Technische Universität Berlin

Faculty IV - Electrical Engineering and Computer Science

Chair of Sensor and Actuator Systems

Analysis and Spike Detection of Neural Data from a CMOS MEA System

Bachelor Thesis

Laura Ibáñez Martínez

March 2019 - September 2019

Technische Universität Berlin
Faculty IV - Electrical Engineering and Computer Science

Chair of Sensor and Actuator Systems
Prof. Dr. Roland Thewes

Analysis and Spike Detection of Neural Data from a CMOS MEA System

Author:	Laura Ibáñez Martínez
Student ID:	408029
Submission date:	05.09.2019
Supervisor:	Dr. Norman Dodel

Affidavit

I hereby confirm that this term paper entitled ”Analysis and Spike Detection of Neural Data from a CMOS MEA System” is the result of my own work. I did not receive any help or support from commercial consultants. All sources and / or materials applied are listed specified in this paper.
Furthermore, I confirm that this paper has not yet been submitted as part of another examination process neither in identical nor in similar form.

Berlin, 05.09.2019
Place, Date

Signature

Abstract

This thesis is intended to deal with the problem of analysis and spike detection in neural data acquired from a CMOS Microelectrode Array (MEA) system. In order for this to be carried out a Graphical User Interface (GUI) application has been created.

The data to analyze comes from an in-vitro recording and stimulation system which uses a 65 x 65 CMOS MEA and it is stored in HDF5 files with the .cmcr extension. The GUI application developed in Python is able to read these files and perform a number of tasks which facilitate the detection and visualization of neural activity within the tissue subjected to analysis. This is accomplished by interpreting the voltage measurements at every coordinate of the array over time for the search of Action Potentials (APs) or spikes. Once the spikes are detected the information is stored to be presented in different ways and a comparison is carried out to detect when and where most activity has taken place.

The GUI application created enables the visualization of both raw and filtered data at a particular coordinate over time and showing the spikes that have been detected. Moreover, a second type of plot makes it possible to view the whole array filtered data for an exact time sample, as well as which coordinates present most neural activity. Finally, a number of tables show useful information about the pixel coordinates, time and height of these spikes.

Keywords: *CMOS MEA system*, *HDF5 file*, *GUI application*, *spike detection.*

Contents

List of Abbreviations

AP Action Potential.

BMI Brain-Machine Interface.

CLI Command Line Interface.

CMOS Complementary Metal-Oxide-Semiconductor.

DC Direct Current.

GUI Graphical User Interface.

HCI Human Computer Interface.

HDF Hierarchical Data Format.

HP High-Pass.

ICA Independent Component Analysis.

IDE Integrated Development Environment.

MEA Microelectrode Array.

MVP Model-View-Presenter.

NEO Nonlinear Energy Operator.

OOP Object-Orientated Programming.

OS Operating System.

PCB Printed Circuit Board.

SWTP Stationary-Wavelet-Transform Product.

WIMP Windows-Icons-Menus-Pointer.

List of Figures

List of Tables

1. Introduction

This thesis is the result of the work achieved after spending an exchange semester at TU Berlin, and more specifically the Chair of Sensor and Actuator Systems, in order to conclude the Bachelor's degree in Industrial Electronics and Automatic Control Engineering coursed at UPC.

1.1. Motivation

As part of the work being developed at the Chair of Sensor and Actuator Systems in cooperation with the NMI Natural and Medical Sciences Institute at the University of Tübingen regarding the design of CMOS MEA systems for interfacing purposes with neural tissue, a topic concerning analysis of the recorded data has been considered appropriate for the development of this thesis. Moreover, the need for a faster and more intuitive way of extracting relevant information from the mentioned data has lead to the idea of developing a GUI application which is able to perform these tasks. This way of analyzing and visualizing the data offers a large number of possibilities and provides an interactive experience with the user.

As applications such as Brain-Machine Interfaces (BMIs) for biomedical purposes within others become more popular the need for neural data analysis and spike detection methods grows. In cases like the actual one for in-vivo and in-vitro techniques the use of Microelectrode Arrays makes it possible to record extracellular activity in neural tissue from a number of electrodes at the same time. These nerve cells communicate by means of Action Potentials (APs) or spikes which can be interpreted from fast, short duration transients in voltage. Appropriate spike detection and spike sorting methods are useful for reducing the amount of relevant information that needs to be stored and for detecting where the nerve cells are found within the neural tissue subjected to analysis. Once the nerve cells are identified different studies to see how the tissue reacts to certain stimulus can be carried out.

1.2. Objectives

The principal objective of this thesis is to provide an application capable of analyzing and visualizing data acquired from the CMOS MEA system and stored in the form of HDF5 files. More specifically, the application aims to present the data in different ways (timeplot for a specific pixel coordinate and grid plot for a specific time sample) and both raw and filtered. The filter applied is a High-Pass (HP) filter to remove the DC offset of the signal. Additionally, some spike detection method must be applied and information such as the number, coordinates and height of the spikes (also referred to as peaks throughout the document) extracted and stored to be presented in different ways, including plotted and in a table view.

In respect to the creation of the GUI application, other objectives need to be accomplished in the first place in order to satisfy the ones mentioned above. These include:

- Familiarize with Python programming language and its main libraries such as NumPy, SciPy and Matplotlib, as well as Object-Orientated Programming (OOP) to work with classes and objects.
- Make use of PyQt5 and Qt Designer in order to build the GUI interface in a simple and intuitive way.
- Learn about the Pandas library and its multi-level indexing to find an appropriate way of storing the spikes information.
- Learn the basics of code benchmarking to time determined methods when running the code in order to detect which actions take more time.

A more detailed explanation of each of these terms will be given in the following sections.

2. Methodology

2.1. Background

To understand the purpose of this work, an outline of what CMOS MEA systems are, which spike detection methods are most commonly used, which are the basic features of Python programming language and what GUI applications are is given.

2.1.1. CMOS MEA systems

CMOS MEA systems are designed to offer a greater insight into how neural activity of single cells or cell tissue takes place. The study of this neural activity is based on the principle of Action Potentials (APs).

Nerve cells contain ion channels which selectively enable the permeation of certain ions such as sodium or potassium. APs are the elementary electrical signal in biological systems and consist of a transient change of the membrane potential in these cells. In a steady state, this potential has a value determined by the ion with highest conductance, usually potassium, which is of around -70 mV. After a voltage drop over a certain threshold caused by either a biological or an artificial stimulus, the Na-channels open and depolarization is further increased. Approximately 1 ms later, these Na-channels are closed again and, after a short delay, the K-channels are opened in turn. These repolarize the membrane and remain open even after reaching the steady-state potential. After a refractory period the K-channels are closed again and a similar situation is achieved as before [3]. Figure 1 shows what one of these transients looks like.

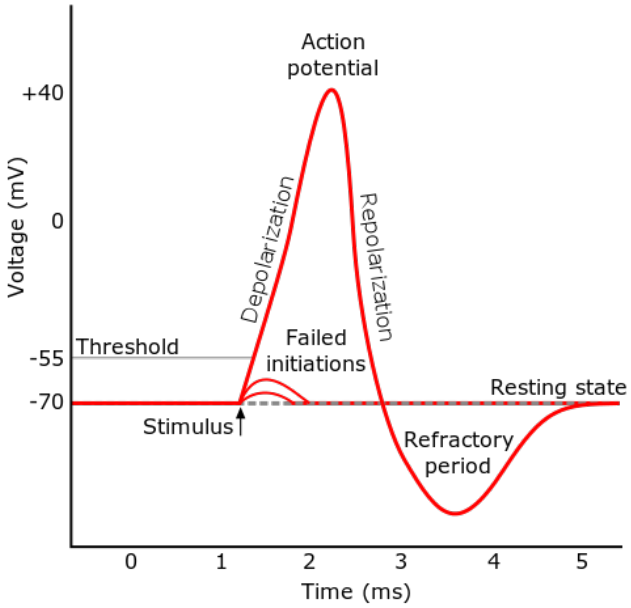


Figure 1: Diagram of an Action Potential and its different stages (Source: https://en.wikipedia.org/wiki/Action_potential).

Most high resolution CMOS MEAs contain between 1000 and 10 k active sites and they can be divided in two groups: those with highly resolved spatial interfacing selectability and those with real imaging capability such as the one used. This means the entire array is read out and the results are images or movies of the neural tissue activity. The number of channels equals the number of sites or is at least of the same order of magnitude and the array readout uses multiplexing. A high bandwidth is also required in the array periphery. The CMOS MEAs used in this case consist of 65 x 65 sensor sites and 32 x 32 stimulation sites, and the sampling rate at which the data is captured is of 20 kHz. The bit depth is of 16 bits. The aforementioned structure results in a total of 4225 recording sites and 1024 stimulation sites, though operation of a subset of the entire area is possible as well. Other parts of the system include the acquisition and control board and the digital interface board. Once the data is acquired from the system it is stored in HDF5 files with the .cmcr extension. This requires large amounts of space, as for a sampling rate of 20 kHz the total data stream is of (16 bit x 4225 x 20 kHz), which is approximately 1.35 Gbit/s. A diagram and pictures of a similar system can be seen in Figure 2. Until recently most attention has been put on recording and little on stimulation, but it is in fact of equal importance. Stimulation is carried out by feeding currents at the wanted sites into the neural tissue, though sometimes voltages are also applied. In most CMOS and non-CMOS MEAs sites are used in a bi-directional manner, but this requires a ”blanking time” after the stimulation pulse is applied. In this case, it has been proven that recording and stimulation can be carried out simultaneously [1], [4].

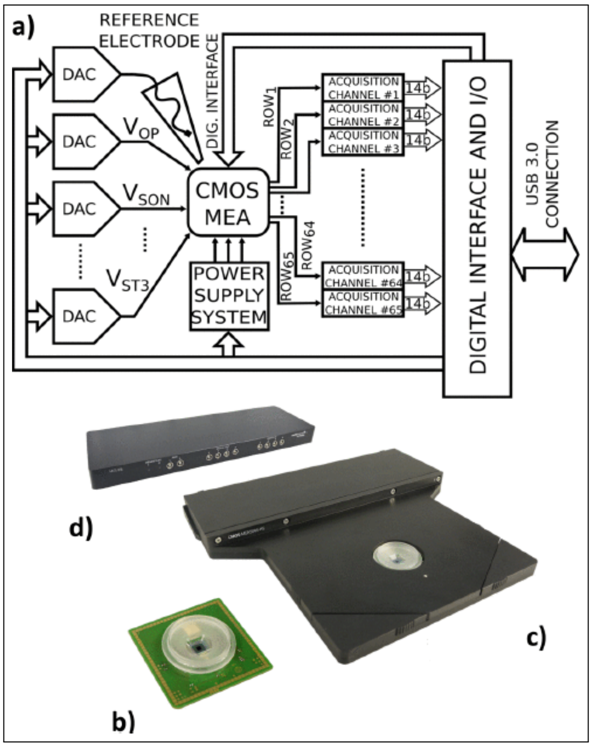


Figure 2: a) Block diagram of the entire MEA measurement system, b) CMOS-based MEA chip on PCB carrier with chamber, c) acquisition and control board, d) digital interface board (from Bertotti et al., 2014, [1]).

2.1.2. Spike detection methods

As said previously, these systems require spike detection methods in order to detect when and where the neural activity has taken place and to be able to understand the behaviour of the neural cells or tissue. Aspects to have in mind when choosing an appropriate method include:

- How fast it is.
- How complex it is.
- How much memory it requires.
- If it is unsupervised or not.
- If it is real-time.
- How successful it is.
- Where in the system it needs to be implemented.

Some of the existing methods are:

1. *Absolute Value Thresholding (Abs)* ([2], [5]): being one of the most simple and commonly used, it consists of applying a threshold to the absolute value of the signal. The threshold is usually obtained the following way:

$$Th = C_A \sigma_N, \quad \sigma_N = \text{median}\left\{\frac{|x(n)|}{0.6745}\right\}, \tag{1}$$

   where C_A is a positive constant which is usually set to around 4 as in [5] and σ_N is an estimate of the standard deviation of the noise.

2. *Nonlinear Energy Operator (NEO)* ([2], [5]): the NEO ψ is defined in discrete time as:

$$\psi[x(n)] = x^2(n) - x(n+\delta) \cdot x(n-\delta), \tag{2}$$

   where $\delta \in [1..10]$ and is normally set to 1 as in [5]. Thresholding is then performed using an integer multiple of the mean value of $\psi[x(n)]$. This method has the advantage of taking into account the frequency and not only the amplitude of the signal.

However, the actual system we are talking about counts so far with a software that uses a different and more complex spike sorting algorithm. As the CMOS MEA chips count with a higher number of sensor sites than of actual neurons, an *Independent Component Analysis (ICA)* can be carried out as an "unmixing" of the raw data. The result of this is a set of signals each of which will ideally correspond to a single neuron. As the algorithm tends to work worse if there are many sensors to be unmixed at the same time, a different algorithm first divides the sensors on the chip into smaller regions and computes the ICA for each of these regions. This algorithm reduces the dimensionality of the data and allows to process several regions in parallel on one or several computers [6].
Other algorithms such as the *Stationary-Wavelet-Transform Product (SWTP)* [5] or one based on a local energy measure (*LocEn*) [2] are presented in the articles mentioned. Every one of these has to prioritize some aspects over others and are more or less appropriate for a determined system depending on the conditions and purposes of each.

2.1.3. Python programming language

Python was first released in 1991 and is one of the most popular programming languages nowadays. Its philosophy makes special emphasis on a syntax that makes code easily readable. Another one of its most characteristic aspects is that it is developed under an open-source license, making it freely usable and distributable. A very large number of tutorials can be found online and it can be easy to learn at a basic level even for first time programmers [7].
Another common Python feature is its multiple programming paradigms, being one of them Object-Orientated Programming (OOP). This paradigm provides a means of structuring programs so that properties and behaviours are grouped into individual objects. It is an approach for modeling real-life items and relations between them, being in this case software objects which have certain data associated to them and can perform determined functions. Next is a brief explanation of the two key elements of OOP [8]:

- Classes: each object is an instance of some *class*. Classes are used to create data structures that contain information about the objects that belong to a specific class. They offer a structure and determine which properties of an object belonging to this class should be specified, but do not provide any real content themselves. Classes can be thought as the skeleton or idea for how an object should be defined.

- Objects: *objects* (or *instances*) are members of a specific class. They have the properties defined by the class but with actual values specific for them. Different objects from the same class will have the same properties but with their own values. These will differentiate one from another but will make them belong to this class and not to another one. Functions associated to any object are called methods.

Another key element of Python are its standard and external libraries. Some of the most commonly used are the following:

- NumPy [9]: being a fundamental package for numerical computation, it contains features such as an N-dimensional array object, complex functions and useful linear algebra. Arbitrary types of data can be defined and it is very efficient as a multi-dimensional data container as well as for its scientific purposes.

- SciPy [10]: it provides many useful numerical routines such as those for numerical integration, interpolation, optimization or statistics.

- Matplotlib [11]: being the most popular Python 2D plotting library, it produces figures in a number of different formats and interactive environments across platforms. It can generate plots, histograms, scatter plots, etc. with only a few lines of code.

2.1.4. GUI applications

Graphical User Interface (GUI) applications are Human Computer Interfaces (HCIs) that use windows, icons and menus and can be manipulated by devices such as a mouse or by touch. They differ from Command Line Interfaces (CLIs) in that these use exclusively text and can only be accessed by means of a keyboard. GUIs make computer operation more intuitive and easier and provide the user with a clear feedback about the effect of each action. They are also flexible meaning that they allow a number of different objects or instances to be displayed at the same time. The most common combination of elements in a GUI is the Windows-Icons-Menus-Pointer (WIMP) paradigm, though many others have been used after it. This type of interaction presents information organized in windows and represented with icons. Commands are compiled together in menus and actions are performed making gestures with the pointer [12].
A common GUI design pattern is the Model-View-Presenter (MVP) pattern. Here, the View and the Model are clearly separated and all presentation logic is pushed to the Presenter. Each of the parts consist of the following [8]:

- Model: it is the interface defining the data to be displayed or acted upon in the user interface.
- View: it is a passive interface that displays data (the model) and routes user commands or events to the presenter to act upon the data.
- Presenter: it acts upon the model and the view. It retrieves data from repositories (the model) and formats it for display in the view.

A diagram of this pattern is presented in Figure 3.

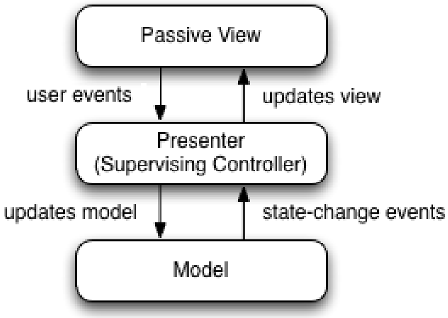


Figure 3: Diagram of the Model-View-Presenter (MVP) architecture pattern (Source: https://en.wikipedia.org/wiki/Model-view-presenter).

Python has a large number of GUI frameworks, being one of these PyQt [13]. PyQt is a blending of Python programming language and the Qt library [14] and is a multiplatform toolkit which runs on all major Operating System such as Unix or Windows. Qt Designer is a Qt tool for designing and building GUIs which allows to design widgets, dialog boxes and main windows.

2.2. Proposed method

Whereas many recent studies prioritize applying the spike detection method in close proximity to the recording sites to reduce the transmission data bandwidth, in this case we are working with already recorded and stored data in order to focus on the GUI application itself. As shown in Table 1 with experimental results provided in [2], from the two most common methods the *Absolute Value Thresholding (Abs)* has a lower error rate. As it is also more simple than the *Nonlinear Energy Operator (NEO)*, this one will be preferred over the two. In respect to the *LocEn* method, it has been considered too complex to be implemented in the first place and left for further consideration in the case of continuing to develop the application in the future.

