Jan Dabrowski, 1765583
Individual Study 1
2019/20
Semester 1
Supervisor: Prof. Hamid Dehghani

Source code available at:
https://git.cs.bham.ac.uk/jxd786/nirs

Software for data acquisition, visualization and analysis for functional near-infrared spectroscopy

My task was to create a production-ready software for visualization and analysis of data from the spectrometer used for functional near-infrared spectroscopy of the brain (fNIRS) in Centre for Human Brain Health (CHBH) at University of Birmingham. During the period of 10 weeks, I created a tool for researchers that will enable effectively identify incorrectly adjusted fiber-optics and visually observe collected data. To achieve that I had to overcome several technical and non-technical challenges, some of them related to fast real-time network communication, understanding the format of the input data and creating user-friendly drag-and-drop configuration editor. In this report, I will describe my process of learning how to resolve some of those issues, what decisions I made and what the consequences of them were. First, I will describe necessary domain knowledge I gained about the task, about the data and associated technology, then I will describe my solution to the problem, including the description of selected technologies, architecture and software engineering techniques. Finally, I will adopt Data Mining techniques to gain insight into the nature of the data, find patterns and design further improvements of the software.

1. Understanding the domain

Functional near-infrared spectroscopy measures changes in blood oxygenation level in the near-scalp layer of the brain by detecting the absorption and phase shift of the near-infrared light.

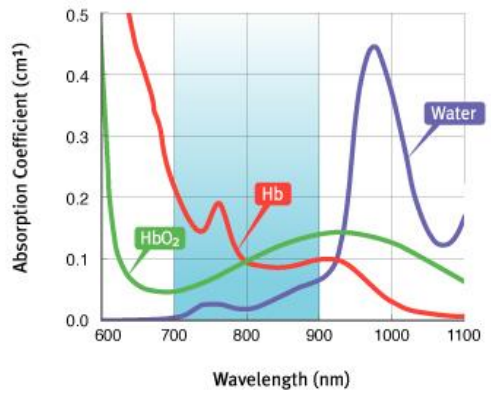

Fig. 1: Wavelength absorption by different materials (ISS, Inc. 2019a. Used with permission)

fNIRS takes advantage of the spectrum of light where skin, tissue and bone are mostly transparent, and haemoglobin is a strong absorber. (ISS Inc, 2019a). Variations of oxygenated and deoxygenated haemoglobin reflect the neuronal activity and can be used to diagnose brain injuries, psychiatric disorders and study of brain activity related to emotions or mobility. This technology is currently widely researched topic due to its safety, non-invasiveness, portability and high temporal resolution. (Boas et al., 2014). One of the problems with successful high-quality data acquisition with fNIRS is its sensitivity to physiological noise not directly related to brain activity, head movement and light from external sources. To minimize the noise in analysed data several pre-processing steps are usually applied, including checking if the usual heartbeat oscillation is present in the signal amplitude (Pinti et al., 2019). Nevertheless, some of the factors can be minimized before or during the data recording session if the operator of the spectrometer is able to visually inspect data collected in each channel. For example, one needs to check if all sources and detectors (optodes) are correctly adjusted and touch the skin in the correct angle. Therefore, convenient visualization of the measured data in real-time is very important to achieve high-quality measurements, and this is the main purpose of the software I am creating.

2. Understanding the data

The stream of measurements needs to be parsed, rearranged and displayed to the user in the configured format. The layout of the displayed values should be easily editable by the user because the relative position of sources and detectors can change dynamically. Data collected by the fNIRS spectrometer can be in various formats, depending on the machine manufacturer, setup and configuration. The spectrometer used in CHBH was manufactured by ISS, Inc and consists of 32 sources and 30 detectors. The device collects data in cycles, a full-cycle contains data from all source-detector pairs. Sources emit light at two wavelengths: 690 nm

