CREAM

Deliverable D3.3

"Report on the development of a computational platform for studying network properties of creative brain"

CONTRACT NO	CREAM - 612022
TYPE OF DOCUMENT	Deliverable: D3.3
DATE	04/06/2015
ABSTRACT	This deliverable reports on the development of a computational platform, which makes use of heterogeneous multicore CPU/GPU architecture specified in D2.3, for revealing network patterns of creative brain from neurophysiological data.
AUTHOR, COMPANY	J. García-Prieto, E. Pereda, J.J. González (ULL) J. Bhattacharya (GOLDSMITHS')
VALIDATOR, COMPANY	R. Sladky (MUW)
WORKPACKAGE	WP3 – Task 3.3
NATURE	**R**: Report
CONFIDENTIALITY LEVEL	**PU:** Public

DOCUMENT HISTORY

Release	Date	Reason of change	Status	Distribution
R0.1	02/05/2015	Creation	Done	Mailing List
R0.2	26/05/2015	Sent for internal review	Done	Mailing List

R0.3	28/05/2015	Modified after first internal review feedback	Done	Mailing List
R0.4	29/05/2015	Submitted final version to the Coordinator for formal review and submission to the EC	Done	Mailing list

Table of Contents

1. Introduction

The assessment of functional (FC) and effective (EC) brain connectivity [1] has become one of the most active fields of research in systems neuroscience. Indeed, it is already well-stablished that some brain functions are not localized in specialized areas or modules but rather they are due to the interactions between brain areas (see, e.g., [2] and references therein). Thus, functional specialization and integration are synergetic concepts, and a thorough study of brain function from EEG data necessarily involves the estimation of the statistical dependence between the signals produced and simultaneously recorded from several brain areas, with (EC) or without (FC) information of the directionality in this dependence[1], as sketched in Figure 1.