Table 1: Error rates of different spike detection methods (from Dragas et al., 2013, [2])

	Abs	*NEO*	*LocEn*
Error Rate (e)	39.91%	41.62%	31.89%

In what refers to the creation of the GUI application, the mentioned PyQt toolkit, more specifically PyQt5, and Qt Designer have been chosen for its development in Python. The Operating System to be used is Linux and the Integrated Development Environment (IDE) chosen for creating the Python application is PyCharm. This has certain advantages over other IDEs such as:

- Coding assistance and analysis, including code completion and syntax and error highlighting among others.
- Project and code navigation with specialized project and file structure views and quick jumping between files, classes, etc.
- Refactoring including renaming of files, classes, functions, local/global variables, etc.

However, the preferred way of running the application and certain parts of it in this case will be IPython. IPython (Interactive Python) [15] is an interactive command-line terminal for Python. It is easy to use and flexible for trying out specific commands and making quick changes to them, which makes the whole process of creating the application faster. The project will also be uploaded to GitLab, a web-based platform for version control and collaborative software development in which developers can host and review code, manage projects and build software.
For the aspect of the application, a main window created in Qt Designer will appear when the application is run containing a menu which will make it possible to open different .cmcr files in different tabs. Once a file is loaded, a timeplot for the first pixel coordinate will appear along with a widget offering different possibilities. These will include selecting a different coordinate, filtering the data and computing

the spikes of the signal. From this window there will also be the possibility of opening a new window of a different type. This second window will show the filtered data of the whole array for the first time sample of the recorded data. A widget will offer the possibility of presenting the data either as a 2D grid plot or as a 3D plot. The widget will also incorporate the possibility of selecting a subarea of the whole array and a it will offer a number of different ways of plotting the spikes. The results acquired from computing the spikes will also be presented in some tables, which will show the coordinates with most spikes and the pixels with spikes occurring at the same or proximate time samples.
All these tasks will have been carried out making use of the main Python libraries described previously, as well as other specific ones such as Qt and Pandas [16]. This last one provides complex data structures and hierarchical axis indexing and will be useful for storing the peaks information which contains multiple indexes such as the x and y pixel coordinates.

2.3. Structure of work

With the above mentioned, the structure of the work to be done results in the following:

1. Familiarize with the basic Linux shell commands and with IPython's main features, as well as PyCharm's, and create an empty project. Upload the project to GitLab.

2. Create the main window in Qt Designer with an appropriate menu. Create a class associated to this window and make it able to open different .cmcr files in different tabs.

3. Create the timeplot widget in Qt Designer and its corresponding class and add the necessary buttons and features. Make it able to interpret the data in the HDF5 file and plot the raw data for the first pixel coordinate. Add the necessary methods for it to offer the following:
    - Give information about the dataset, meaning coordinates and time range values.
    - Possibility of selecting a specific pixel coordinate to be plotted.
    - Show relevant information about the selected coordinate such as mean, peak to peak, standard deviation and maximum and minimum values of the signal.
    - Possibility of choosing between plotting either raw or filtered data.
    - Making it able to compute spikes for a given threshold and for the specified coordinate and plot them.
    - Possibility of changing this threshold value.
    - Possibility of opening a new window of the same type and for the same dataset to be able to compare more than one pixel coordinate at the same time.
    - Possibility of opening a grid plot window.

4. Create the grid plot widget in Qt Designer with the necessary buttons and features. Create its corresponding class and plot the filtered data of the whole array for the first time sample. Add the necessary methods for it to offer:
    - Possibility of choosing between 2D plot and 3D plot.
    - Possibility of selecting a specific time sample for the array data to be plotted.
    - Making it able to compute spikes for a given threshold and for all pixels in the array and plot the pixels with spikes or the number of spikes in each pixel.
    - Possibility of selecting a subarea of the array and computing spikes for the specified area.

- Possibility of selecting a specific range from the original time range and compute spikes for the specified range.
- Possibility of selecting a peaks time plot, which means plotting separately each time sample containing spikes somewhere within the array and making it able to navigate between these time samples.

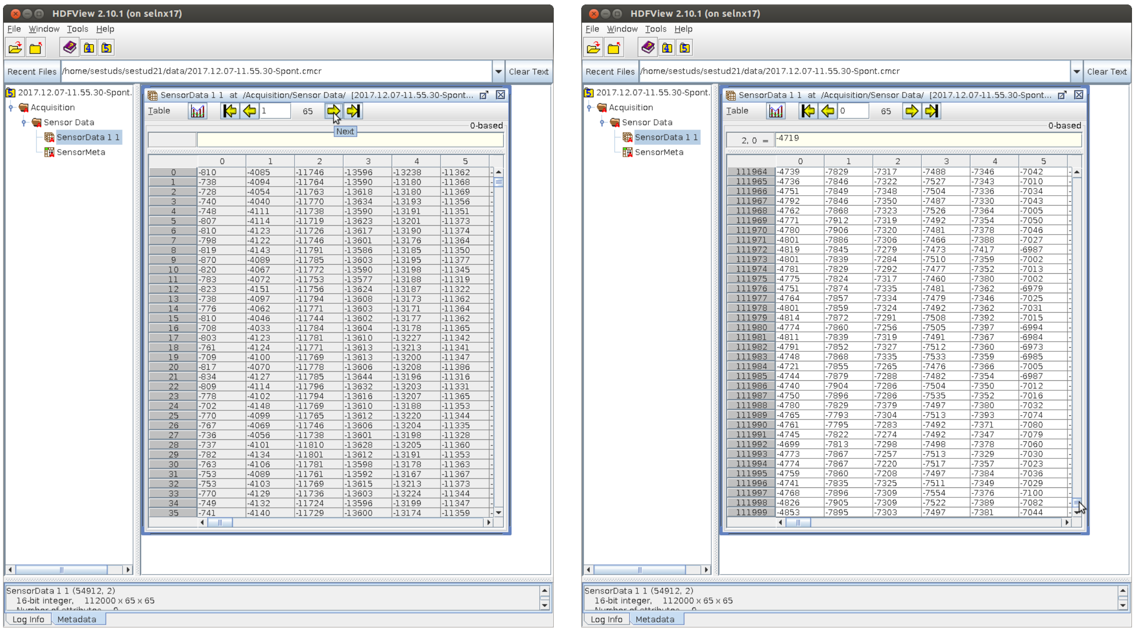
5. Create a table window to appear once spikes in the previous widget are computed. This table window will show all the coordinates with peaks and the number of peaks in each of them. It will offer:
    - Possibility of ordering the table by any of the coordinates or by the number of peaks and of navigating from one row to another by the use of the keyboard arrows.
    - Possibility of selecting one of the coordinates with peaks and have it highlighted on the grid plot, as well as showing a new table with all the time samples with peaks for the selected coordinate.
    - Select one of the time samples in this last table and show a third table with all the pixel coordinates with spikes at this time sample or a near one. Have these coordinates highlighted on the grid plot.

3. Development

This thesis has been developed between March and September of 2019 in the Chair of Sensor and Actuator Systems at TU Berlin. The following section provides a description of the work done during this time.

3.1. Environment setup

For the development of this work, access to a laboratory with computers counting with the necessary software has been provided. As mentioned before, these computers work with the Linux Operating System. Because of lack of previous knowledge about this OS, a first research to understand how the Linux shell works and what are its basic commands had to be done. At the same time, as IPython was going to be used during the development of the application for making trials and running small parts of the code, an initialization to this interactive shell was needed as well. The version to be used was IPython 5.5.0, and the version of Python was 3.6.8. To know with what type of data we were going to deal, HDFView was used, more precisely version 2.10.1. This software is suitable for exploring Hierarchical Data Format (HDF) files and the groups and datasets contained within. Figure 4a shows the result of opening one of the .cmcr acquisition files in HDFView. The columns in the table represent the values of one of the two coordinates, and to navigate between values of the other coordinate the arrows shown by the pointer are used. As it can be seen here, these values range between 0 and 65. Figure 4b shows the extent of the time sample range, going from 0 to 112000.



/home/sestuds/sestud21/data/2017.12.07-11.55.30-Spont.cmcr

SensorData 1 1 at /Acquisition/Sensor Data/ [2017.12.07-11.55.30-Spont...

1 65

	0	1	2	3	4	5
0	-810	-4085	-11746	-13596	-13238	-11362
1	-738	-4094	-11764	-13590	-13180	-11368
2	-728	-4054	-11763	-13618	-13180	-11369
3	-740	-4040	-11770	-13634	-13193	-11356
4	-748	-4111	-11738	-13590	-13191	-11351
5	-807	-4114	-11719	-13623	-13201	-11373
6	-810	-4123	-11726	-13617	-13190	-11374
7	-798	-4122	-11746	-13601	-13176	-11364
8	-819	-4143	-11791	-13586	-13185	-11350
9	-870	-4089	-11785	-13603	-13195	-11377
10	-820	-4067	-11772	-13590	-13198	-11345
11	-783	-4072	-11753	-13577	-13188	-11319
12	-823	-4151	-11756	-13624	-13187	-11322
13	-738	-4097	-11794	-13608	-13173	-11362
14	-776	-4062	-11771	-13603	-13171	-11364
15	-810	-4046	-11744	-13602	-13177	-11362
16	-708	-4033	-11784	-13604	-13178	-11365
17	-803	-4123	-11781	-13610	-13227	-11342
18	-761	-4124	-11771	-13613	-13213	-11341
19	-709	-4100	-11769	-13613	-13200	-11347
20	-817	-4070	-11778	-13606	-13208	-11386
21	-834	-4127	-11785	-13644	-13196	-11316
22	-809	-4114	-11796	-13632	-13203	-11331
23	-778	-4102	-11794	-13616	-13207	-11365
24	-702	-4148	-11769	-13610	-13188	-11353
25	-770	-4099	-11765	-13612	-13220	-11344
26	-767	-4069	-11746	-13606	-13204	-11335
27	-736	-4056	-11738	-13601	-13198	-11328
28	-737	-4101	-11810	-13628	-13205	-11360
29	-782	-4134	-11801	-13612	-13191	-11353
30	-763	-4106	-11781	-13598	-13178	-11363
31	-753	-4089	-11761	-13592	-13167	-11367
32	-753	-4103	-11769	-13615	-13213	-11373
33	-770	-4129	-11736	-13603	-13224	-11344
34	-749	-4132	-11724	-13596	-13199	-11347
35	-741	-4140	-11729	-13600	-13174	-11359

SensorData 1 1 (54912, 2)
16-bit integer, 112000 x 65 x 65

(a) HDFView showing extent of pixel coordinates

/home/sestuds/sestud21/data/2017.12.07-11.55.30-Spont.cmcr

SensorData 1 1 at /Acquisition/Sensor Data/ [2017.12.07-11.55.30-Spont...

0 65

2, 0 = -4719

	0	1	2	3	4	5
111964	-4739	-7829	-7317	-7488	-7346	-7042
111965	-4736	-7846	-7322	-7527	-7343	-7010
111966	-4751	-7849	-7348	-7504	-7336	-7034
111967	-4792	-7846	-7350	-7487	-7330	-7043
111968	-4762	-7868	-7323	-7526	-7364	-7005
111969	-4771	-7912	-7319	-7492	-7354	-7050
111970	-4780	-7906	-7320	-7481	-7378	-7046
111971	-4801	-7886	-7306	-7466	-7388	-7027
111972	-4819	-7845	-7279	-7473	-7417	-6987
111973	-4801	-7839	-7284	-7510	-7359	-7002
111974	-4781	-7829	-7292	-7477	-7352	-7013
111975	-4775	-7824	-7317	-7460	-7380	-7002
111976	-4751	-7874	-7335	-7481	-7362	-6979
111977	-4764	-7857	-7334	-7479	-7346	-7025
111978	-4801	-7859	-7324	-7492	-7362	-7031
111979	-4814	-7872	-7291	-7508	-7392	-7015
111980	-4774	-7860	-7256	-7505	-7397	-6994
111981	-4811	-7839	-7319	-7491	-7367	-6984
111982	-4791	-7852	-7327	-7512	-7360	-6973
111983	-4748	-7868	-7335	-7533	-7359	-6985
111984	-4721	-7855	-7265	-7476	-7366	-7005
111985	-4744	-7879	-7288	-7482	-7354	-6987
111986	-4740	-7904	-7286	-7504	-7350	-7012
111987	-4750	-7896	-7286	-7535	-7352	-7016
111988	-4780	-7829	-7379	-7497	-7380	-7032
111989	-4765	-7793	-7304	-7513	-7393	-7074
111990	-4761	-7795	-7283	-7492	-7371	-7080
111991	-4745	-7822	-7274	-7492	-7347	-7079
111992	-4699	-7813	-7298	-7498	-7378	-7060
111993	-4773	-7867	-7257	-7513	-7329	-7030
111994	-4774	-7867	-7220	-7517	-7357	-7023
111995	-4759	-7860	-7208	-7497	-7384	-7036
111996	-4741	-7835	-7325	-7511	-7349	-7029
111997	-4768	-7896	-7309	-7554	-7376	-7100
111998	-4826	-7905	-7309	-7522	-7389	-7082
111999	-4853	-7895	-7303	-7497	-7381	-7044

SensorData 1 1 (54912, 2)
16-bit integer, 112000 x 65 x 65

(b) HDFView showing extent of time range

Figure 4: Aspect of HDFView when one of the .cmcr acquisition files is opened.

HDFView also gives relevant information about how the data in the file has been captured. In this case, it says the sample period is of 50 µs, which corresponds to the sampling frequency of 20 kHz mentioned before. It also says the values are integers with a bit depth of 16 bits.

In respect to the rest of the software used, PyCharm version 2019.2 and Designer with Qt version 5.9.5 were the most relevant ones for the creation of the application itself and will be described in detail in the following section. The main Python libraries used and their versions were NumPy 1.16.3, SciPy 1.2.1 and Matplotlib 3.0.3. The project has also been managed using GitLab version 2.17.1.

3.2. Application development

As said, a project folder for the application was first created and the project was imported to GitLab. Next is a brief explanation of how the application has been developed with this as a starting point and of which parts compose it.

3.2.1. Analysis class

The first .py file (program file written in Python) that was created was meant to be the one responsible for carrying out the most complex functions on the data, without having any direct relation with the design of the application. This file was also made to be able to be run on its own and therefore make it easier to make trials with the data using the IPython shell. For this an Analysis class was created with the following being its most relevant methods:

- __init__: the HDF5 file is read and its data saved as a NumPy ndarray, in this case a four-dimensional one (x coordinate, y coordinate, time and voltage), as well as the data path. The coordinate and time minimum and maximum values are saved as global variables to be used in the rest of the methods.
- filt_alldata: a High-Pass Bessel filter is applied to the signal of each pixel in the array and the filtered data saved in a new NumPy ndarray. The values for the signals of all pixels in the 57th column are masked because of an incorrect reading from the system.
- compute_allpeaks: a special SciPy function is used for finding peaks in the signal of each pixel. The equation used for finding the threshold which is calculated for each pixel signal is the one described in (1). For a transient to be considered a spike its height has to exceed this threshold and there has to be a minimum distance of 8 (determined after trial and error) samples between two neighboring peaks. For each spike, the coordinates, time at which it takes place and height are saved. After computing all peaks, this information is stored as a Pandas Series, where the x and y coordinates and the times are converted to a MultiIndex and the heights are saved as the actual values.
- update_peaksbinary: when this method is called, the Pandas Series for the computed peaks is analyzed and a 65 x 65 array is created showing for each pixel if there are any peaks (1) or not (0).
- update_peaksnumb: similarly as before, when this method is called the Pandas Series for the computed peaks is analyzed and a 65 x 65 array is created showing for each pixel the number of peaks.

A timer function is also incorporated to the file outside the Analysis class to be able to time any method and therefore know how much time it takes for it to be run. The timer function is incorporated in this file and not in another one because these methods are the ones that take most time, and because it is useful for when the file is run on its own.

3.2.2. PyQtApp class

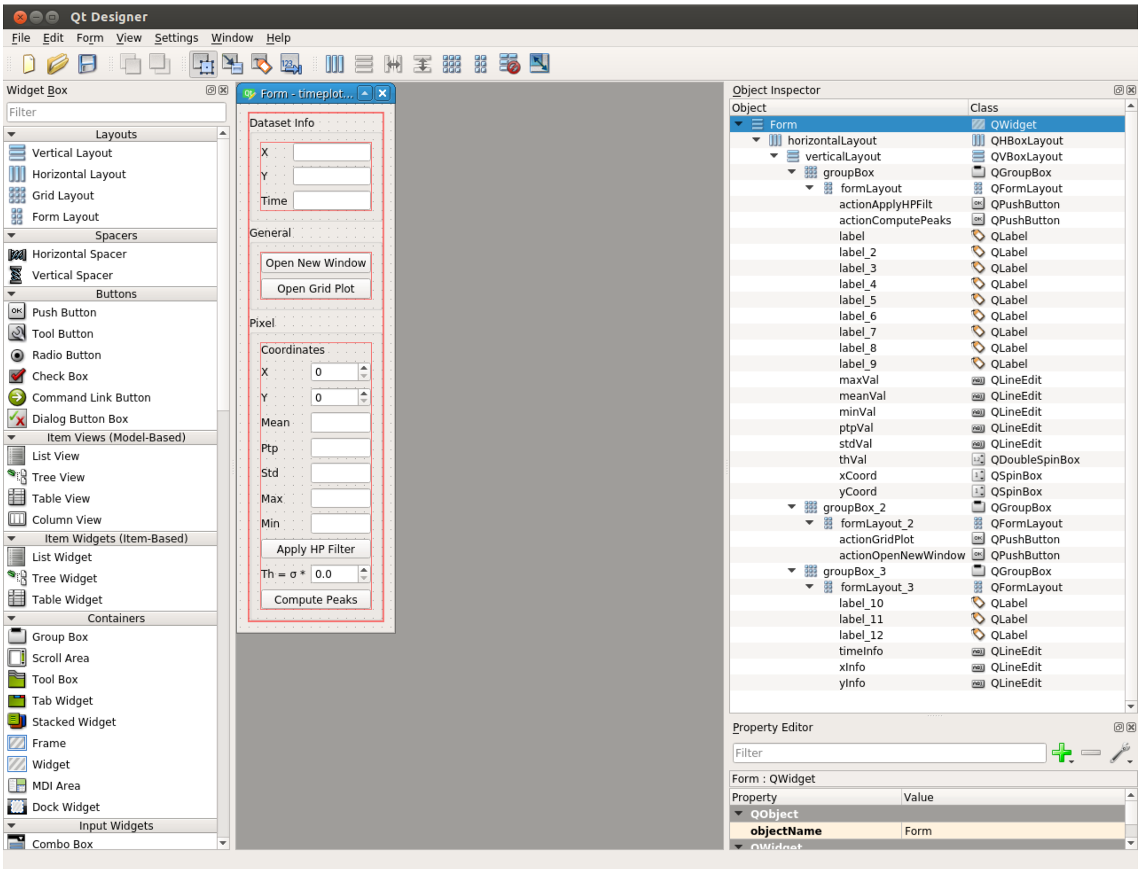
Additionally to the previous file, another .py file for the main aspects of the app was created. At the same time, a main window was created in Qt Designer with a File menu and a tab widget. All main windows and widgets created in Qt Designer are saved as .ui files. The PyQtApp class was created in the .py file with the following main methods:

- __init__: the main window .ui file is loaded and the tabs in the tab widget are made closable in case the user wants to remove them.