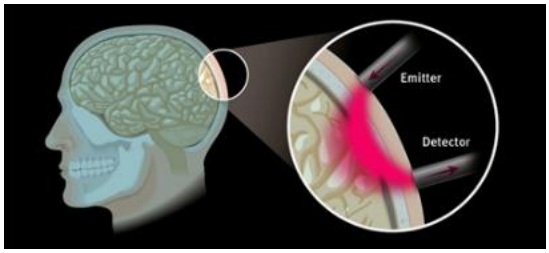

Fig. 2: Light penetration in brain tissue (ISS, Inc. 2019a, Used with permission)

and 830 nm and based on selected time multiplexing and frequency multiplexing scheme. There are 3 modes available – switch 8, 16 and 32, in each mode respective number of sources – 8, 4, 2 are simultaneously turned ON (ISS Inc, 2019b). The number of sources emitting light at once defines the size of a single data point collected in a cycle. Each measurement – a source-detector pair – is described as a complex number, the result of the Fourier transform.

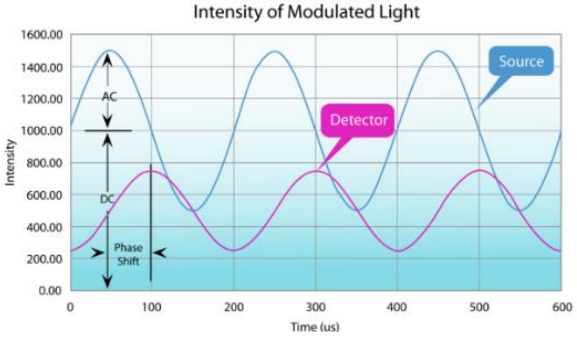

Fig. 3: Source – Detector signal modulation (ISS, Inc. 2019a, Used with permission)

Before displaying, complex numbers need to be converted to AC and Phase Shift values:

$$AC = \sqrt{Re(z)^2 + Im(z)^2}$$

$$Phase = \text{atan}\left(\frac{Im}{Re}\right)$$

3. Integration requirements

The fNIRS spectrometer is operated by the connected computer with manufacturer's software called DMC. Communication with DMC is available through raw TCP/IP connection, which can be used to change some of the data collection configurations and to read a stream of measurements. Configuration can be changed by sending and reading simple ASCII text, however, the stream of measurements is received as binary messages which can be interpreted based on data structures from C++ header files provided by ISS Inc. An important factor of successful parsing of incoming data is correct alignment and size of variables in the structure. Typical C++ compilers add padding or even rearrange variables in

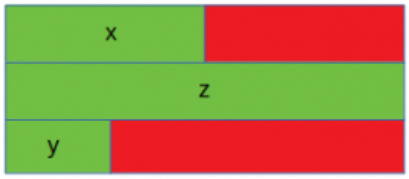

Fig. 4: C++ structure padding (Jain, no date)

structures. On the other hand, variables in binary data collected from DMC are packed sequentially in the order they are defined using just 1-byte boundaries. The correct alignment and size of variables can be validated with "magic numbers" – known constant fields in the incoming packets described in the structure definition and by comparing the value of the field "size" which is a part of the structure itself with the real size of the structure defined in the selected language.

