entropy was used. Equation (5) shows the relationship.

$$DE = \frac{|Hy - H\hat{y}|}{Hy} \cdot 100\%. \qquad (5)$$

Where is the entropy calculated for the original signal y, $H_{\hat{y}}$ is the entropy for the filtered signal $\hat{y}$. The *scipy.stats.entropy* function from the SciPy library was used for this metric [38]. Section 3 presents the results obtained during the execution of the proposed methodology.

3. Results and discussion

The programming and execution of this methodology was performed using the Visual Studio Code editor version 1.88 and using Python 3.9.6. The computer used was a DELL laptop with 8 Gb of RAM and AMD Ryzen 5 2500U processor with Windows 10. The following are the spectra captured from a cell phone LED light, as well as those obtained when processing the virtual data provided by the SeaBreeze library.

3.1. *Hardware*

Ten shots were taken every 500 *ms* (T_S), from an LED light source with an integration time T_I set at 10 *ms*. The number of samples to be considered for averaging N_S was 5. Figure 5 shows all the captured spectra.

3.2. *Acquisition and processing*

The total acquisition time was 4.5 seconds. In Figure 5, in each sample, there was a variation in the distance between the detector and the light source. For this reason, a variation in the amplitude of the captured signals is observed.

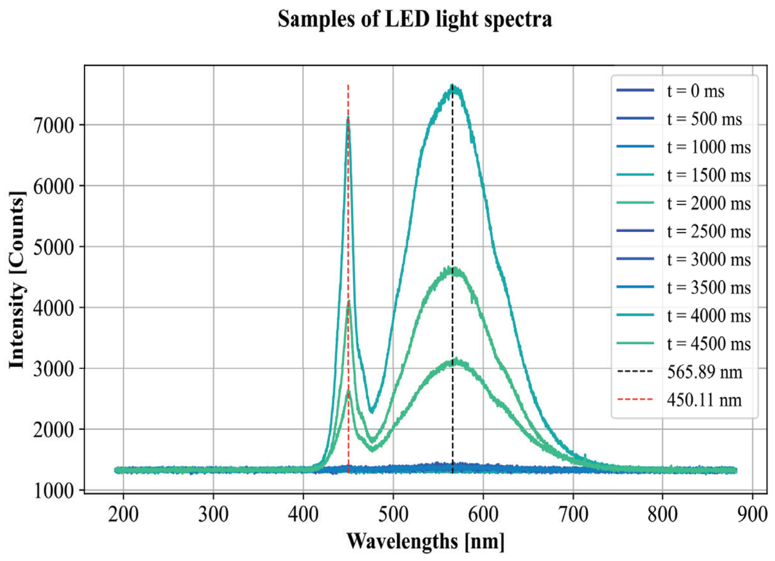


Figure 5. LED light spectra of 10 readings taken every 500ms. The wavelengths associated with the two peaks are approximately 450 nm and 565 nm, corresponding to the blue and green color range, respectively.

It is observed that wavelengths corresponding to violet and ultraviolet are not present. Likewise, infrared light is not captured as a component of this light source. The samples that capture the most information are at times t = [2000, 4000, 4500] *ms*. Thus, the rest of the signals are mainly noise; this was caused by the movement of the light source and to corroborate the real-time change of the spectrum. The following is a series of comparisons between some samples of these signals and their respective processing.

Figure 6 shows the different acquired signals and what was obtained after processing, i.e., offset adjustment and a low-pass filter. In the samples, the signals that did not capture the LED light spectrum, Figure 6a, 6c, 6e and 6g possess an average level of approximately 1330 [Counts], while the amplitudes of the samples at t = [2000, 4000, 4500] *ms* are 3174.76, 7656.53 and 4670.18 [Counts], respectively. This implies that there is a difference in amplitude of up to 5 times between the noise signals and the maximum recorded intensity. Being mainly noise signals, the adjustment and filtering performed do not reveal new relevant information; this can be seen in Figures 6a, 6b, 6d, 6f and 6h.