- on_actionLoad_triggered: made to respond to the Load button in the File menu of the main window, this method makes a File Dialog appear for the user to navigate between folders and select a .cmcr file to load in a new tab.

3.2.3. TimeplotWidget class

As the next step for the development of the app, another .py file was created to manage the timeplot widget. This widget was created simultaneously in Qt Designer and saved as a .ui file, and its final aspect is shown in Figure 5.



Object	Class
Form	QWidget
horizontalLayout	QHBoxLayout
verticalLayout	QVBoxLayout
groupBox	QGroupBox
formLayout	QFormLayout
actionApplyHPFilt	QPushButton
actionComputePeaks	QPushButton
label	QLabel
label_2	QLabel
label_3	QLabel
label_4	QLabel
label_5	QLabel
label_6	QLabel
label_7	QLabel
label_8	QLabel
label_9	QLabel
maxVal	QLineEdit
meanVal	QLineEdit
minVal	QLineEdit
ptpVal	QLineEdit
stdVal	QLineEdit
thVal	QDoubleSpinBox
xCoord	QSpinBox
yCoord	QSpinBox
groupBox_2	QGroupBox
formLayout_2	QFormLayout
actionGridPlot	QPushButton
actionOpenNewWindow	QPushButton
groupBox_3	QGroupBox
formLayout_3	QFormLayout
label_10	QLabel
label_11	QLabel
label_12	QLabel
timeInfo	QLineEdit
xInfo	QLineEdit
yInfo	QLineEdit

Form : QWidget

Property	Value
QObject	
objectName	Form
QWidget	

Figure 5: Development of the timeplot widget and its different objects in Qt Designer.

A TimeplotWidget class was created in the .py file. Some of its main methods include:

- __init__: the timeplot widget .ui file is loaded and an Analysis object is created for the selected .cmcr file. The __init__ method of the Analysis class loads the data and creates the ndarray. A PlotWindow (a class which will be described later) object is created and the raw data of the first coordinate sent to one of its methods for a timeplot to be shown at the side the widget. The mean, peak to peak, standard deviation, maximum and minimum values for the selected pixel coordinate signal are calculated and displayed. Other global variables which will be needed for the following methods are also declared.

- on_actionApplyHPFilter_toggled: when the Apply HP Filter button in the timeplot widget is checked, the filtering method from the Analysis class is called and the filtered data instead of the raw data of the actual coordinate is sent to the same PlotWindow method for it to be plotted next to the widget. The mean, peak to peak, standard deviation, maximum and minimum values are refreshed.

- on_xCoord_valueChanged: the plot is updated whenever the user changes the value for the x coordinate, and so are the mean, peak to peak, standard deviation, maximum and minimum values. An equal method is used for the y coordinate.

- on_actionComputePeaks_toggled: when the Compute Peaks button is checked, the method for computing peaks in the Analysis class is called for it to find the peaks in the signal of the actual coordinate. These are then shown on the plot by means of a method in the PlotWindow class.

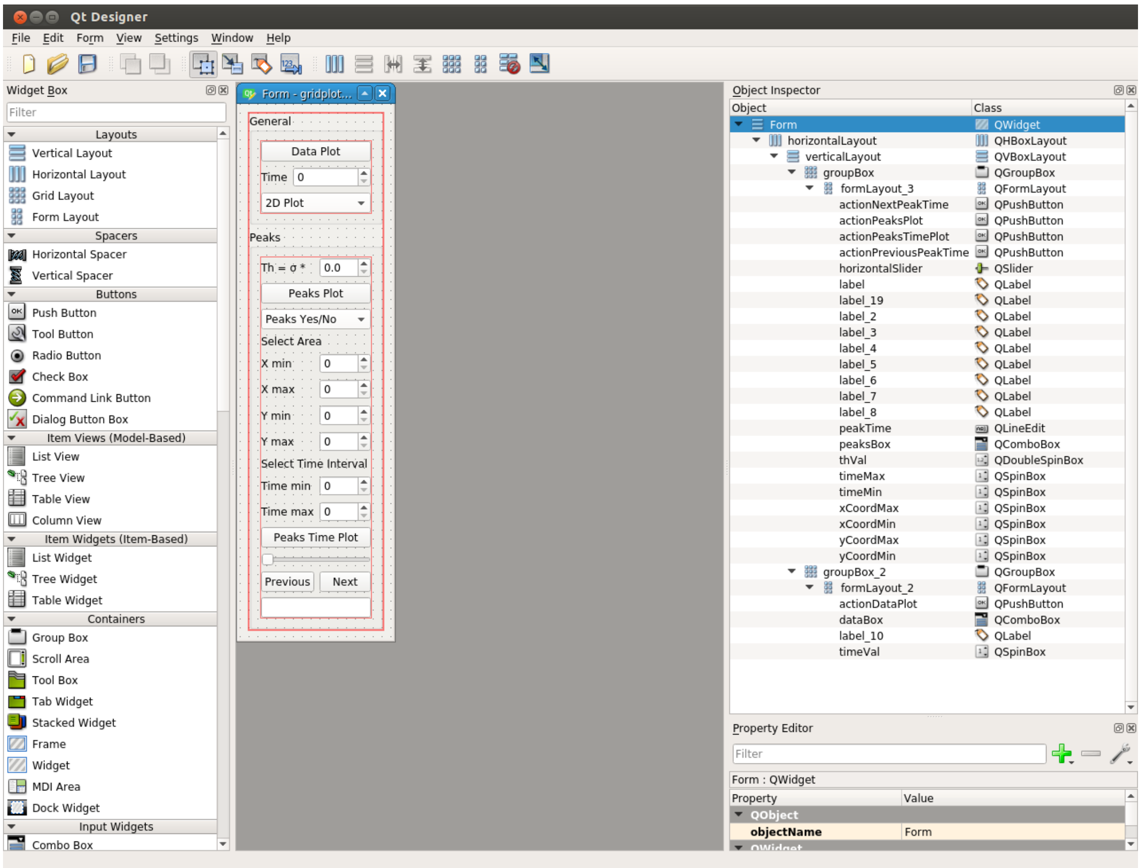
- on_thVal_valueChanged: the peaks are computed again for the new threshold value when it is changed by the user and the new peaks are plotted by the same method in the PlotWindow class.

- on_actionOpenNewWindow_clicked: when the Open New Window button is clicked, a new window of the same class and for the same dataset is opened.

- on_actionOpenGridPlot_clicked: when the Open Grid Plot button is clicked, a new type of window which will be explained next is opened. The data used for creating a GridplotWidget object is the filtered data of the whole array.

3.2.4. GridplotWidget class

After this, a .py file was created to manage the grid plot widget. At the same time, this widget was created in Qt Designer and saved as a .ui file, and its final aspect is shown in Figure 6. This widget is meant to appear in a new window when the previously mentioned Open Grid Plot button in the timeplot widget is clicked.



Object	Class
Form	QWidget
horizontalLayout	QHBoxLayout
verticalLayout	QVBoxLayout
groupBox	QGroupBox
formLayout_3	QFormLayout
actionNextPeakTime	QPushButton
actionPeaksPlot	QPushButton
actionPeaksTimePlot	QPushButton
actionPreviousPeakTime	QPushButton
horizontalSlider	QSlider
label	QLabel
label_19	QLabel
label_2	QLabel
label_3	QLabel
label_4	QLabel
label_5	QLabel
label_6	QLabel
label_7	QLabel
label_8	QLabel
peakTime	QLineEdit
peaksBox	QComboBox
thVal	QDoubleSpinBox
timeMax	QSpinBox
timeMin	QSpinBox
xCoordMax	QSpinBox
xCoordMin	QSpinBox
yCoordMax	QSpinBox
yCoordMin	QSpinBox
groupBox_2	QGroupBox
formLayout_2	QFormLayout
actionDataPlot	QPushButton
dataBox	QComboBox
label_10	QLabel
timeVal	QSpinBox

Form : QWidget

Property	Value
QObject	
objectName	Form
QWidget	

Figure 6: Development of the grid plot widget and its different objects in Qt Designer.

A GridplotWidget class was created in the .py file with the following as its main methods:

- __init__: the grid plot widget .ui file is loaded and the necessary global variables are declared. A new PlotWindow object is created and the filtered data of the whole array for the first time sample is sent to the appropriate PlotWindow method for it to be plotted next to the widget. An Analysis object is also created.

- on_actionDataPlot_clicked: when a different type of plot has been shown last and the Data Plot button is clicked, the filtered data of the whole array for the actual time sample is sent again to the PlotWindow method for it to be plotted.

- on_dataBox_indexChanged: depending on the selection of the user, the PlotWindow method will perform either a 2D grid plot of the array data or a 3D plot.

- on_timeVal_valueChanged: if the user selects a different time sample, the filtered data of the whole array for the selected time sample is sent to the PlotWindow method for it to be plotted.

- on_actionPeaksPlot_clicked: when the Peaks Plot button is clicked, the method for computing peaks in the Analysis class is called and performed for the signals of all pixels in the array. Both methods from the Analysis class previously described which create 65 x 65 arrays from the peaks information are also called. As the first type of peaks plot is selected by default, the array containing either 0 or 1 for each pixel depending on if there are peaks or not is sent to the appropriate PlotWindow method for it to be plotted.

- on_peaksBox_indexChanged: depending on the selection of the user, either the first or the second of the 65 x 65 arrays will be sent to the PlotWindow class for them to be plotted. The second array contains the number of peaks for each pixel.

- on_xCoordMin_valueChanged: if the user selects a different minimum x coordinate, the plot is updated to show only the pixels with an equal or higher x coordinate. If the peaks are computed again they will be calculated only for this region. Equal methods are used for the maximum x coordinate and the minimum and maximum y coordinates.

- on_timeMin_valueChanged: if the user selects a different minimum time sample, the peaks will be computed only for the selected range the next time either the Peaks Plot or the Peaks Time Plot buttons are clicked. An equal method is used for the maximum time sample.

- on_actionPeaksTimePlot_clicked: when the Peaks Time Plot button is clicked, a different method of the PlotWindow class is called and a plot showing where in the array there are peaks at the first time sample where there are peaks somewhere in the array is shown.

- on_horizontalSlider_valueChanged: when the user navigates through the horizontal slider, the different time samples with peaks are selected to be plotted in the same way as before by the PlotWindow method.

- on_actionPreviousPeakTime_clicked: when the Previous button is clicked, the previous time sample with peaks is selected for the same type of plot as before. An equal method is used for the Next button.

- on_ax_changed: if the user uses the zoom function, which is incorporated on the navigation toolbar at the bottom of the plot, to select a smaller area and then clicks on one of the plotting buttons on the widget, only the selected subarea will be shown and the peaks will be calculated only for the pixels within it.

- open_timeplot: if the user clicks on one coordinate or draws a rectangle with the mouse over more than one coordinate, a new window is opened showing the timeplot of the clicked pixel or the pixels within the drawn rectangle. This is done by another PlotWindow method.

- on_tableSel: if a coordinate from the first table of the table window which will be described later is selected by the user, this method highlights this pixel on the grid plot. Similar methods are used for the other two tables.

3.2.5. PlotWindow class

The plot window .py file was created during the development of the timeplot widget and was meant to perform all the plot-related tasks for both this and the grid plot widget. A main window containing the few necessary elements was created in Qt Designer and saved as a .ui file. The PlotWindow class was created in the .py file with the following main methods:

- __init__: the .ui file is loaded and the appropriate global variables are declared, including a canvas and a figure object, both of them classes from the Matplotlib library. A navigation toolbar object is also added at the bottom of the canvas.

- plot: when this method is called from the timeplot widget and the data, either raw or filtered, of the selected coordinate is given, a timeplot is created on the canvas.

- plot_peaks: this method is called from the timeplot widget when the Compute Peaks button is checked and the time values where peaks have been found by the Analysis method are given. These are marked with crosses on top of the previous plot.

- plotdata_2d: when this method is called from the grid plot widget and the filtered data of the whole array for a specific time sample is given, this data is plotted on a 65 x 65 grid. A colorbar object is added next to the canvas to show the range of different values in a specific color palette. The coordinates of a certain pixel will be printed below the canvas if the mouse is placed on top of it.

- plotdata_3d: the same data is plotted than for the previous method but in a 3D surface plot.

- plotpeaks_2d: this method is called when the Plot Peaks button from the grid plot widget is clicked. The 65 x 65 array to plot is given along with which one of the two types of plot has to be performed. For the first type the method will create a grid plot with two different colours, one meaning the pixel contains peaks and the other one meaning it does not. For the second type it will create a grid plot with a range of colours determining the number of peaks on each pixel. The plot performed for the Peaks Time Plot button on the grid plot widget is the same as this second one. The method will also highlight certain pixels if some coordinates from the tables on the table window explained next are selected by the user.

- plot_subplots: if a pixel coordinate from the grid plot is clicked or a rectangle is drawn over more than one of them, this method is called while opening a new window and the timeplots of all the selected pixel coordinates are drawn as subplots. The data to plot is the filtered data and already showing the peaks.

3.2.6. TableWindow class

After developing the grid plot widget, a .py file for creating a table window, which is meant to be opened when the Peaks Plot button in the grid plot widget is clicked, was created, along with a .ui file created in Qt Designer containing a table view. A TableWindow class was created in the .py file with the following main methods:

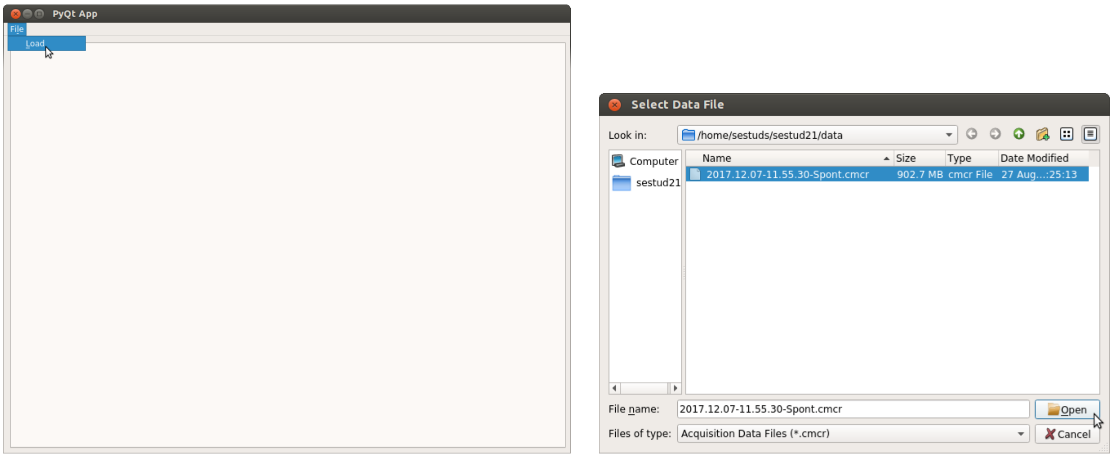
- __init__: the .ui file is loaded and the appropriate global variables are declared. A MyTableModel class is created separately as a table model which will be able to present the given data in the form of a table and an object of this class is created for the first table. The data presented on this table will include the x and y coordinates of the pixels with peaks and the number of peaks in each of them. By default it will be ordered showing the coordinates of the pixel with most peaks first.

- on_rowSel: when the user selects a row from the previous table, that is, a specific pixel coordinate, a new MyTableModel object is defined to create a second table to be shown next to the existing one. This new table will show all the time samples at which there are peaks for the selected pixel coordinate and their height.

- on_timeSel: when the user selects a row from the previous table, that is, a specific time sample, a new MyTableModel object is defined to create a third table to be shown next to the second one. This third table will show all the pixel coordinates with peaks at the selected time sample or a similar one, with a maximum distance of 5 time samples, and their height.

4. Results

In the following section, figures showing the different features of the application and the effects of the previously described actions are provided.
Once the application is run, the main window shown in Figure 7a appears, with a File menu that offers the possibility of loading a file. Once the Load button is clicked, the file dialog shown in Figure 7b appears. As the app works for files with the .cmcr extension, the possibility of viewing only these type of files is given.