4. Selecting technologies

Before choosing languages and technologies to build the solution I divided the problem into two components. First, the backend able to communicate with the manufacturer's software through TCP/IP and to efficiently consume a stream of binary data. Second, easily extendable and maintainable frontend, able to create sophisticated visualizations and intuitive, user-friendly interface. Because of the underlying complexity of backend data transformations, I decided to keep backend and frontend loosely coupled, allowing to run, develop and test each architecture tier in isolation. For both frontend and backend, I considered the following requirements: Maintainability of large projects, modern state-of-the-art frameworks, popularity and community support and if they are free and open source. Moreover, for the backend: fast low-level TCP/IP communication and easy integration with C++ structures. Eventually, I decided to use C# with just released version of .NET Core 3.0 framework for the backend. C# (8.0) is modern, object-oriented, type-safe programming language based on C/C++ containing similar features to Java and JavaScript such as Garbage collection, anonymous functions or unified type system. .NET Core is an open-source, cross-platform general-purpose development platform with many useful built-in features like IoC Container for Dependency Injection, a package manager (NuGet) and CLI (Microsoft, 2019a; Microsoft, 2019b). For the frontend, I used Vue.js, JavaScript framework for creating web user interfaces. It utilizes the concept of virtual DOM (Document Object Model). Vue.js keeps a tree-like structure of references to the real DOM in the memory, keeps track of the variables used to render the view and efficiently and asynchronously keeps it

consistent with the state of the application (comparable to React[1]). Moreover, it incorporates component system allowing developers to easily build complicated interfaces by abstracting them to a tree of components. (Vue.js, 2019a; Vue.js, 2019b)

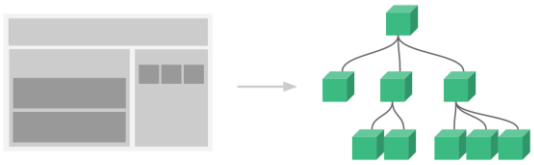

Fig. 5. GUI as a tree of components (Vue.js, 2019b)

Additionally, to make the application easily runnable, I have used Electron.NET community package to bind .NET Core and web-based interface together into a single executable file for any operating system. Electron.NET is based on Electron library, a popular framework for creating native cross-platform applications with web technologies. Some of popular products using Electron are Slack[2], VS Code[3] or Discord[4] (GitHub, 2019a). After discovering that there is no off-the-shelf project template using these 3 libraries, and there is a pending request from the community[5], I decided to create one[6] and made it publicly available on GitHub. The template makes use of VueCliMiddleware[7] package for .NET Core to redirect requests to Vue CLI server during development and to bundle created HTML, JS and CSS files for production and is the foundation for integration between different architecture components. The last missing piece of the architecture is a fast real-time communication channel between backend and frontend. My first approach took advantage of Electron Inter-Process Communication (IPC) to send messages between main and renderer processes (GitHub, 2019b), but messages had been proxied through Electron.NET and handled by a single thread in .NET Core, which turned out to be inefficient and caused delays in real-time data visualization.

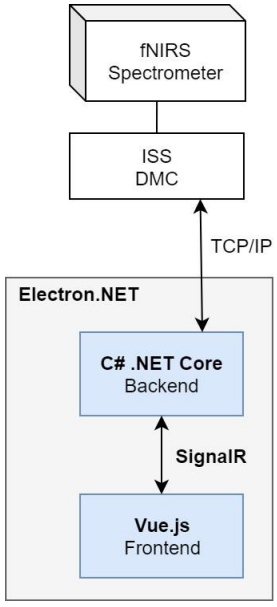

Fig. 6. Technology integration

Finally, I decided to switch to SignalR which is already a part of the .NET Core framework and has a corresponding JavaScript package for frontend. SignalR provides simple API for sending messages between clients in local network or on the internet, it intelligently selects the most efficient available channel for communication - Web Sockets, Ajax long-pooling or Iframe, among others (Microsoft, 2014). SignalR connects directly backend with frontend rendering engine (Chromium[8] used as a part of Electron) without any middleware and messages are handled by multiple threads.

4.1 Backend design

Backend architecture is divided into three interchangeable layers, each layer uses different models from distinct namespaces for data representation and implements a set of interfaces. This allows implementing accurate testing strategies to check whether each part of the application works correctly. One of the most useful applications of the layered architecture during development was to replace the communication with the DMC software with a dummy implementation imitating the expected behaviour.