In the case of Figures 6k, 6m and 6o, which correspond to the moments when the light source had the greatest distance from the detector, the filter accentuated the waveform and allowed us to observe that both peaks are around the same wavelengths of Figure 5. Therefore, using the proposed processing it was possible to identify more accurately the presence of blue tones (Figure 6n and 6p), a situation that is more complex to identify in the unprocessed signals due to the noise present and its amplitude levels. Finally, the offset adjustment and the second-order flat response zero offset filter attenuated the noise while preserving the amplitude and coincidence characteristics with the original wavelengths. The above can be seen in the pairs of Figure 6i, 6j, 6q, 6r and 6s, 6t.

To analyze the filter efficiency, 5 metrics were implemented: the SNR signal to noise ratio, the RMSE value, the R-square coefficient, the DA Amplitude difference and the DE Shannon entropy difference. Section 3.3 presents the value of these metrics for each of the samples.

3.3. *Analysis of adjustment and processing*

To validate that the filter preserves as much useful information as possible, the 5 metrics proposed in section 2.19 were calculated for each filtered signal. Table 13 shows the values according to the time at which they were acquired.

The SNR, RMSE and R-Square values of the signals captured at times t = [2000, 4000, 4500] *ms* turned out to be the highest of all samples. For DA metrics low values represent little discrepancy in amplitudes, while for DE, high values indicate that the filtered signal information is more representative compared to the original spectrum. Therefore, the application of the filter at the 0.1 cut-off point was effective in preserving the waveform of the signals categorized as Light type and maintaining a difference in the original amplitude of up to 4.9 %. Likewise, these signals had an