Look in: /home/sestuds/sestud21/data

Name	Size	Type	Date Modified
2017.12.07-11.55.30-Spont.cmcr	902.7 MB	cmcr File	27 Aug...:25:13

File name: 2017.12.07-11.55.30-Spont.cmcr

Files of type: Acquisition Data Files (*.cmcr)

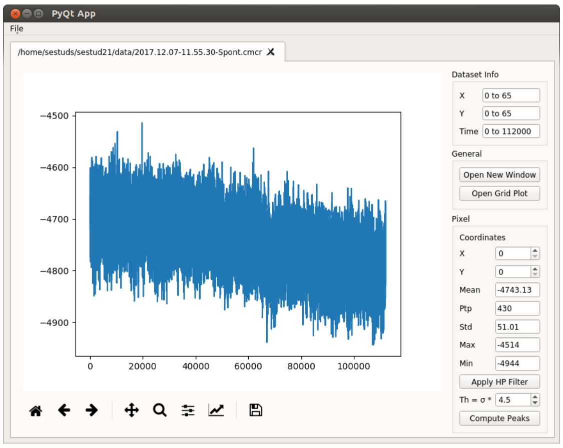
(a) App main window

(b) File dialog for loading a .cmcr file

Figure 7: Aspect of the app main window and how to load a .cmcr file.

4.1. Timeplot analysis results

Figure 8 shows the aspect of the app once a file is loaded. Here we can see the timeplot of the raw data for the first coordinate in the dataset. Next to the plot we can see the timeplot widget with the different functions it offers. The widget also gives information about the dataset, meaning the coordinate and time ranges.



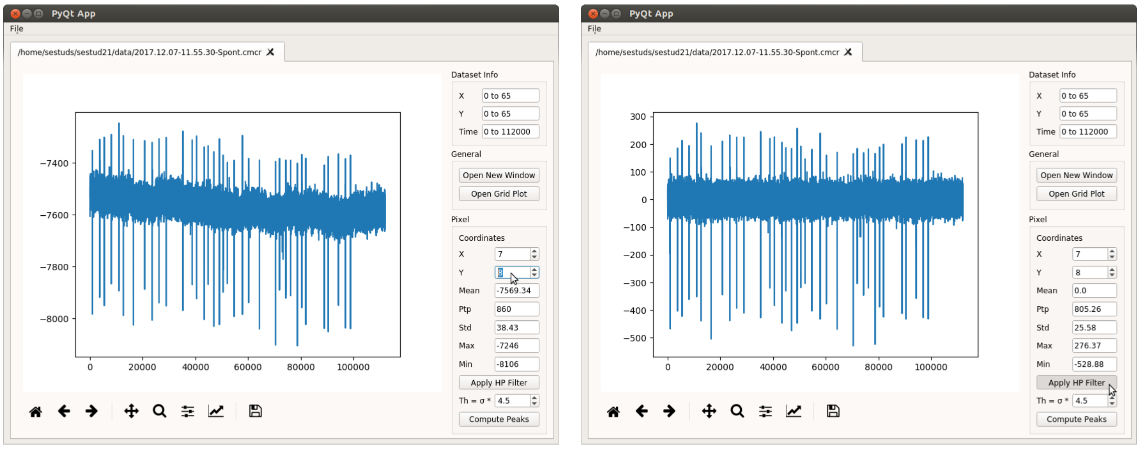
/home/sestuds/sestud21/data/2017.12.07-11.55.30-Spont.cmcr

X: 0 to 65
Y: 0 to 65
Time: 0 to 112000

X: 0
Y: 0
Mean: -4743.13
Ptp: 430
Std: 51.01
Max: -4514
Min: -4944
Th = σ * 4.5

Figure 8: Aspect of app once a .cmcr file is loaded.

The timeplot widget offers the possibility of selecting and plotting the data of a specific pixel coordinate, as we can see in Figure 9a, and shows its mean, peak to peak, standard deviation, maximum and minimum values. It also offers the option of plotting the filtered data of the selected coordinate by means of a checkable button. The filter applied is a High-Pass filter, which can easily be seen in Figure 9b, as the DC offset of the signal is removed.



/home/sestuds/sestud21/data/2017.12.07-11.55.30-Spont.cmcr

X 0 to 65
Y 0 to 65
Time 0 to 112000

X 7
Y 8
Mean -7569.34
Ptp 860
Std 38.43
Max -7246
Min -8106
Th = σ * 4.5

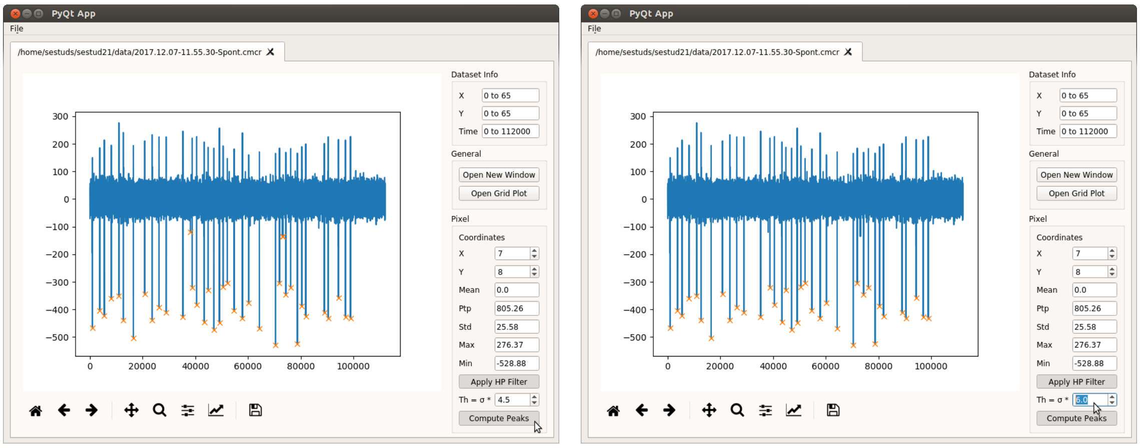
X 7
Y 8
Mean 0.0
Ptp 805.26
Std 25.58
Max 276.37
Min -528.88
Th = σ * 4.5

(a) Raw data timeplot

(b) Filtered data timeplot

Figure 9: Timeplot comparison of raw and filtered data for the selected pixel coordinate.

The timeplot widget contains another checkable button for computing and plotting the peaks of the actual pixel coordinate. This is done by thresholding the absolute value of the signal, and the threshold uses the formula (1). The value which is multiplied by the standard deviation of the noise of the signal can be changed and the difference between using a lower threshold value (Figure 10a) and a higher one (Figure 10b) is shown below.



/home/sestuds/sestud21/data/2017.12.07-11.55.30-Spont.cmcr

X 0 to 65
Y 0 to 65
Time 0 to 112000

X 7
Y 8
Mean 0.0
Ptp 805.26
Std 25.58
Max 276.37
Min -528.88
Th = σ * 4.5

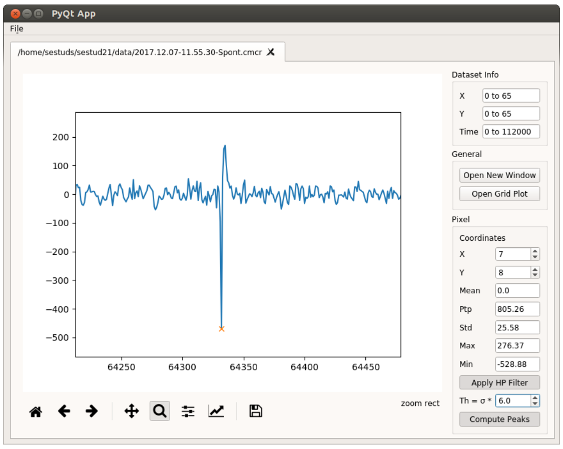
X 7
Y 8
Mean 0.0
Ptp 805.26
Std 25.58
Max 276.37
Min -528.88
Th = σ * 6.0

(a) Threshold = 4.5·σ

(b) Threshold = 6.0·σ

Figure 10: Comparison of computed spikes for two different threshold values.

Another functionality in every plot of the application is the possibility of applying a zoom to visualize a smaller area. This is incorporated as part of the navigation toolbar which appears below the plot. In Figure 11 we can see how this has been used in order to visualize an individual spike.



/home/sestuds/sestud21/data/2017.12.07-11.55.30-Spont.cmcr

Dataset Info

X: 0 to 65

Y: 0 to 65

Time: 0 to 112000

Pixel

Coordinates

X: 7

Y: 8

Mean: 0.0

Ptp: 805.26

Std: 25.58

Max: 276.37

Min: -528.88

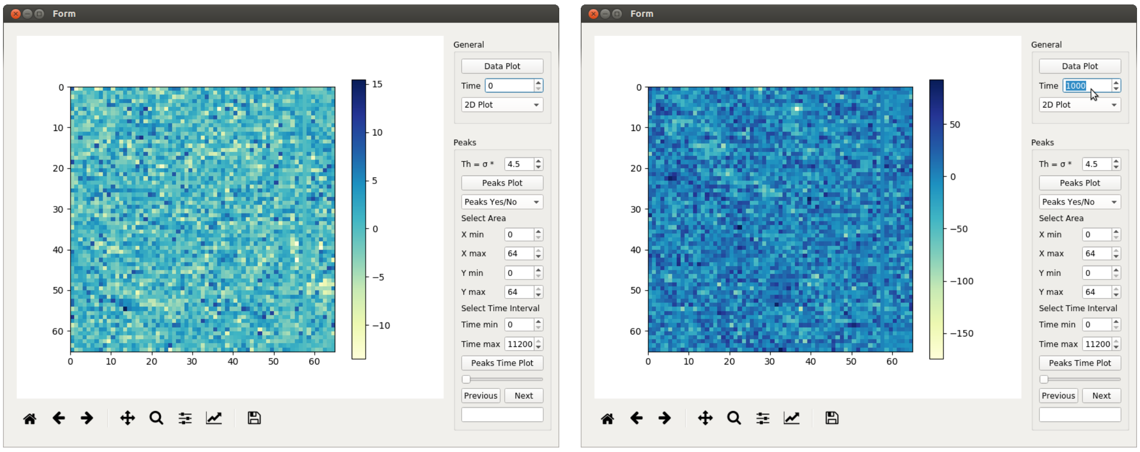
Th = σ *: 6.0

Figure 11: Effect of applying zoom for visualizing an individual spike.

Finally, the timeplot widget incorporates an Open New Window button which opens another window of the same type and for the same dataset in order to compare more than one pixel coordinate at the same time. It also contains the Open Grid Plot button for opening a type of window which will be described next.

4.2. Grid plot analysis results

Figure 12a shows the aspect of the grid plot window once the Open Grid Plot button in the previous window is clicked. Here we can see the grid plot of the filtered data for the whole array and for the fist time sample in the dataset. At the right of the grid plot we can see the grid plot widget with the different functions it incorporates. One of these is the possibility of selecting a specific time sample, as we can see in Figure 12b. The Data Plot button is offered to return to this type of plot from another one at any moment.



Time 0
2D Plot
Th = σ * 4.5
Peaks Yes/No
X min 0
X max 64
Y min 0
Y max 64
Time min 0
Time max 11200

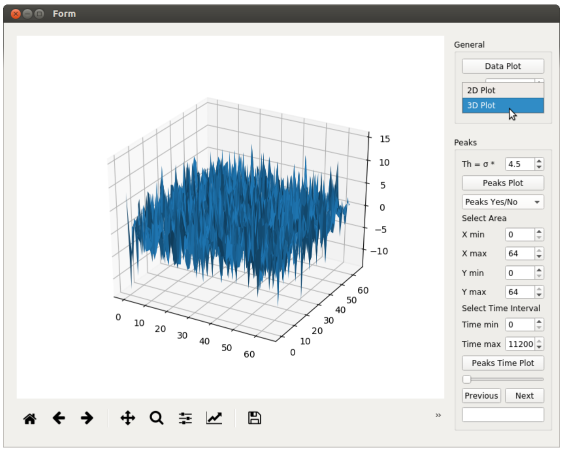
Time 1000

(a) Time = 0

(b) Time = 1000

Figure 12: Grid plot of filtered data at two different time samples.

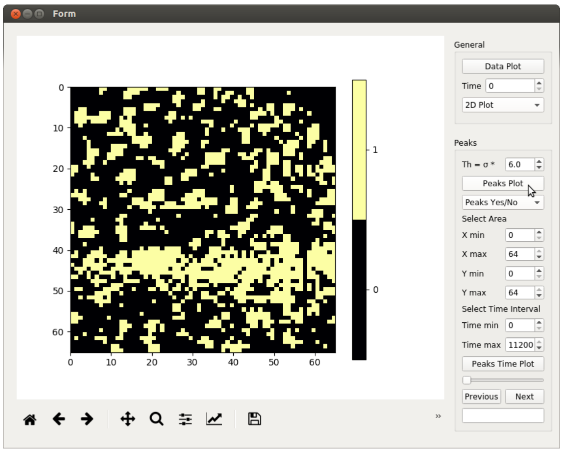
Another possibility that the grid plot widget offers is being able to choose between visualizing the data as a 2D plot or as a 3D plot. Although most functions will be carried out using the 2D plot this 3D plot can be seen in Figure 13 for the same data as in Figure 12a, that is, the filtered data for the first time sample in the dataset.



Th = σ * 4.5
Peaks Yes/No
X min 0
X max 64
Y min 0
Y max 64
Time min 0
Time max 11200

Figure 13: Effect of selecting 3D plot instead of 2D plot.

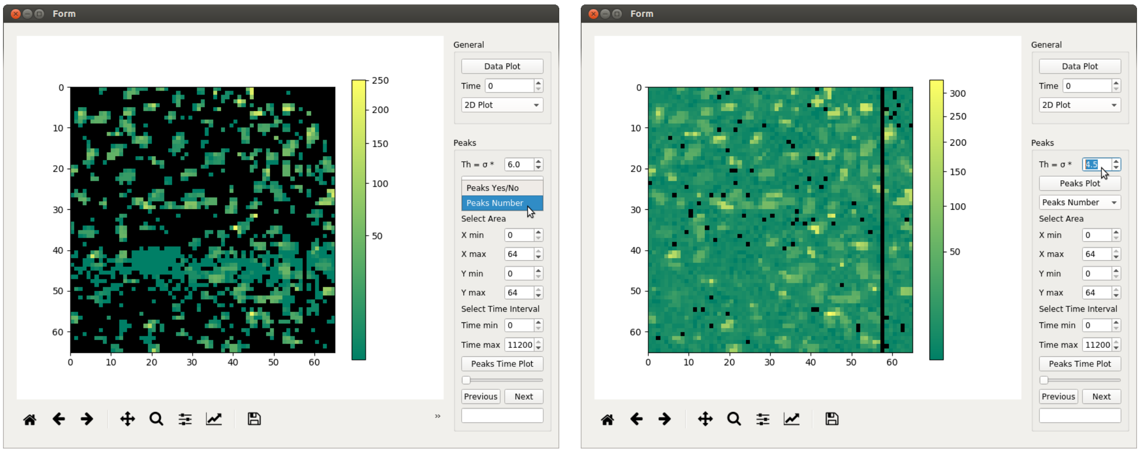
Another and one of the most relevant buttons the grid plot widget incorporates is the Peaks Plot button. When clicked it computes the peaks for the given threshold value and using the already mentioned method and creates the plot shown in Figure 14. This plot shows which pixels contain any spikes (in yellow) and which do not (in black). The Peaks Plot button, similarly to the Data Plot button, can be used to return to this type of plot at any time.



Time 0
2D Plot
Th = σ * 6.0
Peaks Yes/No
X min 0
X max 64
Y min 0
Y max 64
Time min 0
Time max 11200

Figure 14: Aspect of peaks plot showing if there are spikes or not for each coordinate.

Apart from the previous type of peaks plot, the widget offers the possibility to change to a second type, in which the number of peaks at each coordinate is plotted. The pixels without peaks are shown in black. The grid plot widget also makes it possible to change the threshold value, though for computing the peaks after this value has been changed the Peaks Plot button has to be clicked again. The difference between using a higher threshold value (Figure 15a) and a lower one (Figure 15b) for this type of plot is shown below.



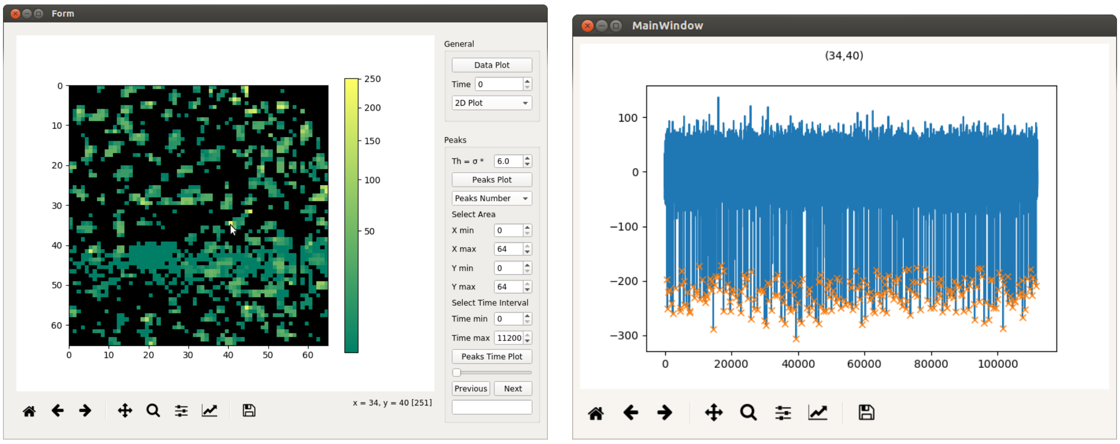
Time 0
2D Plot
Th = σ * 6.0
Th = σ * 4.5
Peaks Number
X min 0
X max 64
Y min 0
Y max 64
Time min 0
Time max 11200

(a) Threshold = 6.0·σ

(b) Threshold = 4.5·σ

Figure 15: Effect of changing the threshold value and selecting second type of peaks plot showing the number of spikes for each coordinate.