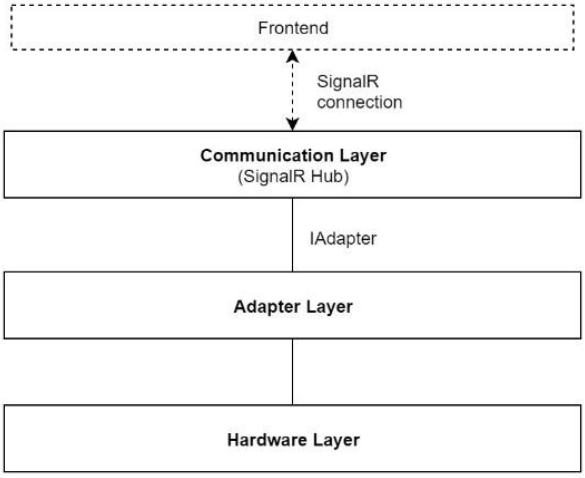

Fig. 7. Backend layers

Communication layer consists of a SignalR Hub, which is an abstraction provided by the framework to define methods callable by the connected clients. The hub expects an object implementing IAdapter interface which defines all interactions with the lower layer, the instance itself is managed by the framework service container and is injected at runtime. Each method handles specific functionality, mostly by communicating with the adapter layer and sending back a response to the

[1] https://reactjs.org/
[2]https://slack.com/
[3]https://code.visualstudio.com/
[4]https://discordapp.com/
[5]https://github.com/ElectronNET/Electron.NET/issues/265
[6]https://github.com/johndab/electron.net-vue
[7]https://github.com/EEParker/aspnetcore-vueclimiddleware
[8]https://www.chromium.org/Home

client that the requested action has been successfully executed. The adapter layer is where the integration with the third-party software starts. All components above the adapter layer can work correctly regardless of how and where the actual data was acquired. This layer closely relies on the hardware layer where raw text or binary messages are parsed and converted to higher-level model. ISSAdapter class implements all protocols necessary for the communication with the ISS fNIRS controller (DMC), it handles connection initialization and termination, settings management and controls continuous data stream reader. The DataReader class performs lowest-level processing of the incoming stream. When the acquisition is enabled, it utilizes definitions from the provided C++ headers to convert the binary stream to structures, store them in unmanaged memory and finally process them according to the selected switch for data collection. Because not all source-detector pairs contain valuable information, therefore when the layout of sources and detectors is changing the graph of requested connections is recomputed and used to filter only necessary values. Many critical aspects of the application have been tested with End-to-End tests. Each test starts the DMC software, performs a series of operations, validates the outcome and closes terminates the DMC process automatically. In this way, many checks related to integration can be performed in a few seconds.

4.2 Frontend design

While designing and developing the user interface I considered a set of assumptions. First, all users will know the domain, notations and hardware functionalities. Second, the primary function of the interface is interaction with spatial data, features related to the location of nodes collecting measurements should be designed very carefully and work seamlessly. Third, all unnecessary elements should be hidden to prioritize data presentation and to keep the interface clean and easy to use. A technique used to keep the final design of the interface consistent with the initial assumptions and requirements is “Wireframing”. Constructing the interface by dividing the space into boxes helps with correct prioritization of content and functionalities. Wireframes do not contain any styling, interactive content or images. (Garrett, 2011). As described earlier, easily scalable and maintainable user interfaces