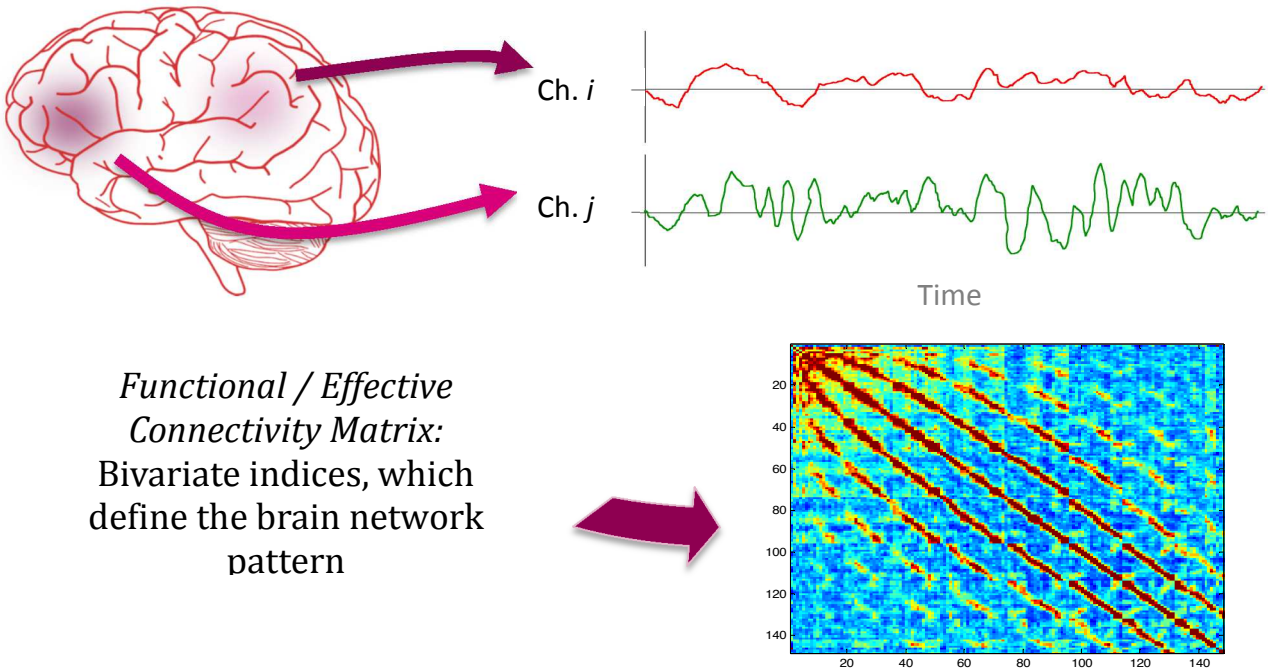

Figure 1: Schematic representation of the steps involved in the estimation of brain network patterns from EEG data. The activity of different brain areas is simultaneously recorded (top) and the statistical dependence (functional connectivity) between any two of these signals is estimated using different indices, which gives rise to a square, real valued, symmetric matrix (the network pattern, bottom). Should the indices provide information about the direction of the dependence, the matrix is also square and real, but not symmetric.

[1] Some authors (e.g., [1]) prefer to reserve the term EC for those situations where the existence of a cause-effect relationship is explicitly studied by fitting the data to appropriate causal models such as dynamic causal modelling, and speak of *directional* FC instead of EC when using indices that provide information on the *direction* of the dependence.

Currently, there exists a plethora of bivariate indices for this purpose [3]–[5]. Frequency domain based measures, such as *phase synchronization* (PS) ones, are among the most commonly used, as the simpler, time-domain traditional measure termed *correlation coefficient* captures only linear associations between neural signals, and is prone to outliers in the data and its interpretation is not always straightforward. Information theory-based indices such as *mutual information* (MI)[6] are theoretically more suitable for this purpose, as they assess a more general form (both linear and nonlinear) of statistical association between two time series [7], and detect, in principle, both amplitude and cross-frequency synchronization [8]. Yet, the practical calculation of MI is complicated. Firstly, the calculation of MI depends on the reliable estimation of both the marginal and the joint probability density functions, which is known to be a very complicated task for short, noisy time series [9], [10] (but see also [11]). Secondly, MI is non-normalized, which means that it is (theoretically) zero for completely independent signals but, unlike most other FC measures, it is not 1 for completely dependent signals. Should these complications be overcome, however, MI is deemed as one of the most powerful (both in general terms and statistically speaking [7]) FC measures at hand.

Along with PS-based indices and MI, there is a third set of FC measures that are worth assessing when analysing brain activity from EEG data, namely those based on the concept of *generalized synchronization* (GS, [3], [12]–[14]). Such indices require the previous reconstruction of the state spaces of the systems under study from their time series, normally using the well-known Takens theorem [15], and the estimation of distances between delayed vectors in high dimensional spaces. Yet in return for this added complexity, when carefully tailored [16], they are able to provide information not only on the extent of dependence, but also on its directionality, which make them an excellent complement to both PS-based indices and MI.

Recently, researchers in the field have recognized the importance of developing computational platforms / toolboxes, that integrate many of these FC indices in a way that can make them accessible to a wider scientific community (see, e.g., [5], [17] for outstanding recent examples). The computational cost associated with some of these measures has also made apparent the need of using high-performance computing facilities such as multicore CPU and/or GPUs [18] to estimate these indices in a reasonable time. Indeed, in the framework of the current CREAM project, if we are to estimate the FC /EC patterns of a subject from his/her EEG and use this information to decide whether and how to stimulate his/her brain using transcranial current stimulation (either direct (tdCS) or alternate (taCS) current) during a single session, it is necessary to develop a computational platform that can carry out the calculation of the indices very quickly and efficiently, by using the software (SW)/hardware (HW) system designed according to the specifications of D2.3. Namely, this platform should include custom-written libraries in a very efficient, general purpose, and system-independent programming code such as C/C++, and should be possible to use it from the MATLAB environment to facilitate its integration with the rest of the libraries of the project such as those devoted to pre-processing the data, reconstructing the neural sources from the scalp EEG and to produce the stimulation. Moreover, this platform should take full advantage of the heterogeneous multicore CPU/GPU architecture available, so that it has to be optimized to estimate the connectivity patterns in the most efficient, fastest possible way thanks to the computing power provided by this HW.

This deliverable is precisely aimed at describing such a platform for revealing network patterns in the brain from EEG data, its main features and the improvements obtained in the estimation of the FC/EC indices as compared to currently existing toolboxes by comparing their

performances in a HW/SW system as the one described in D2.3. This platform is already being used in WP3.4 and 3.5.