The grid plot widget has another functionality that consists of opening a timeplot window for a certain coordinate pixel when it is clicked. This works for all type of plots but becomes most useful with the peaks plot, as the coordinates with most peaks can be clicked for visualizing their timeplot. This is shown in Figure 16a, where one of the coordinates that show a highest number of peaks is clicked. The window that appears can be seen in 16b and shows that in fact the selected pixel coordinate presents a very high number of peaks. The timeplot window which appears is different to the initial one in that it already plots the filtered data and showing the peaks. It also differs in that it does not incorporate the timeplot widget.



Time 0

Th = σ * 6.0

X min 0

X max 64

Y min 0

Y max 64

Time min 0

Time max 11200

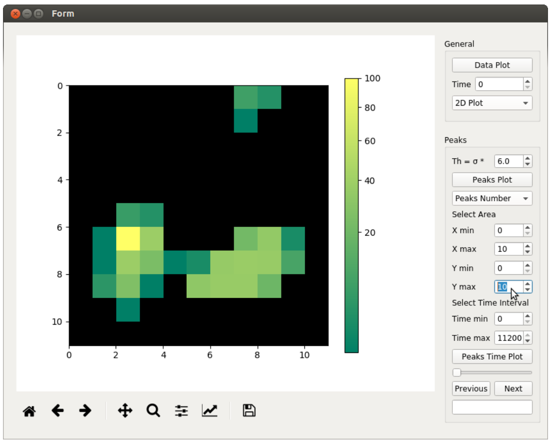
x = 34, y = 40 [251]

(a) Clicking on a pixel coordinate

(b) Coordinate timeplot opens

Figure 16: Effect of clicking on a pixel coordinate and opening its timeplot showing filtered data and spikes.

The widget also gives the possibility of selecting a subarea of the whole array in all type of plots. This is shown in Figure 17 for the second type of peaks plot. Once the new coordinates are entered the button corresponding to the type of plot to show has to be clicked. A subarea can also be selected using the zoom.



Time 0

Th = σ * 6.0

X min 0

X max 10

Y min 0

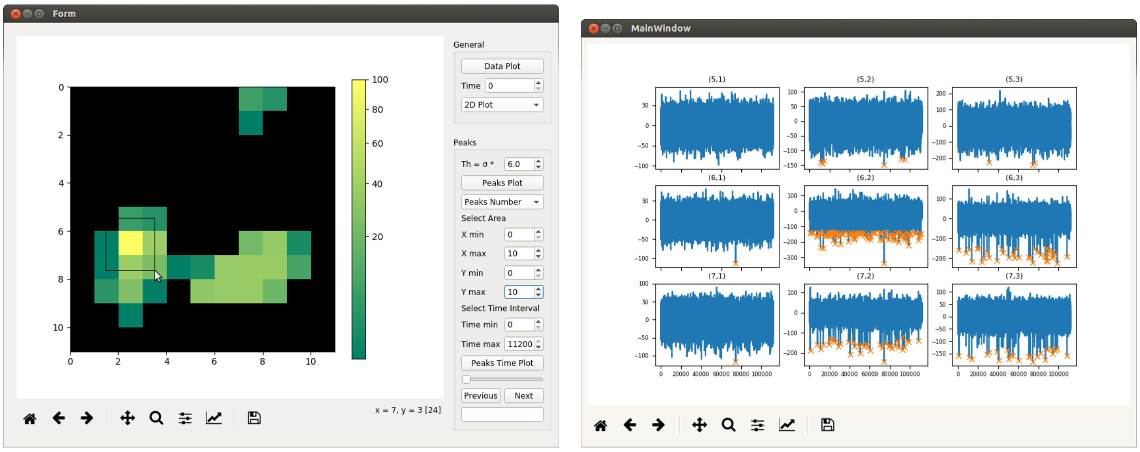
Y max 10

Time min 0

Time max 11200

Figure 17: Result of selecting a subarea of the whole array in second type of peaks plot.

Figure 18a shows a rectangle being drawn with the mouse over more than one pixel coordinate in the peaks plot of the same subarea as before in order to open a timeplot window for this whole region. Figure 18b shows the effect of this action, which is a new window appearing showing the timeplots of the pixels contained in the selected rectangle. As in the previous example, this window already plots the filtered data and showing the peaks. It is also easy to establish a relation between the two types of plots, as for this example the pixel in the middle contains the highest number of spikes and the one on the top left does not contain any.



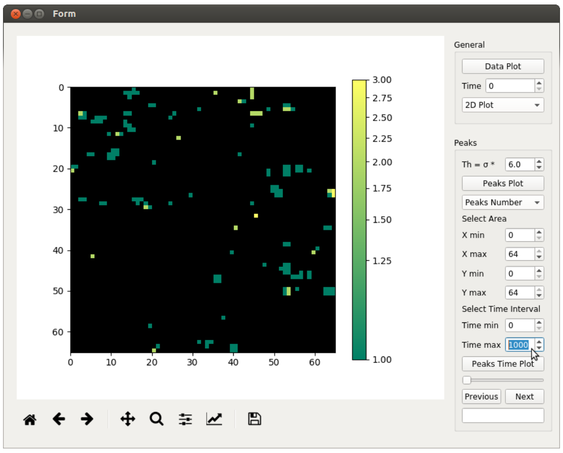
Time 0
Th = σ * 6.0
X min 0
X max 10
Y min 0
Y max 10
Time min 0
Time max 11200
x = 7, y = 3 [24]

(a) Selecting more than one pixel coordinate

(b) Selected timeplots open

Figure 18: Effect of drawing a rectangle with the mouse over more than one pixel coordinate in second type of peaks plot.

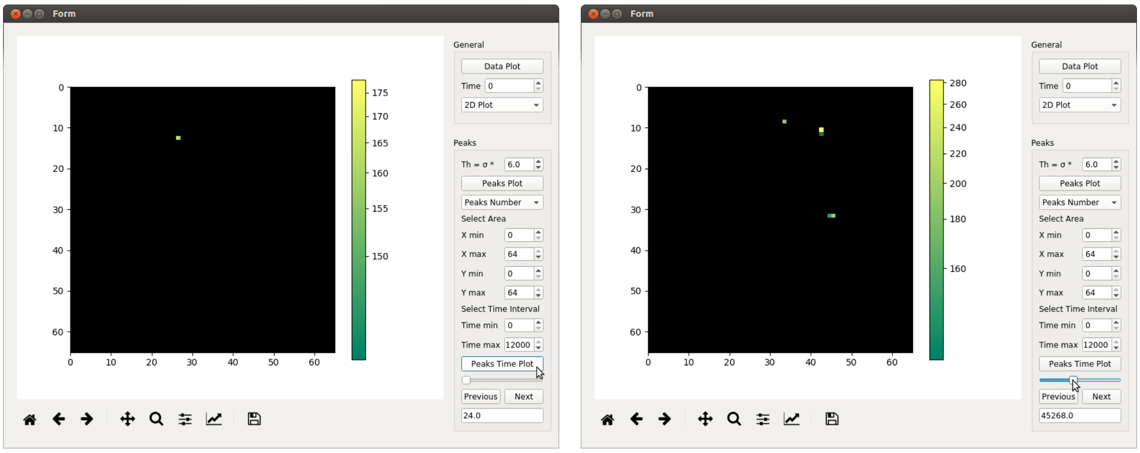
Another possibility that the grid plot widget offers is the selection of a specific time range. Figure 19 shows the selection of a maximum time sample of 1000 instead of 112000 for the second type of peaks plot. The effect of this can be seen in the reduced number of peaks detected.



Time 0
Th = σ * 6.0
X min 0
X max 64
Y min 0
Y max 64
Time min 0
Time max 1000

Figure 19: Result of selecting a different time range in second type of peaks plot.

Finally, a Peaks Time Plot button is incorporated for a third type of plot to be shown. This makes use of the same peaks that have been computed for the previous peaks plot but presents them in a different way. In this type of plot the different time samples where there are peaks somewhere within the array are presented starting from the lowest as shown in Figure 20a. The time sample is shown at the bottom of the widget and the plot shows in which coordinates there is a peak and of what height. Coordinates without pixels are shown in black. A slider is incorporated for navigating between these time samples as shown in Figure 20b, as well as the Previous and Next buttons.



(a) panel: Time 0; Th = σ * 6.0; X min 0; X max 64; Y min 0; Y max 64; Time min 0; Time max 12000; 24.0

(b) panel: Time 0; Th = σ * 6.0; X min 0; X max 64; Y min 0; Y max 64; Time min 0; Time max 12000; 45268.0

(a) First time sample with spikes

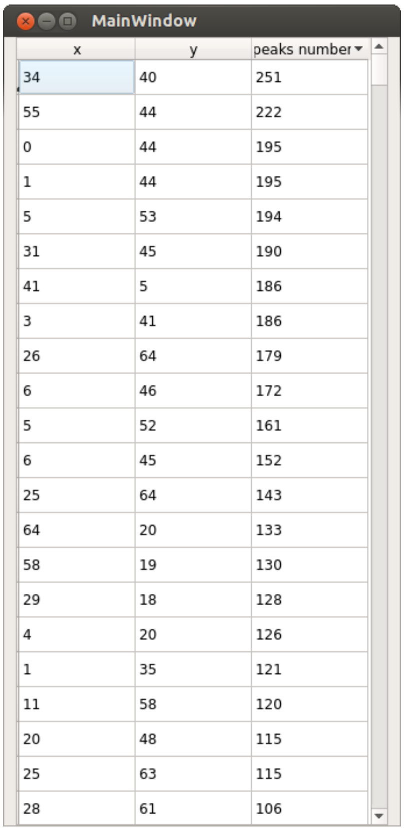
(b) Time sample selected with slider

Figure 20: Aspect of peaks time plot at different time samples with spikes.

4.3. Table view analysis results

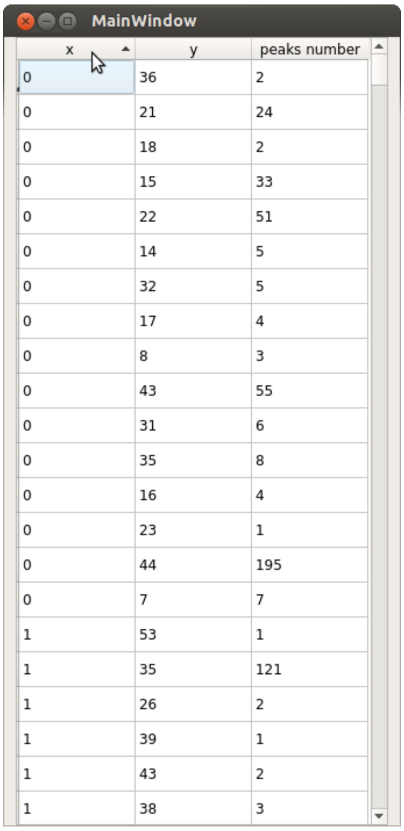
When the Peaks Plot button in the grid plot widget is clicked and the spikes are computed for a specific threshold value, a new window appears which displays data in the form of a table view. These tables aim to present information about the peaks in a more structured way and to make it easier to spot the pixels with most neural activity which usually indicates where the nerve cells are. It also aims to make it easier to visualize in which pixels there is activity taking place at the same or similar times.

The first table that appears is shown below. This includes the coordinates of all the pixels with peaks and the number of spikes found in each of them. By default it is ordered by the descending number of peaks as in Figure 21a, but it can be ordered by the ascending or descending coordinates as shown in Figure 21b. It is also possible to select an individual row and to navigate between rows by using the keyboard arrows.



x	y	peaks number
34	40	251
55	44	222
0	44	195
1	44	195
5	53	194
31	45	190
41	5	186
3	41	186
26	64	179
6	46	172
5	52	161
6	45	152
25	64	143
64	20	133
58	19	130
29	18	128
4	20	126
1	35	121
11	58	120
20	48	115
25	63	115
28	61	106

(a) Table ordered by descending peaks number

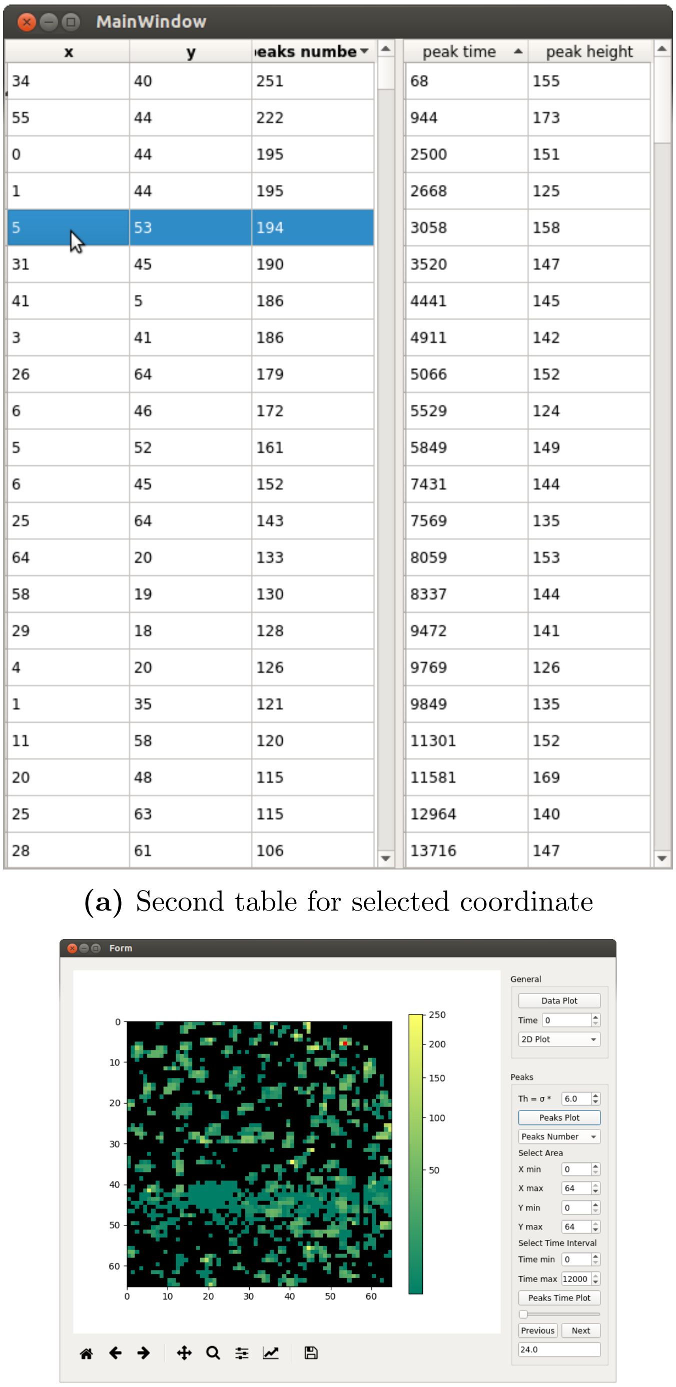


x	y	peaks number
0	36	2
0	21	24
0	18	2
0	15	33
0	22	51
0	14	5
0	32	5
0	17	4
0	8	3
0	43	55
0	31	6
0	35	8
0	16	4
0	23	1
0	44	195
0	7	7
1	53	1
1	35	121
1	26	2
1	39	1
1	43	2
1	38	3

(b) Table ordered by ascending x coordinate

Figure 21: Aspect of table view showing pixel coordinates and number of peaks ordered in two different ways.

Figure 22a shows one of the coordinates with peaks in the previous table being selected either by clicking or by using the keyboard arrows. The result of this is a second table appearing on the right. This table contains all the time samples where there are peaks for the selected pixel coordinate as well as their height. This table can also be ordered by both peak time or height. Additionally, the selected coordinate is highlighted on the peaks plot as shown in Figure 22b.



x	y	eaks numbe
34	40	251
55	44	222
0	44	195
1	44	195
5	53	194
31	45	190
41	5	186
3	41	186
26	64	179
6	46	172
5	52	161
6	45	152
25	64	143
64	20	133
58	19	130
29	18	128
4	20	126
1	35	121
11	58	120
20	48	115
25	63	115
28	61	106

peak time	peak height
68	155
944	173
2500	151
2668	125
3058	158
3520	147
4441	145
4911	142
5066	152
5529	124
5849	149
7431	144
7569	135
8059	153
8337	144
9472	141
9769	126
9849	135
11301	152
11581	169
12964	140
13716	147

(a) Second table for selected coordinate

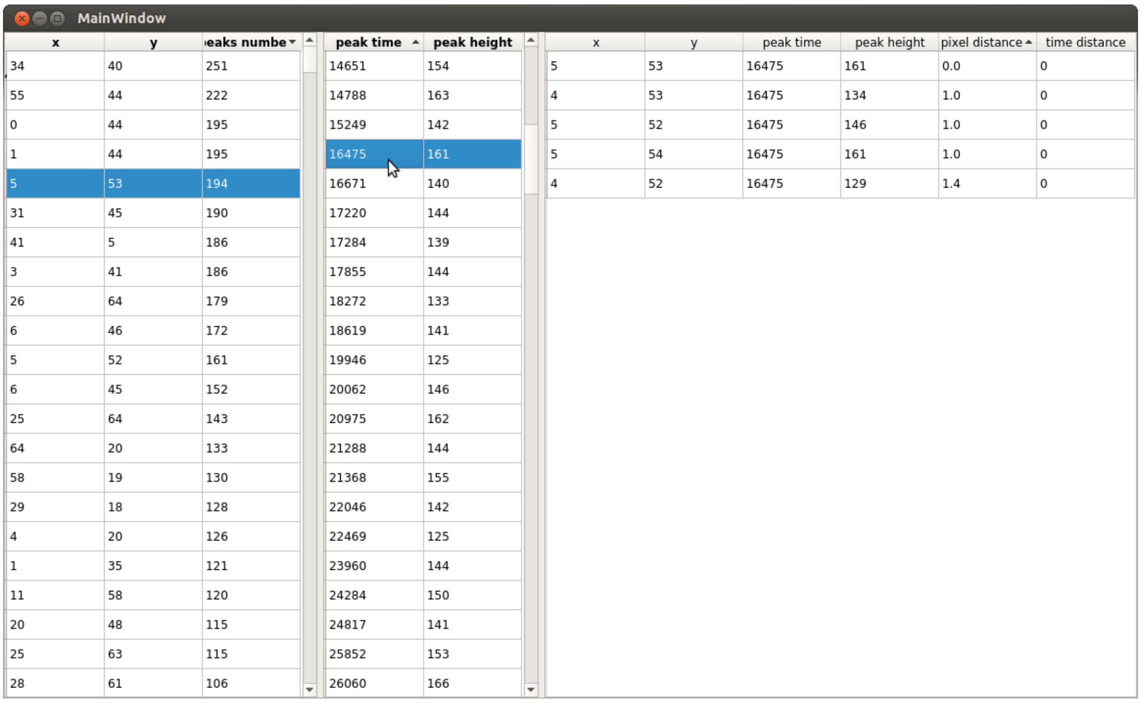
General
Data Plot
Time 0
2D Plot

Peaks
Th = σ * 6.0
Peaks Plot
Peaks Number
Select Area
X min 0
X max 64
Y min 0
Y max 64
Select Time Interval
Time min 0
Time max 12000
Peaks Time Plot
Previous Next
24.0

(b) Highlighted coordinate on peaks plot

Figure 22: Second table showing all spikes for the selected pixel coordinate and the effect on the peaks plot.