Fig. 8: Tree of components

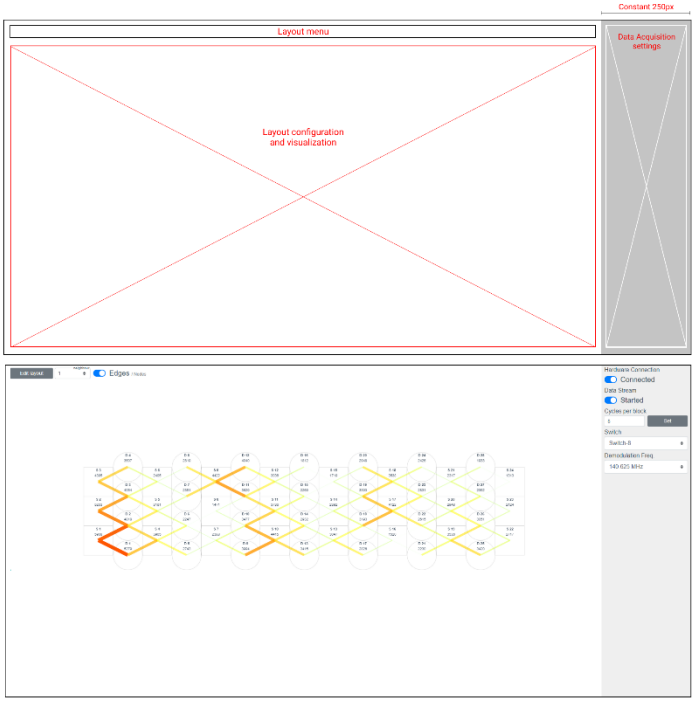

Fig. 9. Wireframe of the interface and the final design

in Vue.js divides application to components. Each component has a state and implements its visual representation and functional logic – e.g. a switch with the name “Hardware connection” manages the state of the connection and sends a request to the backend when the user clicks at the switch. Following the wireframe blueprint helps in separating elements of the interface to components that can be reused across the application and high-level containers responsible for navigation, data

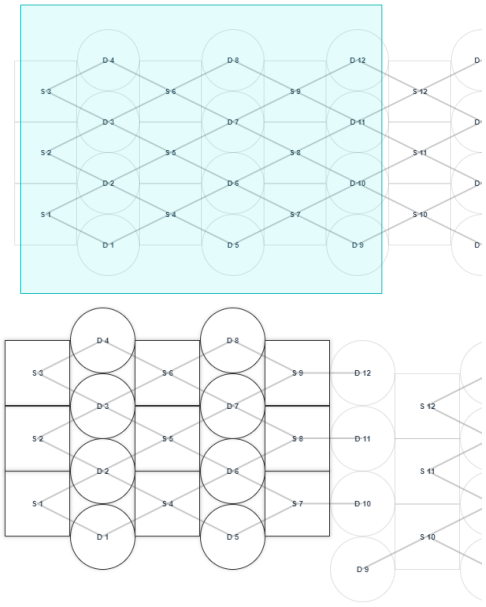

Fig. 10. Layout editor with selection box visible (top) and multiple elements selected and moved together

management or rendering control. The most advanced part of the interface is the layout editor.

To implement the user-friendly drag-and-drop solution I made use of an open-source library[9] providing basic interaction handling and position update. Then, I extended it by adding convenient functionalities for moving multiple elements comfortably, such as drawing a selection rectangle or selecting multiple elements while holding the "Shift" key. Moreover, layout renderer searches for the selected n-th nearest neighbour and displays the connection on the fly while elements are moved

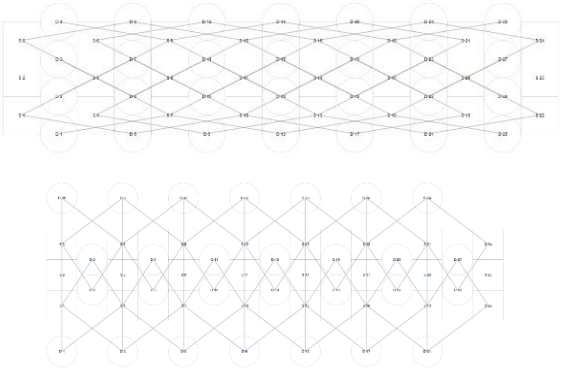

Fig. 11: Example layouts with connections, 3rd nearest neighbour (top) and 2nd nearest neighbour (bottom).

around. This information is necessary to validate the relative position of elements and values displayed during acquisition.