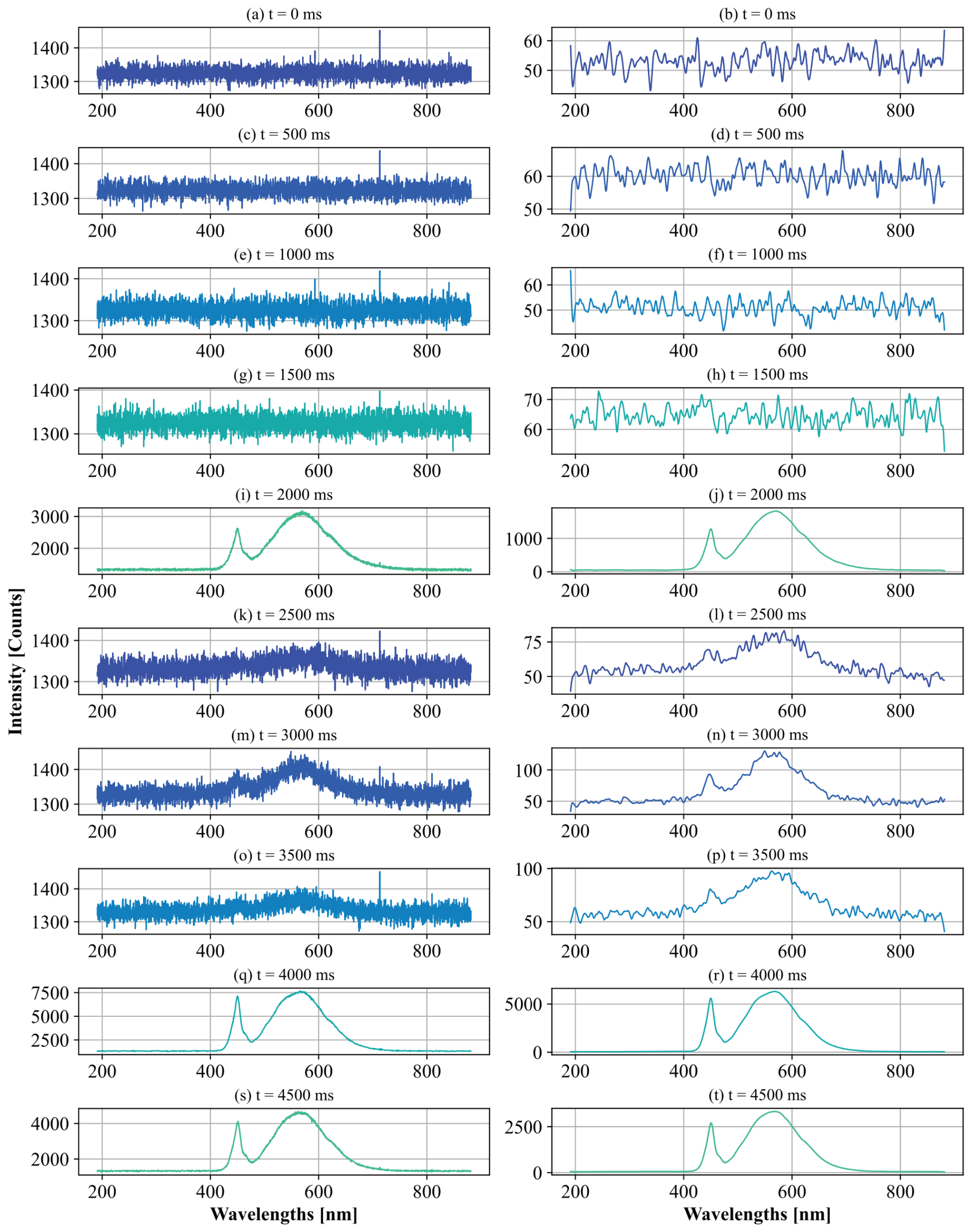


Figure 6. The first column displays the unadjusted spectra. In the second column are the fitted signals filtered by the 2nd order zero offset filter.

SNR of up to 38.54 %, a maximum RMSE value of 27.83 % and a maximum determination coefficient of 0.9998, resulting in a correct performance of the proposed processing block.

For the signals t = [2500, 3000, 3500] *ms* which were captured with a greater distance between the light source and the detector, the result of applying the offset adjustment and filtering, were signals with a coefficient of determination of up to 0.68. That is, the filtered signal managed to maintain 68 % of the information of interest present in the original signal. Likewise, in the spectrum t = 3000 *ms* the change in entropy was up to 0.71; this magnitude is low because the filter eliminated the noise components, but due to the low SNR ratio (12.99 *dB*), the resulting signal still maintains information that is not of interest.

Finally, it is possible to observe that, in all the signals without the presence of the light source of interest, the difference in amplitudes is greater than 83.37 %. This is because of attenuating the noise components that are higher than the cut-off point in the Fourier domain. It can also be seen that these signals have the lowest SNR of the group of samples.

Time [*ms*]	**SNR**	**RMSE**	**R - Squared**	**DA (%)**	**DE (%)**	**Type**
0	10.9253	15.7542	0.0547	88.6760	0.0190	Noise
500	11.9471	15.7301	0.0458	89.4279	0.0114	Noise
1000	10.6042	15.7757	0.0467	83.3754	0.0184	Noise
1500	12.4802	15.8233	0.0509	85.3460	0.0113	Noise
2000	**31.6326**	**18.209**	**0.9989**	**4.9052**	**8.1775**	**Light**
2500	11.8748	15.8183	0.2609	70.1125	0.1395	Noise
3000	12.9992	15.8471	0.6887	44.7016	0.7118	Noise
3500	12.7887	15.7033	0.3956	69.0415	0.2112	Noise
4000	**38.5495**	**27.8310**	**0.9998**	**1.8674**	**9.0734**	**Light**
4500	**35.4580**	**20.8991**	**0.9996**	**3.1997**	**9.4096**	**Light**

Table 13. Validation metrics applied to the LED light test spectra (T_I = 10 *ms*).

To visualize the effect of the processing block on all the resulting signals, the comparative bar chart in Figure 7 was made. Each bar was normalized according to the maximum values of each metric in Table 13.

3.4. *Dynamic real-time data capture system analysis*

During the acquisition tests, the double thread functionality was used to attend requests for changes in the system hyperparameters. Likewise, tests were performed with different sampling times T_S to demonstrate the accuracy of the system. 1000 readings were taken for each desired T_S sampling time, and the actual time spent during the $\overline{T_S}$ program run was averaged. The results are shown in Table 14.

All measurements were performed using a constant value of $T_I = 10$ *ms*, an average of $N_S = 5$ and a selection of the full range of wavelengths available in the spectrometer: $wv_L = [191.09, 881.41]$ nanometers.