2. Executive summary (always public)

The estimation of network patterns from EEG activity requires the implementation of a computational platform that allows carrying out all the necessary processing steps fast and smoothly. Besides, a comprehensive description of such patterns entails the calculation of different indices of functional and effective connectivity, which analyse different aspects of EEG signal. We developed such a platform consisting on a set of very fast software libraries that efficiently compute three sets of functional connectivity indices (phase synchronization and generalized synchronization ones, as well as mutual information) using the API OpenMP. This API allows multi-platform shared memory multiprocessing programming in C, thereby allowing to take advantage of modern multicore CPU architectures. We also produce an extensive set of realistic simulations on simulated data comprising a wide range of sample lengths and sensor numbers to compare the performance of the developed platform as compared to existing toolboxes in many different situations, as well as that of custom-made GPU-based implementations in MATLAB. The results show that our libraries greatly outperforms (by at least one order of magnitude in the worst case) the fastest CPU-based implementations. Instead, the GPU-based ones, while extremely efficient in the calculations, suffer from the bottleneck, inherent to these devices, of the speed of data transfer from system's to GPU's memory and backwards. This means that they will be only competitive, as compared to CPU implementations, for very large number of calculations (i.e., for –possibly unrealistic- number of variables/samples).

3. HW and SW aspects

3.1. HW platform

Figure 2 shows the HW architecture on which the computational SW platform described here was tested. This HW was chosen to mimic the one described in D2.3, with the only difference that the CPUs were slightly faster (3.0 vs. 2.6 GHz) and had a 20% larger cache (25 vs. 20MB). Although the number of cores per CPU was also higher in our platform (10 vs. 8), we only used a maximum of 8 in the simulations.

Specifically, the components of our HW platform were:

- Motherboard SuperMicro X9DRG-OTF-CPU with 8 PCIe ports;
- 2x Ten-Core Intel Xeon Processor E5-2690 v2 @3.00GHz 25MB Cache;
- 1x NVIDIA GTX Titan graphics cards;
- 64 GB RAM;

all of which are consumer hardware components.

For details on the reasons behind the choice of these components as well as specifics on any of them, please refer to D2.3.

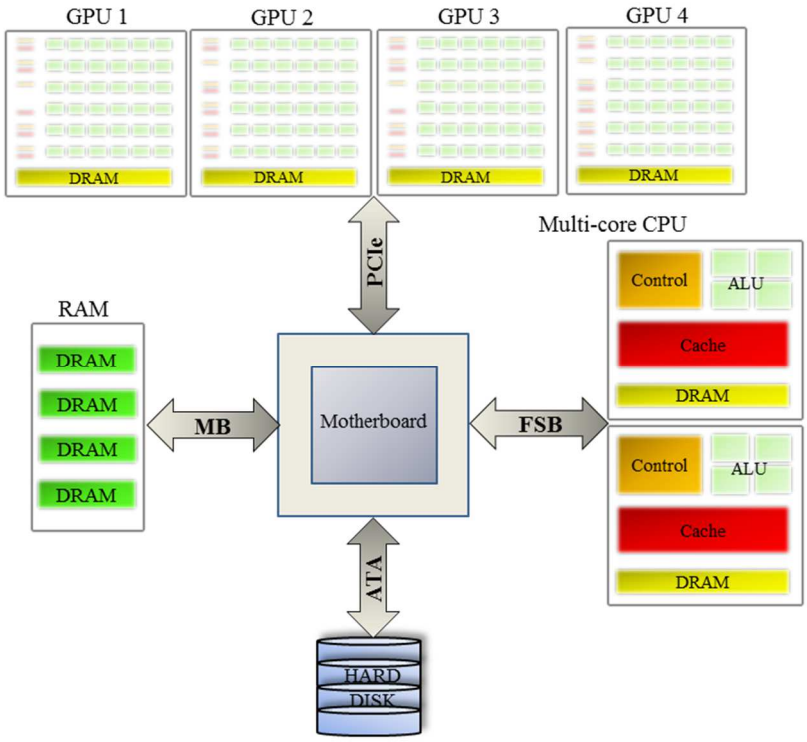

Figure 2: Structure of the computational platform in which the algorithms were tested, as described in D2.3

3.2. Software aspects

As detailed also in D2.3 (section 3.1), we decided to develop software having MATLAB or Simulink user interfaces, with the corresponding high performance libraries written in C code and compiled as MEX files to be called from MATLAB environment. In particular, we used the following version of MATLAB, and its toolboxes, running on Windows 8.1 x64 for development and testing of the MATLAB wrappers and the GPU implementations[2]:

- MATLAB 8.1.0.604 (R2013a)
- Digital Signal Processing Toolbox v8.4
- Signal Processing Toolbox v6.19

[2] To ensure forward compatibility, we have also checked that the libraries work on the latest available version of MATLAB and its corresponding toolboxes (namely, MATLAB Version: 8.5.0.197613 (R2015a), Digital Signal Processing Toolbox v9.0, Signal Processing Toolbox v7.0, Parallel Computer Toolbox v6.6, Statistical and Machine Learning Toolbox10.0 and MATLAB compiler v6.0)

- Parallel Computing Toolbox v6.2
- Statistics Toolbox v8.2
- MATLAB Compiler v4.18.1.

For the development of the portable C code, we only make use of the standard C library, thus making the code portable and platform independent. Although we used latest version (3.3.4, released Nov. 2014) of the FFTW subroutine library (http://www.fftw.org/) for computing the discrete Fourier transform (DFT) of arbitrary input sizes. This library is free software, distributed under the terms of the GNU General Public License with versions for all mayor operating systems, and to the best of our knowledge according to the benchmarks publicly available at http://www.fftw.org/benchfft/, FFTW's performance is typically superior to that of other publicly available FFT software and is even competitive with vendor-tuned codes. In those cases, were one specific routine is based on previously existing software in the same programming language (e.g., the MILCA routine for MI implementation), we explicitly indicate it in the text.

For the development of the portable C programs and compilation of the MEX files we used Microsoft Visual Studio 2010 Ultimate, which supports OpenMP v2.0.

4. Overview of the different FC/EC algorithms in the platform

As indicated in the Introduction, and according to the latest findings in the literature, we have included three different families of FC/EC indices in the platform. Namely, PS indices, GS indices and mutual information. Henceforth, we briefly describe each of these families and the indices selected within them.

4.1. PS indices

PS indices are arguably amongst the most popular methods to estimate functional connectivity between two neurophysiological signals (and more specifically, EEGs –see, e.g., [19]–[21] for recent examples). The concept of PS and its estimation from time series dates back to 1996, when Rosenblum and colleagues [22], in a seminal paper, demonstrated that, in regimes of weak coupling, the phases of two (possibly chaotic) oscillators may synchronize even if their amplitudes remain uncorrelated. Shortly afterwards, Tass and colleagues [23] demonstrated the applicability of this concept in human neuroscience. Since then, there have been many applications in different contexts, where PS indices have proven successful to estimate the degree of FC between two EEGs (see, e.g., [3], [19] and references therein). Estimating PS between two signals consists of three different sequential steps.

1) Define the phase of each individual signal.

Thus, given a broad band, real valued signal $x_k(t)$ (recorded from an EEG electrode or alternatively, representing the activity of a reconstructed neural source as estimated from the scalp EEG using any source reconstruction method), we have to transform it into a complex-

valued, narrow band[3] one using a suitable mathematical transformation (such as the Hilbert, the Fourier of the Wavelet transform[4]. In our case, we first filtered the signals in the frequency band of interest by means of a non-recursive or Finite Impulse Response (FIR) filter, which was applied twice to the data, once forwards once backwards, to ensure, thanks to the linear response of the filter, that it produces exactly zero phase distortion, therefore not introducing any spurious phase coupling between the signals. Then, we applied the Hilbert transform (HT) to the data to produce the analytic representation of $x_k(t)$. Thus, we first obtain the HT of this signal:

$$\tilde{x}_k(t) = \frac{1}{\pi} p.v. \int_{-\infty}^{\infty} \frac{x_k(\tau)}{t-\tau} d\tau \qquad (1)$$

where p.v. denotes Cauchy principal value. Note that the HT is actually the convolution of $x_i(t)$ with the tempered distribution p.v. 1/πt, and can be obtained easily in the frequency domain as the product of both Fourier transforms.

Then, the analytic representation of $x_i(t)$ is the complex-valued signal defined as:

$$x_k^a(t) = x_k(t) + i\,\tilde{x}_k(t) \qquad (2)$$

From (2), it is straightforward to define the phase as:

$$\phi_k(t) = arctag \frac{\tilde{x}_k(t)}{x_k(t)} \qquad (3)$$

2) Use the phase to estimate the degree (if any) of PS.