The interface is not fully responsive, but it is scalable to a certain degree, to display the layout always in the centre of the screen the position of each element is normalized before saving and calculated dynamically when the application starts or when the window is resized. The frontend is also responsible for initializing the connection with backend and handling reconnection. To make sure that the solution is robust to any communication-related issues I created my own abstraction layer for registering event listeners, dispatching messages and reinstantiating the connection. This service allowed me to easily switch from Electron's IPC communication to the SignalR (as described in 4.) and to test the user interface behaviour in isolation.

5. Data analysis

To further improve the software in helping to collect high-quality data I decided to design a set of additional mechanisms to detect and mark possibly incorrect measurements. In order to do that I asked for a dataset from any previous experiment collected on a human subject. Without any deeper insight about the conditions, machine setup or the examined subject I made use of Data Mining techniques to find potential nature of the values that can be useful for the user. For this set of experiments, I decided to use Python instead. Measurements from some source-detector channels show clear heartbeat oscillation, when analysed correctly it can determine the quality of the data.

First, after creating a function to consume the file with measurements, I searched for the heartbeat oscillation in the signal amplitude. Usual heart rate at rest ranges from 60 to 100 beats per minute (British Heart Foundation, 2017), which approximately gives 1-1.6 beats per second. Data were acquired at a rate of 40 measurements per second, hence I expect to find a trend every 40-64 measurements.

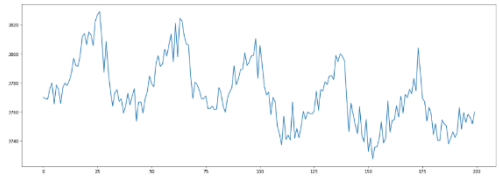

Fig. 12. Visible heartbeat oscillation between source no. 2 and detector no. 2 for 200 measurements (~5 seconds).

Although some source-detector pairs indeed show visible systematic oscillation, this pattern is not present for all channels. As noted by Pinti et. al (2015) absence of heartbeat oscillations signifies low-quality measurements, therefore that the channel should not be considered in further analysis.

A straightforward method to examine the relationship between variables that change together in time is a measure of covariance (Dunham, 2002). Values of such noisy channels would be mostly random and independent, therefore the covariance between them should be close to 0. For instance, from the covariance matrix (Fig. 14). I can deduce that there is a small amount of noise for channels between source 2 and detectors 1, 2 and 6. However, the covariance for source 2 between detector 4 and all other channels is very close to 0, therefore I do

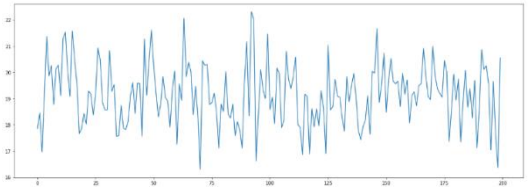

Fig. 13. No visible heartbeat oscillation between source no. 2 and detector no. 4 for 200 measurements (~5 seconds).

9 https://github.com/jbaysolutions/vue-grid-layout

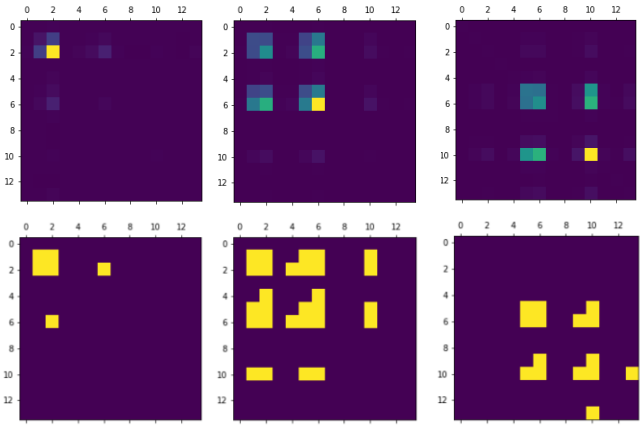

Fig. 14. Correlation matrix for 200 measurements between all detectors for sources 2, 5 and 8. Second row thresholded with absolute covariance greater than 15. Brighter colour signifies higher covariance.