T_I (*ms*)	$\overline{T_S}$ (*ms*)	Error (%)
5000	5001.0	0.0193
1000	1004.1	0.4134
500	511.0	2.1975
200	230.2	15.0936
100	129.1	29.0974
50	120.9	141.7637

Table 14. System accuracy with respect to sampling time.

According to the recorded data, the system has an accuracy of up to 99.98 % when the sampling time is equal to 5 seconds. As T_S decreased, the actual average reading time stabilized at approximately 120 *ms*. This time is the time used to execute the Processing block, the Display block and the Save action, in addition to the time it takes for the system to fully transmit the spectrum to the computer. According to the tests performed, the greatest number of resources is occupied by the interactive graphic, which is updated every time there is a new reading, averaging 105.02 *ms*.

Thus, it is possible to establish that the designed software allows manipulating parameters during the acquisition stage and without stopping it, allowing the system to be implemented in experiments involving different sampling times. Because the system is dynamic and real-time, it can be used in a wide variety of

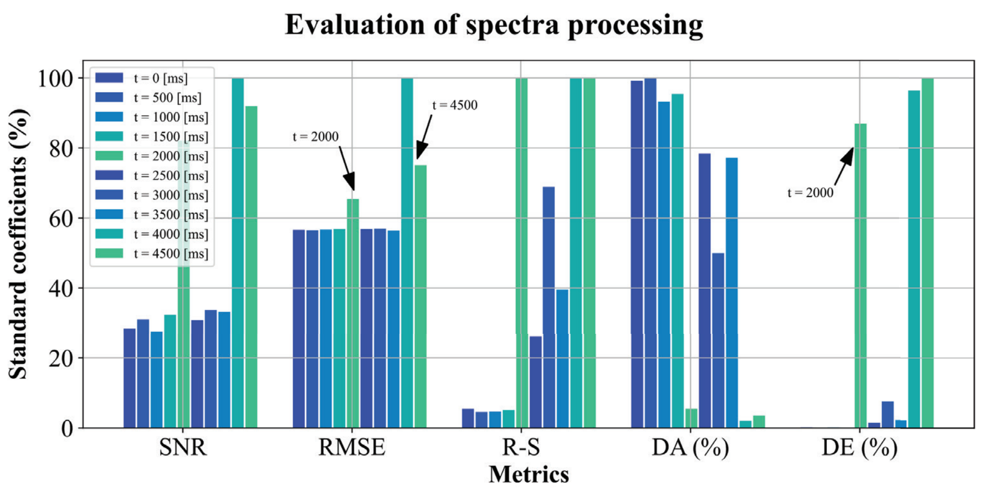


Figure 7. Comparative plot by metric of the 10 LED light test samples. The bars with seaweed green shades are mostly distinguishable from the set of readings. Signals with noise do not show a large variation in bar height (colors with sea-blue shades). Finally, signals at times t = [2500, 3000, 3500] *ms* have bars with values that can be considered intermediate because they still have information about the distribution of LED light intensities.

applications in the field of optical biosensors. Since many biological processes take minutes to hours, it is possible to analyze changes using spectrophotometric techniques [44-47]. For example, in [48] it is indicated that they used 3 hours to analyze changes in the refractive indices of cells sitting on a plasmo-mechanical sensor integrated with microfluidics; during this process, readings were taken every 30 minutes. Likewise, in [49], they state that most detection studies of SARS-COV2 and other viruses take between 10 minutes to 3 hours.

Therefore, the designed system is capable of being potentially useful during the detection process of metabolites or pathogens present in biological tissue, food, and beverages and even in studies of environmental pollutants, since it allows establishing sampling times of the spectra with high precision for values greater than 500 *ms*; considering the use of T_I in 10 *ms*.

The system responded to requests to update the hyperparameters at any instant and was able to display the changes related to the graph (wv_L and T_I), in addition to saving the data in a CSV file in different sheets, according to the T_I and T_S. It is necessary to mention that, so far, the adjustment of these times was totally manual. This implies that the T_I required by the experiment had to be considered to establish an optimal T_S sampling time. It is proposed to use prediction techniques based on neural networks that find the relationship between both times as a function $T_S = f(T_I)$. This in order that the sampling time does not interfere with the reading process of the experiment and affect its performance. In addition, for new versions of the system, it is possible to contribute to the decrease of the latency time of the processing block by making a third thread dedicated to writing the data in the CSV file and extending the possibility of using sampling times lower than 500 *ms* (considering the integration time, as previously mentioned).