Similarly, Figure 23a shows one of the peak times being selected and a third table appearing. This third table contains the coordinates of the pixels that contain peaks at this or a similar time sample, as well as their height and distance to the pixel selected in the first place. This table can also be ordered by any of its columns. The coordinates are also highlighted in a different colour in the peaks plot as shown in Figure 23b.

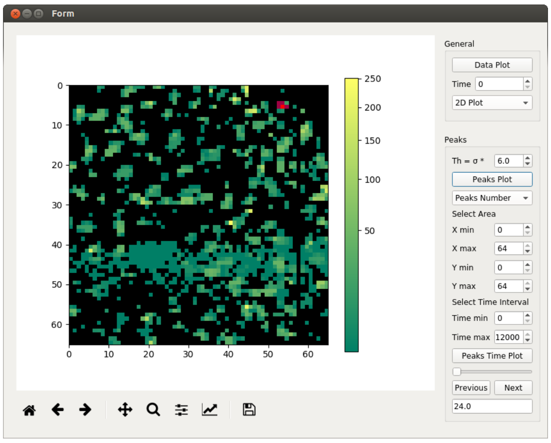


x	y	eaks numbe
34	40	251
55	44	222
0	44	195
1	44	195
5	53	194
31	45	190
41	5	186
3	41	186
26	64	179
6	46	172
5	52	161
6	45	152
25	64	143
64	20	133
58	19	130
29	18	128
4	20	126
1	35	121
11	58	120
20	48	115
25	63	115
28	61	106

peak time	peak height
14651	154
14788	163
15249	142
16475	161
16671	140
17220	144
17284	139
17855	144
18272	133
18619	141
19946	125
20062	146
20975	162
21288	144
21368	155
22046	142
22469	125
23960	144
24284	150
24817	141
25852	153
26060	166

x	y	peak time	peak height	pixel distance	time distance
5	53	16475	161	0.0	0
4	53	16475	134	1.0	0
5	52	16475	146	1.0	0
5	54	16475	161	1.0	0
4	52	16475	129	1.4	0

(a) Third table for selected time sample



Time 0

2D Plot

Th = σ * 6.0

Peaks Number

Select Area

X min 0

X max 64

Y min 0

Y max 64

Select Time Interval

Time min 0

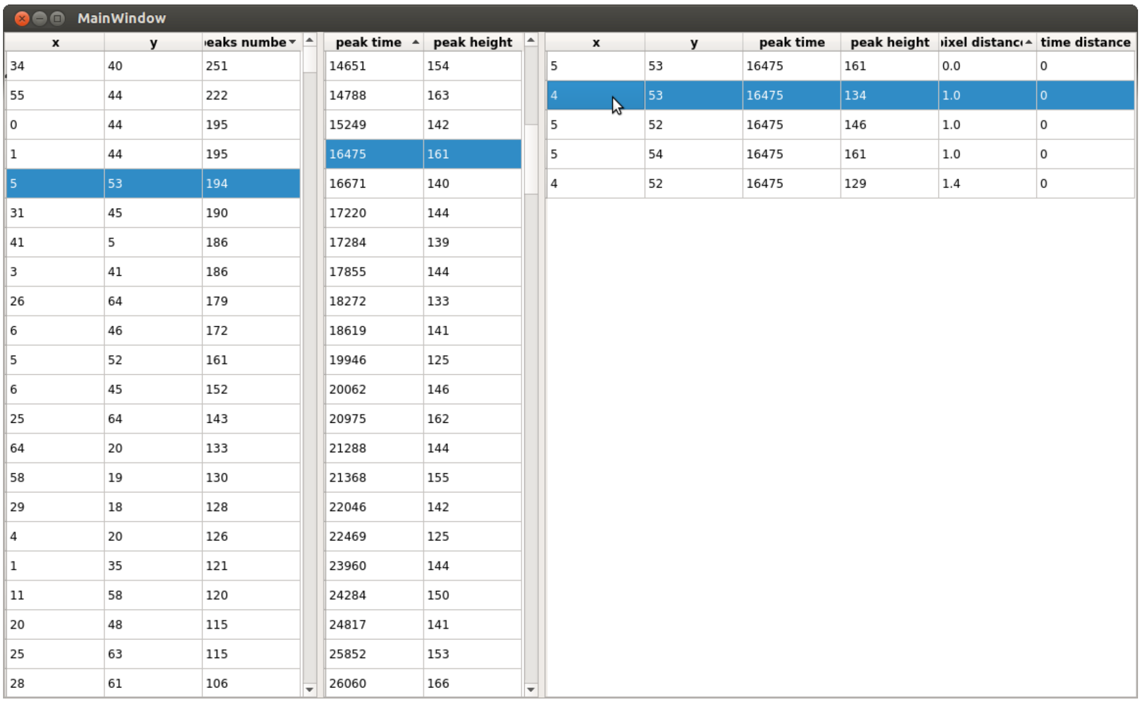
Time max 12000

24.0

(b) Highlighted region on peaks plot

Figure 23: Third table showing all coordinates with spikes for the selected time sample or a near one and the effect on the peaks plot.

Finally, Figure 24a shows one of the pixel coordinates in the third table being selected. This coordinate is highlighted in the peaks plot as shown in Figure 24b.



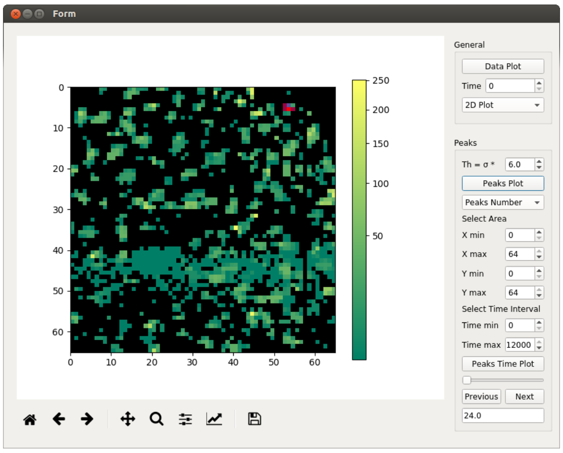
MainWindow

x	y	eaks numbe
34	40	251
55	44	222
0	44	195
1	44	195
5	53	194
31	45	190
41	5	186
3	41	186
26	64	179
6	46	172
5	52	161
6	45	152
25	64	143
64	20	133
58	19	130
29	18	128
4	20	126
1	35	121
11	58	120
20	48	115
25	63	115
28	61	106

peak time	peak height
14651	154
14788	163
15249	142
16475	161
16671	140
17220	144
17284	139
17855	144
18272	133
18619	141
19946	125
20062	146
20975	162
21288	144
21368	155
22046	142
22469	125
23960	144
24284	150
24817	141
25852	153
26060	166

x	y	peak time	peak height	ixel distanc	time distance
5	53	16475	161	0.0	0
4	53	16475	134	1.0	0
5	52	16475	146	1.0	0
5	54	16475	161	1.0	0
4	52	16475	129	1.4	0

(a) Selection of pixel coordinate on the third table



Form

General

Data Plot

Time 0

2D Plot

Peaks

Th = σ * 6.0

Peaks Plot

Peaks Number

Select Area

X min 0

X max 64

Y min 0

Y max 64

Select Time Interval

Time min 0

Time max 12000

Peaks Time Plot

Previous Next

24.0

(b) Highlighted coordinate on the peaks plot.

Figure 24: Selection of one of the pixel coordinates on the third table and the effect on the peaks plot.

5. Summary

Overall, it can be stated that the principal objectives set prior to the development of the thesis have been achieved and expectations have been fulfilled. A GUI application has been created which is able to analyze neural data recorded from a CMOS MEA system and which successfully detects neural activity within it, allowing its visualization in a number of different ways. As previously said, this information is useful for reducing the amount of information that needs to be stored and for detecting where the nerve cells are within the tissue subjected to analysis. The GUI application offers a number of useful features such as:

- Being able to perform a timeplot for any one of the pixel coordinates in the array and to display either the raw or the filtered data, as well as the peaks found by the spike detection method.
- Give the user the option of trying out different threshold values and in this way evaluate the spike detection method by himself, which can be useful when a dataset with different results from the expected ones needs to be analyzed.
- Being able to perform either a 2D or a 3D grid plot of the whole array data for any given time sample.
- Being able to show in which pixels there has been spikes detected, as well as the number of them, and in which pixels there has been no spikes found. Being able to visualize the different spikes that have been found showing in which pixels they take place and starting from the lowest time sample.
- Give the user the possibility of selecting only a subarea of the array or a specific time sample range to make faster trials and to focus the attention where needed.
- Give the user the possibility to compare the timeplots of more than one pixel at the same type as well as those of a subarea of pixels selected in the grid plod.
- Being able to show, in the form of tables, which pixels have most peaks and localize them in the grid plot. Show correspondence between peaks from different pixels that happen at similar time samples and that probably belong to the same electrical signal.

Moreover, it can be said that the GUI application created is intuitive and easy to learn and use, and provides an interactive experience with the user by means of the WIMP paradigm based on windows, icons, menus and pointer. It also allows different objects to be displayed at the same time.

In what refers to the aspects learned, the development of the thesis has been an opportunity to learn about different software environments and to gain an in-depth knowledge about programming in Python. It has also given the chance to learn the

basics about CMOS MEA systems and which parts compose them, as well as what APs are and how nerve cells communicate between them.
However, a more complex spike detection algorithm could have been used, such as the *LocEn* method or one based on ICA, though the method chosen presents a low error rate and a high efficiency compared to other algorithms that take more time and effort to be developed and implemented. Another aspect to improve would be the structure of the application, as more emphasis could have been put on the MVP pattern and the files could have been organized in a better way. In a similar way, the filtered data or the peak data calculated for a specific dataset could have been stored somehow as cache of the application for it to not have to be computed every time.

A. Appendix: Code

Python file 1: analysis.py

```
import h5py
import numpy as np
from scipy.signal import find_peaks, bessel, filtfilt
import time
import pandas as pd
import numpy.ma as ma


def timerfunc(func):

    def function_timer(*args, **kwargs):
        start = time.time()
        value = func(*args, **kwargs)
        end = time.time()
        runtime = end - start
        msg = ”The runtime for {func} took {time} seconds to
    complete”
        print(msg.format(func=func.__name__,
                                 time=runtime))
        return value
    return function_timer


class Analysis:

    def __init__(self, data_path):
        self.data_path = data_path

        with h5py.File(data_path) as hdf:
            self.data = hdf['Acquisition/Sensor Data/
   SensorData 1 1'][:,:,:]

        self.timemin = 0
        self.timemax = self.data.shape[0]
        self.timemaxaux = self.timemax-self.timemin
        self.xmin = 0
        self.xmax = self.data.shape[1]
        self.xmaxaux = self.xmax-self.xmin
        self.ymin = 0
        self.ymax = self.data.shape[2]
        self.ymaxaux = self.ymax-self.ymin
        self.allfiltdata = None

        self.th = 4.5
        self.peaks = np.array([])
        self.peaksbinary = np.array([])
        self.peaksnumb = np.array([])

    def filter_data(self, x, y):
        b, a = bessel(2, 0.03, 'high')
        self.filtdata = filtfilt(b, a, self.data[:self.
   timemaxaux,(x-self.xmin),(y-self.ymin)])
        return self.filtdata

    @timerfunc
    def filt_alldata(self):
        self.allfiltdata = np.zeros((self.timemaxaux,self.
   xmaxaux,self.ymaxaux))
        b, a = bessel(2, 0.03, 'high')
        for i in range(0, self.xmaxaux):
            for j in range(0, self.ymaxaux):
                self.allfiltdata[:self.timemaxaux,i,j] =
   filtfilt(b, a, self.data[:self.timemaxaux,i,j])
        if self.ymax >= 57:
            self.allfiltdata[:,:,57 - self.ymin] = ma.masked
        print("Filt data ready")
        return self.allfiltdata

    @timerfunc
    def compute_allpeaks(self, data, xmin, xmax, ymin, ymax,
    timemin, timemax):
        self.acc_x = []
        self.acc_y = []
        self.acc_times = []
        self.acc_heights = []
        self.acc_numbs = []

        self.timemin = timemin
        self.timemax = timemax

        for i in range(xmin, xmax):
            for j in range(ymin, ymax):
                self.compute_peaks(data[:,i,j])
                if self.sortedpeaks.size:
                 self.acc_ifpeaks(i,j)

     self.array = np.array([self.acc_x,self.acc_y,self.
acc_times])

     if self.array.size:
         tuples = list(zip(*self.array))
         index=pd.MultiIndex.from_tuples(tuples)
         self.peaks = pd.Series(self.acc_heights, index=
index)
         self.sortedpeaktimes = np.unique(np.sort(self.
acc_times, axis=0))
         self.peaksheights = self.peaks.droplevel(level
=2)

         index2=(index.droplevel(level=2)).
drop_duplicates()
         for i,j in index2:
             if isinstance(self.peaksheights.loc[i,j], np
.float64) is True:
                 self.acc_numbs = np.concatenate((self.
acc_numbs, np.array([1])), axis=0)
             else:
                 self.acc_numbs = np.concatenate((self.
acc_numbs, np.array([len(self.peaksheights.loc[i,j].
values)])), axis=0)
         self.peaknumbs = (pd.Series(self.acc_numbs,
index=index2)).sort_values(ascending=False)

         print(self.peaks)
         print(self.peaknumbs)
         print(”Peak data ready”)

     else:
         print(”No peak data”)

     return self.peaks

 def compute_peaks(self, data):
     dataaux = data[self.timemin:self.timemax]
     self.thaux = self.th*(data.std())
     self.peaktimes, properties = find_peaks(abs(dataaux)
, height=self.thaux, distance=8)
     if self.timemin != 0:
         self.peaktimes = np.asarray([self.peaktimes[i]+
self.timemin for i in np.arange(len(self.peaktimes))])
     self.heights = properties[”peak_heights”]
     self.sortedpeaks = np.asarray([[self.peaktimes[i],
self.heights[i]] for i in np.arange(len(self.peaktimes))
])
     return self.peaktimes

 def acc_ifpeaks(self, x, y):
     for i in range(len(self.sortedpeaks)):
         self.acc_x = np.concatenate((self.acc_x, np.
array([x])), axis=0)
         self.acc_y = np.concatenate((self.acc_y, np.
array([y])), axis=0)
         self.acc_times = np.concatenate((self.acc_times,
 np.array([self.sortedpeaks[i,0]])), axis=0)
         self.acc_heights = np.concatenate((self.
acc_heights, np.array([self.sortedpeaks[i,1]])), axis=0)

 def update_peaksbinary(self):
     self.peaksbinary = np.zeros(((self.xmax-self.xmin),(
self.ymax-self.ymin)))
     for i,j in self.peaksheights.index:
         self.peaksbinary[int(i),int(j)] = 1

 def update_peaksnumb(self):
     self.peaksnumb = np.zeros(((self.xmax-self.xmin),(
self.ymax-self.ymin)))
     for i,j in self.peaksheights.index:
         if isinstance(self.peaksheights.loc[i,j], np.
float64) is True:
             self.peaksnumb[int(i),int(j)] = 1
         else:
             self.peaksnumb[int(i),int(j)] = len(self.
peaksheights.loc[i,j].values)

 def update_sortedpeaktimes(self, xmin, xmax, ymin, ymax)
:
     self.peaks = self.peaks.loc[xmin:xmax,ymin:ymax]
     print(self.peaks)
     self.sortedpeaktimes = np.unique(np.sort((self.peaks
.droplevel(level=[0,1])).index.values))
     self.peaksheights = self.peaks.droplevel(level=2)
     print(self.sortedpeaktimes)
```

```
    def update_peaktimeplot(self, time):
        self.peaktime = time
        self.peaksaux = self.peaks[:,:,self.sortedpeaktimes[self.peaktime]]
        self.peaksarray = np.zeros(((self.xmax-self.xmin),(self.ymax-self.ymin)))
        for i,j in self.peaksaux.index:
            self.peaksarray[int(i),int(j)] = self.peaksaux[i,j]
        return self.peaksarray


if __name__ == "__main__":

    ana = Analysis("../../../../data/2017.12.07-11.55.30-Spont.cmcr")
    peaks = ana.compute_allpeaks(ana.filt_alldata(), ana.xmin, ana.xmax, ana.ymin, ana.ymax, ana.timemin, ana.timemax)
```