not expect the heartbeat to be present (Fig. 13). Such estimation can be calculated efficiently for a batch of measurements and displayed in real-time to the user. Furthermore, as demonstrated by Perdue et al. (2014), raw intensity values can be used to measure the explicit heart rate of the subject by fitting a Gaussian function to the data. The pulse can be estimated by finding local maxima, manually tunning standard deviation and calculating the time between peaks. I approached the problem differently, the oscillation of the signal can be interpreted as the varying intensity of subsequent pixels.

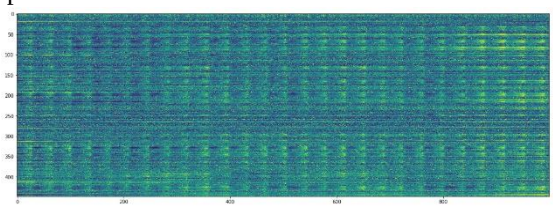

Fig. 15. Signals from all source-detector channels stacked together (y axis) for 1000 measurements (x axis)

Then the task of finding the pulse in the signal can be compared to detecting edges in a noisy image. One of the methods is to perform discrete convolution with the one dimensional Laplacian of Gaussian (LoG) operator. The Laplacian highlights regions of the signal where the intensity changes rapidly, to reduce the noise influence this operator can be combined with the estimation of the Gaussian smoothing filter. After convolving the signal with the LoG, precise position of the edge can be detected by finding the zero-crossing points. (Fisher, 2003).

Although this method is sensitive to noise, there are many well-known variations and improvements available, such as the use of hysteresis or thresholding. Most importantly, the convolution and zero-crossing can be applied to the incoming signal continuously with a very small cache, hence it is very efficient to use for real-time analysis.

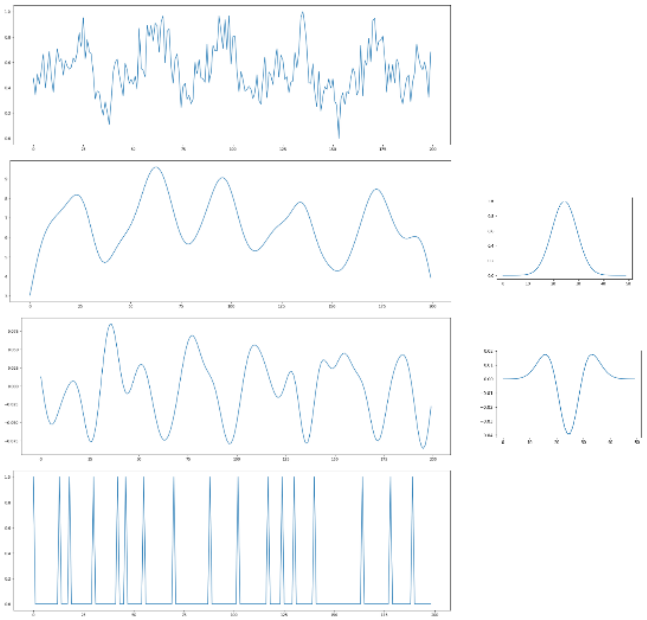

Fig. 16. From the top: raw signal, convolved with gaussian (mean=50, std=5), convolved with LoG (mean=50, std=5), zero crossing

Conclusion

To create high-quality software, it is necessary to gain a deeper understanding of the domain and to understand a wider context. I had to solve many problems on a wide variety of levels, research the necessary knowledge to make decisions and face the consequences. I had to understand the domain of near-infrared spectroscopy and the reason why the software for good data visualisation is needed. Then, I had to determine how to communicate with the machine and how to parse the data. Later, I designed and implemented the system capable of fulfilling the requirements. Finally, I experimented with the collected data to learn about techniques used for Data Mining and find potential future improvements of the system. I have learned a lot about many aspects of software development, from low-level communication and bytes alignment to high-level cross-technology integrations and user interface wireframing. Processing and analysing raw data is an opportunity to discover more knowledge and interesting insights. In the future, I want to study Data Mining more comprehensively, experiment with different techniques and on data from various domains.