Finally, the system was able to perform the readings in real-time and allowed to save the spectra of interest every nT_S seconds, with $n \in N$. This has practical effects when observing the behavior of the phenomenon and not saving the information until a multiple of T_S time. This helps to reduce the complexity of the subsequent analysis processes required.

This system has the potential to be introduced, for the most part, inside a microcontroller. Opening the possibility of generating a portable system dedicated to the acquisition and cleaning of spectra. Moreover, in the era of Industry 4.0, it is possible to integrate IoT technology, cloud computing and analytics, AI, and machine learning techniques to the proposed system to analyze and process the information and finally contribute to detection in less time and with higher accuracy, depending on the field of application [50].

4. Conclusions

The presented software achieves a sampling time accuracy exceeding 99 %, and its multithreaded processing capability allows for the dynamic management of hyperparameters in response to user requests, resulting in reduced overall latency times. The captured spectra could be displayed on-screen in real-time, with properties of the coordinate axis boundaries automatically adjusted based on the amplitudes of the recorded signal or the selection of wavelengths of interest.

The use of second-order filters with flat response and zero-phase are effective for attenuating noise present in spectral signals, even when the R-Square value is approximately 60 %. Additionally, the system maintained a difference in maximum amplitudes of less than 5 %; however, filter efficiency tends to increase when the SNR of the original signal itself is higher from the beginning.

As an additional feature, it was found that it is possible to store the light spectrum data at a rate different from that stipulated by T_S. Thereby, it is possible to store spectra at each update interval of the T_S plot. In practice, this feature is intended to support the visualization of the dynamics of the studied system while controlling the size of the generated file.

Thus, a Python-based system developed for real-time acquisition and processing of light spectra offers a versatile and efficient solution for a wide variety of applications, with notable potential in sensors. By enabling the real-time detection and analysis of electromagnetic spectra, particularly in optical biosensors, the system contributes significantly to measuring optical properties such as transmittance, reflectance, or absorbance. This capability extends to calculating sensograms during biofunctionalization experiments, enhancing research possibilities.

Finally, the system's adaptability allows for seamless integration into different hardware setups, reducing technological dependencies on other software solutions. This aspect is particularly advantageous as it eliminates the need for costly licenses or additional software packages associated with commercial options like Ocean Optics spectrometers or MATLAB. Leveraging Python's open-source nature and libraries like Python-SeaBreeze, developers can create tailored solutions without constraints imposed by proprietary software, thereby fostering innovation and exploration in spectroscopy research and analysis.

Acknowledgement

The authors would like to thank Instituto Politécnico Nacional for the use of its facilities, BEIFI SIP20240477 and SIP20241266 projects, and CONAHCYT for its support with postgraduate scholarships and for the project CONAHCYT N°319037 "Escuela Mexicana de Ventilación".