Python file 2: app.py

```
import sys
from PyQt5.QtCore import pyqtSlot
from PyQt5.QtWidgets import QFileDialog, QApplication, QMainWindow
from PyQt5.uic import loadUi

from tabs.timeplot_widget import TimeplotWidget

class PyQtApp(QMainWindow):
    def __init__(self):
        super(PyQtApp, self).__init__()
        loadUi('../_res/ui/mainwindow.ui', self)
        self.prepare_tab_widget()

    def prepare_tab_widget(self):
        tb= self.tabWidget
        tb.setTabsClosable(True)
        for tab in range(tb.count()-1, -1, -1):
            self.tabWidget.removeTab(tab)
        tb.tabCloseRequested.connect(self.close_tab)
    def close_tab(self, index):
        self.tabWidget.removeTab(index)

    @pyqtSlot()
    def on_actionLoad_triggered(self):
        options = QFileDialog.Options()
        options |= QFileDialog.DontUseNativeDialog
        filename, _ = QFileDialog.getOpenFileName(self,"
   Select Data File",'../../../../data',"All Files (*);;
   Acquisition Data Files (*.cmcr)",options=options)

        if filename:
            self.tabWidget.addTab(TimeplotWidget(parent=self
   .tabWidget,filename=filename),filename)

if __name__ == '__main__':
    app = QApplication(sys.argv)
    win = PyQtApp()
    win.show()
    sys.exit(app.exec_())
```

Python file 3: timeplot_widget.py

```
import sys
sys.path.insert(0, '../')
from PyQt5.QtCore import pyqtSlot
from PyQt5.QtWidgets import QWidget
from PyQt5.uic import loadUi

from analysis.analysis import Analysis
from viewer.plotting.plot_window import PlotWindow
from viewer.tabs.peaks_widget import GridplotWidget


class TimeplotWidget(QWidget):
    def __init__(self, filename=None, path=None , parent=
   None, *args, **kwargs):
        super().__init__(parent)
        loadUi('../_res/ui/neuro_widget.ui',self)
        self.filename = filename
        self.analysis = Analysis(self.filename)
        layout = self.horizontalLayout.layout()
```

```
      if parent is None:
          self.setGeometry(10, 10, 790, 510)
      self.xval = self.analysis.xmin
      self.yval = self.analysis.ymin
      self.data = self.analysis.data[:, self.xval−self.
analysis.xmin, self.yval−self.analysis.ymin]
      self.filterchecked = False
      self.peakschecked = False
      self.plot_widget = PlotWindow(data=self.data,
filename=self.filename)
      self.xCoord.setValue(self.xval)
      self.yCoord.setValue(self.yval)
      layout.insertWidget(0, self.plot_widget)
      self.plot_widget.setParent(self)
      self.plot_widget.show()
      self.actionApplyHPFilt.toggled.connect(self.
on_actionApplyHPFilt_toggled)
      self.actionComputePeaks.toggled.connect(self.
on_actionComputePeaks_toggled)
      self.refresh_values(self.data)
      self.thVal.setValue(self.analysis.th)
      self.open_windows = []
      self.xInfo.setText(str(self.analysis.xmin)+” to ”+
str(self.analysis.xmax))
      self.yInfo.setText(str(self.analysis.ymin)+” to ”+
str(self.analysis.ymax))
      self.timeInfo.setText(str(self.analysis.timemin)+”
to ”+str(self.analysis.timemax))

  def refresh_values(self, data):
      mean = str(round(data.mean(), 2))
      self.meanVal.setText(mean)
      ptp = str(round(data.ptp(), 2))
      self.ptpVal.setText(ptp)
      std = str(round(data.std(), 2))
      self.stdVal.setText(std)
      max = str(round(data.max(), 2))
      self.maxVal.setText(max)
      min = str(round(data.min(), 2))
      self.minVal.setText(min)

  @pyqtSlot(bool)
  def on_actionApplyHPFilt_toggled(self, checked):
      if checked:
          self.filterchecked = True
      elif not checked:
          self.filterchecked = False
      self.update_plot()

  def filt_data(self):
      self.analysis.filt_alldata()

  @pyqtSlot(int)
  def on_xCoord_valueChanged(self, value):
      self.xval = value
      self.update_plot()

  @pyqtSlot(int)
  def on_yCoord_valueChanged(self, value):
      self.yval = value
      self.update_plot()

  @pyqtSlot(float)
  def on_thVal_valueChanged(self, value):
      self.analysis.th = value
      self.update_plot()

  @pyqtSlot(bool)
  def on_actionComputePeaks_toggled(self, checked):
      if checked:
          self.peakschecked = True
          self.update_plot()
      elif not checked:
          self.peakschecked = False
          self.update_plot()

  def update_plot(self):
      if self.filterchecked is True:
          if self.analysis.allfiltdata is None:
              self.data = self.analysis.filter_data(self.
xval,self.yval)
          else:
              self.data = self.analysis.allfiltdata[:,self
.xval-self.analysis.xmin,self.yval-self.analysis.ymin]
      elif self.filterchecked is False:
          self.data = self.analysis.data[:,self.xval-self.
analysis.xmin,self.yval-self.analysis.ymin]
      self.plot_widget.plot(self.data, self.xval, self.
yval, self.analysis.timemin, self.analysis.timemax)
      self.refresh_values(self.data)
      if self.peakschecked is True:
          self.plot_widget.plot_peaks(self.analysis.
compute_peaks(self.data))

  @pyqtSlot()
  def on_actionOpenNewWindow_clicked(self):
      win = Timeplotwidget(filename=self.filename, new=
True)
      win.show()
      self.open_windows.append(win)

  @pyqtSlot()
  def on_actionSurfacePlot_clicked(self):
      if self.analysis.allfiltdata is None:
          self.filt_data()
      win = GridplotWidget(filename=self.filename, data=
self.analysis.allfiltdata)
      win.show()
      self.open_windows.append(win)
```

Python file 4: gridplot_widget.py

```
import sys
sys.path.insert(0, '../')
import numpy as np
import math
from PyQt5.QtCore import pyqtSlot
from PyQt5.QtWidgets import QWidget
from PyQt5.uic import loadUi

from analysis.analysis import Analysis
from viewer.plotting.plot_window import PlotWindow
from viewer.tabs.table_window import TableWindow


class GridplotWidget(QWidget):
    def __init__(self, filename=None, path=None , parent=
None, data = None, *args, **kwargs):
        super().__init__(parent)
        loadUi('../_res/ui/peaks_widget.ui',self)
        self.filename = filename
        self.data = data
        self.analysis = Analysis(self.filename)
        layout = self.horizontalLayout.layout()
        self.setGeometry(10, 10, 850, 550)
        self.timemin = self.analysis.timemin
        self.timemax = self.analysis.timemax
        self.time = self.timemin
        self.timeaux = self.time-self.timemin
        self.xmin = self.analysis.xmin
        self.xCoordMin.setValue(self.xmin)
        self.xmax = self.analysis.xmax
        self.xCoordMax.setValue(self.xmax)
        self.ymin = self.analysis.ymin
        self.yCoordMin.setValue(self.ymin)
        self.ymax = self.analysis.ymax
        self.yCoordMax.setValue(self.ymax)
        self.plot_widget = PlotWindow(filename=self.filename
, data=self.data)
        self.plot_widget.plotdata_2d(self.data[self.timeaux
,:,:],self.xmin,self.xmax,self.ymin,self.ymax)
        layout.insertWidget(0,self.plot_widget)
        self.plot_widget.setParent(self)
        self.plot_widget.show()
        self.thVal.setValue(self.analysis.th)
        self.peaktime = 0
        self.th_changed = False
        self.peaksplot_type = 0
        self.peaksBox.currentIndexChanged.connect(self.
on_peaksBox_indexChanged)
        self.dataBox.currentIndexChanged.connect(self.
on_dataBox_indexChanged)
        self.timeBox.currentIndexChanged.connect(self.
on_timeBox_indexChanged)
        self.horizontalSlider.valueChanged.connect(self.
on_horizontalSlider_valueChanged)
        self.plot_widget.canvas.mpl_connect('
button_press_event', self.on_click)
        self.plot_widget.canvas.mpl_connect('
button_release_event', self.on_release)
        self.open_windows = []
        self.ax_changed = False
        self.time_changed = False
        self.dataplot_type = 0
        self.timeplot_type = 0
        self.timeVal.setValue(self.time)
        self.timeVal.setRange(self.timemin, self.timemax)
        self.timeMin.setValue(self.timemin)
        self.timeMin.setRange(self.timemin, self.timemax)
        self.timeMax.setValue(self.timemax)
        self.timeMax.setRange(self.timemin, self.timemax)
        self.marker = False

    def update_axes(self):
        self.plot_widget.ax.callbacks.connect('xlim_changed'
, self.on_ax_changed)
        self.plot_widget.ax.callbacks.connect('ylim_changed'
, self.on_ax_changed)

    @pyqtSlot()
    def on_actionDataPlot_clicked(self):
        self.update_dataplot()

    @pyqtSlot(int)
    def on_dataBox_indexChanged(self, value):
        if value == 0:
            self.dataplot_type = 0
        elif value == 1:
            self.dataplot_type = 1
        self.update_dataplot()

    def update_dataplot(self):
        self.timeaux = self.time-self.timemin
        if self.dataplot_type == 0:
            self.plot_widget.plotdata_2d(self.data[self.
timeaux,:,:],self.xmin,self.xmax,self.ymin,self.ymax)
        elif self.dataplot_type == 1:
            self.plot_widget.plotdata_3d(self.timeaux,self.
data,self.xmin,self.xmax,self.ymin,self.ymax)
        self.update_axes()

    @pyqtSlot(int)
    def on_timeVal_valueChanged(self, value):
        self.time = value
        self.update_dataplot()

    @pyqtSlot(float)
    def on_thVal_valueChanged(self, value):
        self.analysis.th = value
        self.th_changed = True

    @pyqtSlot()
    def on_actionPeaksPlot_clicked(self):
        if not self.analysis.peaks.size or self.th_changed
is True or self.ax_changed is True or self.time_changed
is True:
            self.analysis.compute_allpeaks(self.data, self.
xmin, self.xmax, self.ymin, self.ymax, self.timemin, self.
timemax)
            self.open_tablewindow()
            self.analysis.update_peaksbinary()
            self.analysis.update_peaksnumb()
            self.th_changed = False
            self.ax_changed = False
        if not self.analysis.peaksbinary.size:
            self.analysis.update_peaksbinary()
            self.analysis.update_peaksnumb()
        self.update_peaksplot()

    def update_peaksplot(self):
        if self.peaksplot_type == 0:
            self.discretedata = self.analysis.peaksbinary
        elif self.peaksplot_type == 1:
            self.discretedata = self.analysis.peaksnumb
        self.markerdata = np.array(self.discretedata)
        if self.marker is True:
            if self.table_window.tablemodel3 is not None:
                for i in range(len(self.table_window.
tablemodel3.data)):
                    self.markerdata[int(list(self.
table_window.tablemodel3.data[i].values())[0]),int(list(
self.table_window.tablemodel3.data[i].values())[1])] = -3
                if self.table_window.pixrow is not None:
                    self.markerdata[self.table_window.pix_x,
self.table_window.pix_y] = -2
            self.markerdata[int(self.table_window.x),int(
self.table_window.y)] = -1
        self.plot_widget.plotpeaks_2d(self.markerdata, self.
xmin, self.xmax, self.ymin, self.ymax, self.marker, self.
peaksplot_type)
        self.update_axes()

    @pyqtSlot(int)
```

```
  def on_peaksBox_indexChanged(self, value):
      if value == 0:
          self.peaksplot_type = 0
      elif value == 1:
          self.peaksplot_type = 1
      if not self.analysis.peaks.size:
          self.analysis.compute_allpeaks(self.data, self.
xmin, self.xmax, self.ymin, self.ymax, self.timemin, self.
timemax)
          self.analysis.update_peaksbinary()
          self.analysis.update_peaksnumb()
      if not self.analysis.peaksbinary.size:
          self.analysis.update_peaksbinary()
          self.analysis.update_peaksnumb()
      self.update_peaksplot()

  @pyqtSlot()
  def on_actionPeaksTimePlot_clicked(self):
      if not self.analysis.peaks.size or self.th_changed
is True or self.ax_changed is True or self.time_changed
is True:
          self.analysis.compute_allpeaks(self.data, self.
xmin, self.xmax, self.ymin, self.ymax, self.timemin, self.
timemax)
          self.peaktime = 0
          self.th_changed = False
          self.ax_changed = False
      self.analysis.update_sortedpeaktimes(self.xmin, self.
xmax, self.ymin, self.ymax)
      self.horizontalSlider.setMaximum(len(self.analysis.
sortedpeaktimes)-1)
      self.update_timeplot()

  @pyqtSlot(int)
  def on_horizontalSlider_valueChanged(self, value):
      if not self.analysis.peaks.size:
          self.analysis.compute_allpeaks(self.data, self.
xmin, self.xmax, self.ymin, self.ymax, self.timemin, self.
timemax)
          self.horizontalSlider.setMaximum(len(self.
analysis.sortedpeaktimes)-1)
      self.peaktime = value
      self.update_timeplot()
```

```
 def update_timeplot(self):
    if self.timeplot_type == 0:
       self.plot_widget.plotpeaks_2d(self.analysis.
update_peaktimeplot(self.peaktime),self.xmin,self.xmax,
self.ymin,self.ymax,self.marker,1)
    elif self.timeplot_type == 1:
       self.plot_widget.plotdata_3d(int(self.analysis.
sortedpeaktimes[self.peaktime]),self.data,self.xmin,self.
xmax,self.ymin,self.ymax)
    self.peakTime.setText(str(self.analysis.
sortedpeaktimes[self.peaktime]))
    self.update_axes()

 @pyqtSlot()
 def on_actionPreviousPeakTime_clicked(self):
    if not self.analysis.peaks.size:
       self.analysis.compute_allpeaks(self.data,self.
xmin,self.xmax,self.ymin,self.ymax,self.timemin,self.
timemax)
       self.peaktime = 0
       self.horizontalSlider.setMaximum(len(self.
analysis.sortedpeaktimes)-1)
    elif self.peaktime > 0:
       self.peaktime = self.peaktime - 1
       self.horizontalSlider.setValue(self.peaktime)
    self.update_timeplot()

 @pyqtSlot()
 def on_actionNextPeakTime_clicked(self):
    if not self.analysis.peaks.size:
       self.analysis.compute_allpeaks(self.data,self.
xmin,self.xmax,self.ymin,self.ymax,self.timemin,self.
timemax)
       self.peaktime = 0
       self.horizontalSlider.setMaximum(len(self.
analysis.sortedpeaktimes)-1)
    elif self.peaktime < (len(self.analysis.
sortedpeaktimes)-1):
       self.peaktime = self.peaktime + 1
       self.horizontalSlider.setValue(self.peaktime)
    self.update_timeplot()

 @pyqtSlot(int)
 def on_xCoordMin_valueChanged(self, value):
        self.xmin = value
        self.ax_changed = True

    @pyqtSlot(int)
    def on_xCoordMax_valueChanged(self, value):
        self.xmax = value+1
        self.ax_changed = True

    @pyqtSlot(int)
    def on_yCoordMin_valueChanged(self, value):
        self.ymin = value
        self.ax_changed = True

    @pyqtSlot(int)
    def on_yCoordMax_valueChanged(self, value):
        self.ymax = value+1
        self.ax_changed = True

    @pyqtSlot(int)
    def on_timeMin_valueChanged(self, value):
        self.timemin = value
        self.time_changed = True

    @pyqtSlot(int)
    def on_timeMax_valueChanged(self, value):
        self.timemax = value
        self.time_changed = True

    def on_ax_changed(self, axes):
        self.ax_changed = True
        self.x_ax = self.plot_widget.ax.get_ylim()
        self.xmin = math.floor(self.x_ax[1])
        self.xCoordMin.setValue(self.xmin)
        self.xmax = math.ceil(self.x_ax[0])
        self.xCoordMax.setValue(self.xmax-1)
        self.y_ax = self.plot_widget.ax.get_xlim()
        self.ymin = math.floor(self.y_ax[0])
        self.yCoordMin.setValue(self.ymin)
        self.ymax = math.ceil(self.y_ax[1])
        self.yCoordMax.setValue(self.ymax-1)

    def on_click(self, event):
        if self.plot_widget.toolbar._active is None and self
.timeplot_type != 1 and self.dataplot_type != 1:
```

```
            self.xclick = int(event.ydata)
            self.yclick = int(event.xdata)

    def on_release(self, event):
        if self.plot_widget.toolbar._active is None and self
.timeplot_type != 1 and self.dataplot_type != 1:
            self.xrelease = int(event.ydata)
            self.yrelease = int(event.xdata)
            self.open_timeplot()

    def open_timeplot(self):
        win = PlotWindow(filename=self.filename)
        if self.xclick == self.xrelease and self.yclick ==
self.yrelease:
            win.setGeometry(10, 10, 690, 510)
            win.title = True
            win.plot(self.data[:, self.xclick-self.analysis.
xmin, self.yclick-self.analysis.ymin], self.xclick, self.
yclick, self.timemin, self.timemax)
            win.plot_peaks(self.analysis.compute_peaks(self.
data[:, self.xclick-self.analysis.xmin, self.yclick-self.
analysis.ymin]))
        else:
            win.setGeometry(10, 10, 300+(180*(self.yrelease-
self.yclick+1)), 200+(140*(self.xrelease-self.xclick+1)))
            win.plot_subplots(self.data, self.xclick, self.
xrelease+1, self.yclick, self.yrelease+1, self.analysis.th)
        win.show()
        self.open_windows.append(win)

    @pyqtSlot(int)
    def on_timeBox_indexChanged(self, value):
        if value == 0:
            self.timeplot_type = 0
        elif value == 1:
            self.timeplot_type = 1
        self.update_timeplot()

    def open_tablewindow(self):
        self.table_window = TableWindow(filename=self.
filename, data=self.analysis.peaknumbs, timedata=self.
analysis.peaks)
        self.table_window.show()
        self.open_windows.append(self.table_window)
        self.table_window.selmodel.selectionChanged.connect(
   self.on_tableSel)

    def on_tableSel(self, sel, desel):
        self.marker = True
        self.update_peaksplot()
        self.marker = False
        self.table_window.selmodel_2.selectionChanged.
   connect(self.on_table2Sel)

    def on_table2Sel(self, sel, desel):
        self.marker = True
        self.update_peaksplot()
        self.marker = False
        self.table_window.selmodel_3.selectionChanged.
   connect(self.on_table3Sel)

    def on_table3Sel(self, sel, desel):
        self.marker = True
        self.update_peaksplot()
        self.marker = False
```