References

Boas, D.A., Elwell, C.E., Ferrari, M. and Taga, G. (2014) 'Twenty years of functional near-infrared spectroscopy: introduction for the special issue.' *NeuroImage*, 85 (1), pp. 1-5, doi: 10.1016/j.neuroimage.2013.11.033

British Heart Foundation (no date) *What is a normal pulse rate?* Available at: https://www.bhf.org.uk/informationsupport/heart-matters-magazine/medical/ask-the-experts/pulse-rate (Accessed 07 December 2019).

Dunham, M.H. (2002) *Data Mining: Introductory and Advanced Topics*. Upper Saddle River, N.J.: Prentice Hall/Pearson Education.

Fisher, R., Perkins, S., Walker, A. and Wolfart, E. (2000) Available at: https://homepages.inf.ed.ac.uk/rbf/HIPR2/log.htm (Accessed 07 December 2019).

Garett, J.J. (2010) *The Elements of User Experience: User-Centered Design for the Web and Beyond*. 2nd edn. Berkeley, CA: New Riders.

GitHub (2019a) *Electron Apps.* Available at: https://electronjs.org/apps (Accessed 07 December 2019).

GitHub (2019b) *Electron Documentation*. Available at: https://electronjs.org/docs (Accessed 07 December 2019).

ISS, Inc. (2019a) *Imagent*. Available at: http://www.iss.com/biomedical/instruments/imagent.html (Accessed 07 December 2019).

ISS, Inc. (2019b) *Imagent for fNIRS and EROS measurements.* Available at: http://www.iss.com/resources/pdf/technotes/Imagent_fNIRS_EROS_Measurements.pdf (Accessed 07 December 2019).

Jain, R. (no date) *Is sizeof for a struct equal to the sum of sizeof of each member?* Available at: https://www.geeksforgeeks.org/is-sizeof-for-a-struct-equal-to-the-sum-of-sizeof-of-each-member/ (Accessed 07 December 2019).

Microsoft Corporation (2014) *Introduction to SignalR* Available at: https://docs.microsoft.com/en-us/aspnet/signalr/overview/getting-started/introduction-to-signalr (Accessed 07 December 2019).

Microsoft Corporation (2019a) *About .NET Core* Available at: https://docs.microsoft.com/en-us/dotnet/core/about (Accessed 07 December 2019).

Microsoft Corporation (2019b) *A Tour of the C# Language*. Available at: https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/ (Accessed 07 December 2019).

Perdue, K. L., Westerlund, A., McCormick, S. A. and Nelson, C. A. (2014) 'Extraction of heart rate from functional near-infrared spectroscopy in infants'. *Journal of biomedical optics*, 19(6), 067010. doi:10.1117/1.JBO.19.6.067010

Pinti, P., Scholkmann, F., Hamilton, A., Burgess P. and Tachtsidis I. (2019) 'Current Status and Issues Regarding Pre-processing of fNIRS Neuroimaging Data: An Investigation of Diverse Signal Filtering Methods Within a General Linear Model Framework', *Front. Hum. Neurosci.*, doi: 10.3389/fnhum.2018.00505

Vue.js (2019a) *Comparison with Other Frameworks.* Available at: https://vuejs.org/v2/guide/comparison.html (Accessed 07 December 2019).

Vue.js (2019b) *Introduction*. Available at: https://vuejs.org/v2/guide/index.html (Accessed 07 December 2019).