References

1. Togawa, T., Tamura, T., & Öberg, P. Å. (2011). *Biomedical sensors and instruments* (2nd ed.). Taylor & Francis Group.
https://doi.org/10.1201/b10775
2. Khandpur, R. S. (2003). *Handbook of biomedical instrumentation* (2nd ed.). McGraw-Hill.
3. International Union of Pure and Applied Chemistry (IUPAC). (2014). Biosensor. In *IUPAC eBooks*.
4. Gómez, D. R. (2012, February 23). *Biosensores ópticos de alta sensibilidad basados en técnicas de modulación plasmónica*. [Online]. Available:
http://hdl.handle.net/10347/5134
5. Damborský, J. K., & Š., J. (2016). Optical biosensors. *Essays in Biochemistry*, 60, 91–100.
https://doi.org/10.1042/EBC20150010
6. Marín Silva, D. A. (2022). *Funcionalización de matrices a base de polímeros biodegradables mediante incorporación de nanopartículas*. Universidad Nacional de La Plata.
7. Tektronix. (2023). *Analysis, Fundamentals of Real-Time Spectrum*. [Online]. Available:
https://www.tek.com/en/documents/primer/fundamentals-real-time-spectrum-analysis
8. Optics, O. (n.d.). *USB Programmer: Installation and Operation Instructions*. Florida.
9. Simulink, M. &. (n.d.). *Ocean Optics Spectrometer support from Instrument Control toolbox*. [Online]. Available: https://la.mathworks.com/hardware-support/ocean-optics-spectrometers.html
10. Lutz, M. (2010). *Programming Python: Powerful object-oriented programming* (4th ed.). O'Reilly Media, Inc.
11. Python Documentation. (2024, April 6). *Python frequently asked questions*. [Online]. Available:
https://docs.python.org/es/3/faq/general.html
12. Datta, S. (2022, July 14). *FreeCodeCamp.org*. [Online]. Available:
https://www.freecodecamp.org/news/run-python-script-how-to-execute-python-shell-commands-in-terminal/
13. Poehlmann, A. (2019). *Python-Seabreeze documentation*. [Online]. Available:
https://python-seabreeze.readthedocs.io/en/latest/
14. Nagar, S. (2017). *Introduction to Python for engineers and scientists: Open source solutions for numerical computation*. Apress.
https://doi.org/10.1007/978-1-4842-3204-0_2
15. Halvorsen, H. P. (2020). *Python for science and engineering.*
16. Burn, I. (2022, November 8). *CCM*. [Online]. Available:
https://es.ccm.net/ordenadores/linux/3810-que-es-un-shell-y-para-que-se-utiliza/

17. Wheeler, S., & Buck, A. (2023, June 28). *Microsoft Learn*. [Online]. Available: https://learn.microsoft.com/en-us/powershell/scripting/overview?view=powershell-7.4

18. Brownlee, J. (2023, November 22). *Python Multiprocessing: The complete guide*. Super Fast Python. [Online]. Available: https://superfastpython.com/multiprocessing-in-python/

19. Patrizio, A. (2022, September 23). *Single-core vs. multi-core CPUs*. [Online]. Available: https://www.networkworld.com/article/971425/single-core-vs-multi-core-cpus.html

20. Ramanathan, R. M. (2015). *Intel multi-core processors*.

21. Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating system concepts* (10th ed.). WILEY.

22. Bala, P. C. (2022, December 8). *Python Threading: An Introduction*. [Online]. Available: https://geekflare.com/python-threading/

23. Python Software Foundation. (2024, April 6). *Threading - thread-based parallelism*. [Online]. Available: https://docs.python.org/3/library/threading.html

24. Coursera Staff. (2023, November 29). *What are scripting languages? (And why should I learn one?)* Coursera. [Online]. Available: https://www.coursera.org/articles/scripting-language

25. Wheeler, S. (2024, April 3). *About execution Policies - PowerShell*. Microsoft Learn. [Online]. Available: https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_execution_policies?view=powershell-7.4

26. Wheeler, S. (n.d.). *Set-ExecutionPolicy (Microsoft.PowerShell.Security) - PowerShell*. Microsoft Learn. [Online]. Available: https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.security/set-executionpolicy?view=powershell-7.4. [Accessed April 1, 2024]

27. Microsoft. (n.d.). *Download .NET Framework | Free official downloads*. [Online]. Available: https://dotnet.microsoft.com/en-us/download/dotnet-framework

28. Microsoft. (n.d.). *Download .NET Framework 4.8 | Free official downloads*. [Online]. Available: https://dotnet.microsoft.com/en-us/download/dotnet-framework/net48

29. Saleh, M. (2024, February 24). *Latest supported Visual C++ Redistributable downloads*. Microsoft Learn. [Online]. Available: https://learn.microsoft.com/en-us/cpp/windows/latest-supported-vc-redist?view=msvc-170

30. CodeOp. (2023, March 24). *7 key backend programming languages explained*. [Online]. Available: https://codeop.tech/7-key-backend-programming-languages-explained/

31. Chocolatey Software. (n.d.). *Chocolatey - the package manager for Windows*. [Online]. Available: https://chocolatey.org/

32. Gerend, J. (2023, February 3). *CD*. Microsoft. [Online]. Available: https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/cd

33. GeeksforGeeks. (2021, June 29). *Difference between Terminal, Console, Shell, and Command Line*. [Online]. Available: https://www.geeksforgeeks.org/difference-between-terminal-console-shell-and-command-line/

34. Ocean Insight. (n.d.). *Glossary on Spectroscopy and Technical terms | Ocean Insight*. [Online]. Available: https://www.oceaninsight.com/knowledge-hub/glossary/

35. Proaki, J. G., & Manolakis, D. G. (2007). *Digital signal processing: Principles, algorithms and applications*. Pearson Education.