Python file 5: plot_window.py

```
import sys
sys.path.insert(0, '../')
import matplotlib as mpl
from PyQt5.QtWidgets import QMainWindow, QWidget,
   QVBoxLayout, QSizePolicy
from PyQt5.uic import loadUi
from matplotlib import cm
from matplotlib.backends.backend_qt5agg import
   FigureCanvasQTAgg as FigureCanvas
mpl.rcParams['toolbar'] = 'toolmanager'
from matplotlib.backends.backend_qt5agg import
   NavigationToolbar2QT as NavigationToolbar
from matplotlib.figure import Figure
import matplotlib.colors as colors
from analysis.analysis import Analysis
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
```

```
class PlotWindow(QMainWindow):
    def __init__(self, filename=None, parent=None):
        super().__init__(parent)
        loadUi('../_res/ui/plot_mainwindow.ui', self)
        self.filename = filename
        self.analysis = Analysis(self.filename)
        self.figure = Figure()
        self.canvas = FigureCanvas(self.figure)
        layout = QVBoxLayout()
        self.canvas.setSizePolicy(QSizePolicy.Expanding,
   QSizePolicy.Expanding)
        self.canvas.updateGeometry()
        layout.insertWidget(0, self.canvas)
        widget = QWidget()
        widget.setLayout(layout)
        self.setCentralWidget(widget)
        self.toolbar = NavigationToolbar(self.canvas, self)
        layout.addWidget(self.toolbar)
        self.ax = None
        self.colorbar = None
        self.title = False

    def plot(self, data, x, y, timemin, timemax):
        self.data = data
        if self.ax is not None:
            self.ax.clear()
        else:
            self.ax = self.canvas.figure.subplots()
        if self.title is True:
            self.figure.suptitle("(" + str(x) + "," + str(y)
   + ")", fontsize=10)
        if timemin != self.analysis.timemin or timemax !=
   self.analysis.timemax:
            self.ax.set_xlim(left=timemin, right=timemax)
        self.ax.plot(self.data)
        self.ax.figure.canvas.draw()

    def plot_peaks(self, data):
        self.peaktimes = data
        self.ax.plot(self.peaktimes, self.data[self.
   peaktimes], 'x')
        self.ax.figure.canvas.draw()

    def plotdata_2d(self, data, xmin, xmax, ymin, ymax):
      self.data = data[xmin:xmax,ymin:ymax]
      if self.ax is not None:
          self.ax.remove()
      self.ax = self.canvas.figure.subplots()
      cmap = cm.get_cmap('YlGnBu')
      img = self.ax.imshow(self.data, cmap=cmap, extent=[
ymin,ymax,xmax,xmin])
      if self.colorbar is not None:
          self.colorbar.remove()
      self.colorbar = self.figure.colorbar(img,ax=self.ax)
      self.ax.format_coord = lambda x,y: 'x = %i, y = %i'
% (y, x)
      self.canvas.draw_idle()

  def plotpeaks_2d(self, data, xmin, xmax, ymin, ymax,
marker, plottype):
      self.data = data[xmin:xmax,ymin:ymax]
      if self.ax is not None:
          self.ax.remove()
      self.ax = self.canvas.figure.subplots()
      if self.colorbar is not None:
          self.colorbar.remove()
      if plottype is 0:
          self.data = np.ma.masked_where(self.data < 0,
self.data)
          cmap = cm.get_cmap('inferno', 2)
          img = self.ax.imshow(self.data, cmap=cmap,
extent=[ymin,ymax,xmax,xmin])
          self.colorbar = self.figure.colorbar(img,ax=self
.ax)
          self.colorbar.set_ticks([0.25, 0.75])
          self.colorbar.set_ticklabels(['0', '1'])
      elif plottype is 1:
          self.data = np.ma.masked_where(self.data <= 0,
self.data)
          cmap = cm.get_cmap('summer')
          cmap.set_bad(color='black')
          img = self.ax.imshow(self.data, cmap=cmap, norm=
colors.PowerNorm(gamma=0.5), extent=[ymin,ymax,xmax,xmin
])
          self.colorbar = self.figure.colorbar(img,ax=self
.ax)
      if marker is True:
          self.data = data
```

```
            self.data = np.ma.masked_where(self.data > -2,
self.data)
            self.ax.imshow(self.data, cmap='Spectral',
extent=[ymin,ymax,xmax,xmin])
            self.data = data
            self.data = np.ma.masked_where(self.data != -1,
self.data)
            self.ax.imshow(self.data, cmap='hsv', extent=[
ymin,ymax,xmax,xmin])
        self.ax.format_coord = lambda x,y: 'x = %i, y = %i'
% (y, x)
        self.canvas.draw()

    def plot_subplots(self, data, xmin, xmax, ymin, ymax, th
):
        x = xmax-xmin
        y = ymax-ymin
        self.analysis.th = th
        self.ax = self.canvas.figure.subplots(x, y, sharex=
True)
        for i in range (xmin, xmax):
            for j in range(ymin, ymax):
                dataaux = data[:,i-self.analysis.xmin,j-self
.analysis.ymin]
                if xmin==(xmax-1) or ymin==(ymax-1):
                    if xmin==(xmax-1):
                        indaux = j-ymin
                    elif ymin==(ymax-1):
                        indaux = i-xmin
                    self.ax[indaux].plot(dataaux)
                    self.ax[indaux].set_title("(" + str(i) +
 "," + str(j) + ")", fontsize=8)
                    self.ax[indaux].tick_params(labelsize=6)
                    self.analysis.compute_peaks(dataaux)
                    self.ax[indaux].plot(self.analysis.
peaktimes, dataaux[self.analysis.peaktimes], 'x')
                else:
                    self.ax[i-xmin, j-ymin].plot(dataaux)
                    self.ax[i-xmin, j-ymin].set_title("(" +
str(i) + "," + str(j) + ")", fontsize=8)
                    self.ax[i-xmin, j-ymin].tick_params(
labelsize=6)
                    self.analysis.compute_peaks(dataaux)
                    self.ax[i-xmin, j-ymin].plot(self.
   analysis.peaktimes, dataaux[self.analysis.peaktimes], 'x'
   )

    def plotdata_3d(self, time, data, xmin, xmax, ymin, ymax
   ):
        self.time = time
        self.data = (data[self.time,xmin:xmax,ymin:ymax]).T
        self.ax.remove()
        if self.colorbar is not None:
            self.colorbar.remove()
            self.colorbar = None
        self.ax = self.canvas.figure.add_subplot(111,
   projection='3d')
        X = np.arange(ymin,ymax)
        Y = np.arange(xmin,xmax)
        X, Y = np.meshgrid(X, Y)
        self.ax.plot_surface(X, Y, self.data)
        self.canvas.draw()
```

Python file 6: table_window.py

```
import sys
sys.path.insert(0, '../')
from PyQt5.QtCore import Qt, QAbstractTableModel, QVariant
from PyQt5.QtWidgets import QMainWindow, QTableView
from PyQt5.uic import loadUi
import numpy as np
import math


class MyTableModel(QAbstractTableModel):

    def __init__(self, data, parent=None, *args):
        QAbstractTableModel.__init__(self, parent, *args)
        self.data = data
        self.datalist = np.zeros((len(self.data),len(self.
   data[0])))

    def flags(self, index):
        return Qt.ItemIsEnabled | Qt.ItemIsSelectable

    def rowCount(self, parent):
        return len(self.data)
    def columnCount(self, parent):
        return len(self.data[0])

    def data(self, index, role):
        if not index.isValid():
            return QVariant()
        elif role != Qt.DisplayRole:
            return QVariant()
        return QVariant(list(self.data[index.row()].values()
   )[index.column()])

    def headerData(self, column, orientation, role=None):
        if role != Qt.DisplayRole:
            return QVariant()
        if orientation == Qt.Horizontal:
            return QVariant(list(self.data[0].keys())[column
   ])

    def sort(self, Ncol, order):
        for i in range(len(self.datalist)):
            self.datalist[i] = [j for j in list(self.data[i
   ].values())]
        self.layoutAboutToBeChanged.emit()
        arg = (np.argsort(np.argsort(self.datalist[:,Ncol]))
   ).tolist()
        zipped=zip(arg,self.data)
        self.data = [x for _, x in sorted(zipped)]
        if order == Qt.AscendingOrder:
            self.data.reverse()
        self.layoutChanged.emit()

class TableWindow(QMainWindow):

    def __init__(self, parent=None, data=None, timedata=None
   ):
        super().__init__(parent)
        loadUi('../_res/ui/table_mainwindow.ui',self)
        self.data = data
        self.timedata = timedata
        self.model_data = []
        self.model_timedata = []
        self.model_timecoordsdata = []
        self.tablemodel3 = None
```

```
     self.create_data()
     self.tablemodel = MyTableModel(self.model_data, self
)
     self.tableView.setModel(self.tablemodel)
     self.tableView.sortByColumn(2,Qt.AscendingOrder)
     self.layout = self.horizontalLayout.layout()
     self.selmodel = self.tableView.selectionModel()
     self.selmodel.selectionChanged.connect(self.
on_rowSel)

 def create_data(self):
     for i in range(len(self.data)):
         item = {}
         item['x'] = str(int(self.data.index[i][0]))
         item['y'] = str(int(self.data.index[i][1]))
         item['peaks number'] = str(int(self.data.values[
i]))
         self.model_data.append(item)

 def on_rowSel(self, sel):
     if self.model_timedata:
         if self.model_timecoordsdata:
             self.tableView_3.deleteLater()
             self.tableView_3 = None
             self.model_timecoordsdata = []
         self.tableView_2.deleteLater()
         self.tableView_2 = None
         self.model_timedata = []
     self.row = (sel.indexes())[0].row()
     self.create_timedata()
     self.tablemodel2 = MyTableModel(self.model_timedata,
 self)
     self.tableView_2 = QTableView()
     self.tableView_2.setModel(self.tablemodel2)
     self.selmodel_2 = self.tableView_2.selectionModel()
     self.selmodel_2.selectionChanged.connect(self.
on_timeSel)
     self.layout.addWidget(self.tableView_2)
     self.tableView_2.setSelectionBehavior(QTableView.
SelectRows)
     self.tableView_2.setVerticalScrollBarPolicy(Qt.
ScrollBarAlwaysOn)
     self.tableView_2.setMinimumWidth(220)
     self.tableView_2.setMaximumWidth(220)
     self.tableView_2.setSortingEnabled(True)
     self.setMaximumWidth(525)
     self.tablemodel3 = None

 def create_timedata(self):
     self.x = int(list((self.tablemodel.data[self.row]).
values())[0])
     self.y = int(list((self.tablemodel.data[self.row]).
values())[1])
     for i in range (len(self.timedata[self.x][self.y])):
         item2 = {}
         item2['peak time'] = str(int(self.timedata[self.
x][self.y].index[i]))
         item2['peak height'] = str(int(round(self.
timedata[self.x][self.y].values[i])))
         self.model_timedata.append(item2)

 def on_timeSel(self, sel):
     if self.model_timecoordsdata:
         self.model_timecoordsdata = []
         self.layout.removeWidget(self.tableView_3)
     timeind = (sel.indexes())[0].row()
     self.time = int(list((self.tablemodel2.data[timeind
]).values())[0])
     self.create_timepixelsdata()
     self.tablemodel3 = MyTableModel(self.
model_timecoordsdata, self)
     self.tableView_3 = QTableView()
     self.tableView_3.setModel(self.tablemodel3)
     self.layout.addWidget(self.tableView_3)
     self.tableView_3.setMinimumWidth(605)
     self.tableView_3.setMaximumWidth(605)
     self.tableView_3.setSortingEnabled(True)
     self.tableView_3.sortByColumn(4,Qt.DescendingOrder)
     self.tableView_3.setSelectionBehavior(QTableView.
SelectRows)
     self.selmodel_3 = self.tableView_3.selectionModel()
     self.selmodel_3.selectionChanged.connect(self.
on_pixSel)
     self.pixrow = None

 def create_timepixelsdata(self):
     self.range = range((int(self.time)-5), (int(self.
time)+5))
```

```
    acc_dist = []
    for i in range (len(self.timedata)):
       if self.timedata.index[i][2] in self.range:
          item3 = {}
          item3['x'] = str(int(self.timedata.index[i
][0]))
          item3['y'] = str(int(self.timedata.index[i
][1]))
          item3['peak time'] = str(int(self.timedata.
index[i][2]))
          item3['peak height'] = str(int(round(self.
timedata.values[i])))
          dist = round(math.sqrt(((self.timedata.index
[i][0]-self.x)**2)+(self.timedata.index[i][1]-self.y)**2)
,1)
          item3['pixel distance'] = str(dist)
          item3['time distance'] = abs(int(self.
timedata.index[i][2])-int(self.time))
          self.model_timecoordsdata.append(item3)
          acc_dist = np.concatenate((acc_dist, np.
array([dist])), axis=0)

 def on_pixSel(self, sel):
    self.pixrow = (sel.indexes())[0].row()
    self.pix_x = int(list((self.tablemodel3.data[self.
pixrow]).values())[0])
    self.pix_y = int(list((self.tablemodel3.data[self.
pixrow]).values())[1])
```

Literature

[1] G. Bertotti, D. Velychko, N. Dodel, S. Keil, D. Wolansky, B. Tillak, M. Schreiter, A. Grall, P. Jesinger, S. Rohler, M. Eickenscheidt, A. Stett, A. Moller, K. H. Boven, G. Zeck, and R. Thewes, "A CMOS-based sensor array for in-vitro neural tissue interfacing with 4225 recording sites and 1024 stimulation sites," *IEEE 2014 Biomedical Circuits and Systems Conference, BioCAS 2014 - Proceedings*, pp. 304–307, 2014.

[2] J. Dragas, D. Jackel, F. Franke, and A. Hierlemann, "An unsupervised method for on-chip neural spike detection in multi-electrode recording systems," *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBS*, pp. 2535–2538, 2013.

[3] M. Jenkner, M. Tartagni, A. Hierlemann, and R. Thewes, "Cell-based CMOS sensor and actuator arrays," *IEEE Journal of Solid-State Circuits*, vol. 39, no. 12, pp. 2431–2437, 2004.

[4] G. Bertotti, F. Jetter, N. Dodel, S. Keil, C. Boucsein, A. Moller, K. H. Boven, G. Zeck, and R. Thewes, "Artifact-compensated time-continuous recording from neural tissue during stimulation using a capacitively coupled in-vitro CMOS-MEA with 4k recording and 1k stimulation sites," *Proceedings - 2016 IEEE Biomedical Circuits and Systems Conference, BioCAS 2016*, pp. 256–259, 2016.

[5] S. Gibson, J. W. Judy, and D. Marković, "Technology-aware algorithm design for neural spike detection, feature extraction, and dimensionality reduction," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 18, no. 5, pp. 469–478, 2010.

[6] C. Leibig, T. Wachtler, and G. Zeck, "Unsupervised neural spike sorting for high-density microelectrode arrays with convolutive independent component analysis," *Journal of Neuroscience Methods*, vol. 271, pp. 1–13, 2016. [Online]. Available: http://dx.doi.org/10.1016/j.jneumeth.2016.06.006

[7] Python Software Foundation, "About Python™ — Python.org," 2019. [Online]. Available: https://www.python.org/about/

[8] Real Python, "Object-Oriented Programming (OOP) in Python 3," 2019. [Online]. Available: https://realpython.com/python3-object-oriented-programming/

[9] T. Oliphant, "NumPy: A guide to NumPy," USA: Trelgol Publishing, 2006–. [Online]. Available: http://www.numpy.org/

[10] E. Jones, T. Oliphant, P. Peterson *et al.*, "SciPy: Open source scientific tools for Python," 2001–. [Online]. Available: http://www.scipy.org/

[11] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing In Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[12] The Linux Information Project, "GUI Definition," 2004. [Online]. Available: http://www.linfo.org/gui.html

[13] PyQT, "Pyqt5 reference guide," 2015. [Online]. Available: https://www.riverbankcomputing.com/static/Docs/PyQt5/

[14] The Qt Company, "Qt: Cross-platform software development for embedded desktop," 2019. [Online]. Available: https://www.qt.io

[15] F. Pérez and B. E. Granger, "IPython: a system for interactive scientific computing," *Computing in Science and Engineering*, vol. 9, no. 3, pp. 21–29, May 2007. [Online]. Available: https://ipython.org

[16] W. McKinney, "Data structures for statistical computing in python," in *Proceedings of the 9th Python in Science Conference*, S. van der Walt and J. Millman, Eds., 2010, pp. 51 – 56.