Given two signals, the condition of PS is formally established as

$$\left| n\phi_k(t) - m\phi_l(t) \right| < const \qquad (4)$$

[3] In order to get a physically meaningful definition of the phase, it is necessary to have a narrow band signal. Alternative definitions of the phase, based on recurrences in the state space, which do not require filtering, are also possible, [44] yet their applicability in the present context remains to be proved. Besides, they require the estimation of cross recurrence plots, which greatly increases the computational costs associated to the calculation without any foreseeable benefit. Thus, we do not cover them here and do not include them in the platform.

[4] It has been demonstrated that, in the present framework, the three methods are essentially equivalent [45], [46]

where | | stands for absolute value. Namely, the difference between the two phases remains bounded for a combination of positive integers m and n (the case m=n=1 is the most usually found in practice). Practically, the estimation of PS consists in studying the distribution of the cyclic relative phase:

$$\varphi_{kl}(t) = (\phi_k(t) - \phi_l(t)) \bmod 2\pi \quad (5)$$

The most commonly used method to study this distribution is by calculating the *mean phase coherence* [24] (also termed as *phase locking value,* PLV):

$$PLV = \left| \left\langle e^{i\varphi_{kl}(t)} \right\rangle \right| \quad (6)$$

Where < > stands for average value. Defined in this way, (6) ranges between 0 (no PS) and 1 (perfect PS), although in real life data, its value is always in between these two extreme ones. In the analysis of neurophysiological data, especially scalp EEG, a known issue is the influence on PLV of volume conduction effects and the choice of the reference (see, e.g., [20], [21] for recent studies dealing with this issues). Thus, Stam and colleagues [25] suggested an alternative index, termed phase lag index (PLI), which is insensitive to zero lag coupling, and therefore is not affected by these two effects.

$$PLI = \left| \left\langle sign(\varphi_{kl}(t)) \right\rangle \right| \quad (7)$$

where sign (x) is the sign function (+1 if x>0; -1 if x<0). Vinck and colleagues [19] further improve this definition by introducing the weighted PLI (wPLI), by giving a different weight to each of the values of the relative phase (the closer they are to zero, the most likely they are to be affected by measurement noise, and the lower the weight they receive):

$$wPLI = \frac{\left| \left\langle img(e^{i\varphi_{kl}(t)}) \right\rangle \right|}{\left\langle \left| img(e^{i\varphi_{kl}(t)}) \right| \right\rangle} \quad (8)$$

where *img()* denotes imaginary part.

Although, in principle, (8) should be the optimal choice given its properties, it must be taken into account that there exists neurologically meaningful couplings between brain areas that take place at zero lag [26], and will be missed by both PLI and wPLI. Not surprisingly, recent results [21] [27] show that there is no such thing as a perfect PS index, and indeed PLV can be sometimes better than either PLI/wPLI in uncovering differences between groups and / or conditions. As a result, we included the three indices in the computational platform.

3) *Estimation of the significance of the index*

Finally, as an additional step, it is always convenient to estimate whether a given value of any of the PS indices described above is significantly different from zero, i.e., it is the result of true

FC between the data or just a spuriously high value due to, e.g., shortness of the data. For this purpose, different types of surrogate data are used ([28], [29]), yet the construction of a good set of such surrogates and the subsequent estimation of any of the PS indices from this set is unfeasible if we are to keep the computational time within a certain bound. There is, however, an alternative that can be applied in the case of the PLV, and which we include also in the corresponding library. It is based on the fact that the PLV is also the mean resultant length of the circular distribution of the relative phase [30]. For this measure, there exist well-stablished tests of uniformity of the distribution, which allow estimating the probability (p-value) that, for a given length of the relative phase series, the corresponding PLV value has been obtained by chance. In the corresponding function, we use the approximation due to Wilkie [31] of the Rayleigh test, whereby the probability is of PLV being greater of a certain value *K* for *n* samples is estimated as:

$$\Pr(nPLV^2 > K) = \exp\left\{\left[1 + 4n + 4(n^2 - nK)\right]^{1/2} - (1 + 2n)\right\} \qquad (9)$$

Although this estimation is based in the assumption that the underlying distribution of the cyclic relative phase is a von Mieses distribution (i.e., that consecutive values of (5) are approximately independent), which is not always fulfilled, it has the advantage of providing a continuous, parametric estimation of the p-value associated to each PLV without the need of constructing the surrogates. This p-values can be latter corrected for multiple comparisons using, e.g., False Discovery Rate [32], thereby providing a good estimation of the significance of each PLV.