36. The MathWorks, Inc. (n.d.). *Zero-phase digital filtering - MATLAB filtfilt- MathWorks*. [Online]. Available: https://la.mathworks.com/help/signal/ref/filtfilt.html?lang=en

37. NumPy Team. (2023, September 16). *NumPy*. [Online]. Available: https://numpy.org/

38. The SciPy Community. (2024, April 3). *SciPy documentation*. [Online]. Available: https://docs.scipy.org/doc/scipy/index.html

39. Semmlow, J. (2005). *Circuits, systems, and signals for bioengineers: A MATLAB-based introduction*. Academic Press.

40. Hagan, M. T., Demuth, H. B., Beale, M. H., & Jesús, O. D. (2014). *Neural network design*. Martin Hagan.

41. Date, S. (2022, March 21). *The complete guide to R-squared, adjusted R-squared and pseudo-R-squared*. Medium. [Online]. Available: https://towardsdatascience.com/the-complete-guide-to-r-squared-adjusted-r-squared-and-pseudo-r-squared-4136650fc06c

42. Zheng, Q.-W., Lingping, K., Jeng-Shyang, P., & Wei-Min, W. (2024). A novel discrete artificial bee colony algorithm combined with adaptive filtering to extract fetal electrocardiogram signals. *Expert Systems with Applications*, 247, 123173. https://doi.org/10.1016/j.eswa.2024.123173

43. Li, C., Deng, H., Yin, S., Wang, C., & Zhu, Y. (2023). sEMG signal filtering study using synchrosqueezing wavelet transform with differential evolution optimized threshold. *Results in Engineering*, 18, 101150. https://doi.org/10.1016/j.rineng.2023.101150

44. Sehrish, B., Aqsa, T., Ijaz, K., Maham, L., Silvana, A., Hongxia, Z., & Akhtar, H. (2024). A review of nanophotonic structures in optofluidic biosensors for food safety and analysis. *Trends in Food Science & Technology*, 147, 104428. https://doi.org/10.1016/j.tifs.2024.104428

45. Alemayehu, G. K., Abebe, B. G., Alemu, K. H., Tamirat, A. D., Mulubirhan, D., & Habtamu, D. M. (2023). Optoplasmonic biosensor for lung cancer telediagnosis: Design and simulation analysis. *Sensors International*, 4, 100232. https://doi.org/10.1016/j.sintl.2023.100232

46. Domínguez García, V., & Ramírez Durán, N. (2017). *Temas selectos de biomedicina en Ciencias de la Salud*. Ediciones Eón/Universidad Autónoma del Estado de México.

47. Katey, B., Voiculescu, I., Penkova, A. N., & Untaroiu, A. (2023). A review of biosensors and their applications. *ASME Open Journal of Engineering*, 2, 020201. https://doi.org/10.1115/1.4063500

48. Solís Tinoco, V. I., Lechuga, L. M., Sepúlveda, B., & Jiménez Jiménez, D. (2016). Development of integrated plasmomechanical sensors in microfluidic devices for live cell analysis. [Art]. Universitat Autònoma de Barcelona.

49. Bakr, A. T., Qussay, A.-J., Surjeet, C., Yousif, A. M., Sarvesh, R., Vishal, C., & Arsad, N. (2024). State-of-the-art telemodule-enabled intelligent optical nano-biosensors for proficient SARS-CoV-2 monitoring. *Microchemical Journal*, 197, 109774. https://doi.org/10.1016/j.microc.2023.109774

50. IBM. (2024, April 8). *What is Industry 4.0 and how does it work?* [Online]. Available: https://www.ibm.com/topics/industry-4-0

51. Ocean Insight. (2015). *FLAME Scientific-Grade Spectrometer: Installation and Operation Manual.*