4.2. GS indices

As commented in the Introduction, at the beginning of the eighties of the 20th century, the Dutch mathematician F. Takens proved a theorem [15] whereby, under general conditions, it is possible to reconstruct the state space of a complex dynamical system (even nonlinear systems in chaotic regime) using the consecutive values of one of its time series. Indeed, he demonstrated that, given the time series x(k), the delayed vectors defined as:

$$X_i = (x(i), x(i+\tau), \ldots, x(i+(m-1)\tau) \qquad (10)$$

are equivalent to the original state vectors. In (10), *m* is the so-called embedding dimension, which has to be at least equal to the dimension of the original system, and τ is the delay time, which has to ensure that two consecutive components of the vector are (almost) independent. Usually, m is estimated using the heuristic approach termed false nearest neighbours, whereas τ can be estimated using the autocorrelation or the auto mutual information function of the data [33] .

In the case of FC studies, this idea allows for a sophisticated assessment of the degree of statistical dependence between two EEG channels, x and y. For this purpose, delayed state vectors X_i and Y_i are first reconstructed from *x(t)* and *y(t)*, as in (10). Then, let $a_{i,j}$ (respectively, $b_{i,j}$) be the time indices of the *k* nearest neighbours of X_i (resp. Y_i). The existence of FC between both EEGs entails that vectors close in the state space of *X*, are also close in *Y*, which can be measured using different bivariate indices [3], [17]. We have included several of them, which we describe henceforth, in the computational platform.

1) Similarity index S

It is the earliest developed index of GS from two time series [34]. It is defined as:

$$S^{(k)}(X \mid Y) = \frac{1}{N}\sum_{i=1}^{N}\frac{R_i^{(k)}(X)}{R_i^{(k)}(X \mid Y)} \qquad (11)$$

where $R_i^{(k)}(X)$ is the average Euclidean distance between the X_i and its k nearest neighbours, with time indices $a_{i,j}$, and $R_i^{(k)}(X \mid Y)$ is the same but calculated taking the indices of the nearest neighbours of Y_i, $b_{i,j}$. The existence of GS between x and y produces that these k so-called *Y-conditioned* neighbours of the reconstructed vectors of X are closer to them than should be expected by chance, thus the ratio in (11) and the index itself are close to 1 (equal to 1 for identical signals). On the other hand, if there is no GS, the Y-conditioned neighbours are equivalent to vectors randomly chosen all over the attractor, and the index is close (but not equal) to 0. The corresponding version in the reconstructed state space of Y, $S^{(k)}(\mathrm{Y} \mid \mathrm{X})$, can be calculated analogously.

2) H index

The similarity index above is the simplest implementation of a bivariate GS index relying on the comparison between nearest and conditioned neighbours. Yet it has been shown [12], [35], [36] that it is not the best choice to get information about the directionality of the interaction. Besides, its value for completely independent signals is not zero but depends on the average size of the reconstructed attractor, which in turn changes with the complexity of each signal as well as with the number of available data points. Instead, a variant of this index, termed H, has been proposed, which is defined as:

$$H^{(k)}(X \mid Y) = \frac{1}{N}\sum_{i=1}^{N}\log\left(\frac{R_i(X)}{R_i^{(k)}(X \mid Y)}\right) \qquad (12)$$

where $R_i(X)$ is the average distance between X_i and any other reconstructed vector in X (the so-called radius of the attractor). Clearly, H equals 0 for independent signals, no matter the size of the attractor, yet it is not normalized (i.e., its upper bound does depend on the individual signals).

3) M index

A more appropriate way of normalizing the distances was defined by Andrzejak and colleagues [37]:

$$M^{(k)}(X \mid Y) = \frac{1}{N}\sum_{i=1}^{N}\frac{R_i(X) - R_i^{(k)}(X \mid Y)}{R_i(X) - R_i^{(k)}(X)} \qquad (13)$$

Now, for independent signals, the numerator (and therefore the index) equals zero, whereas for identical signals, the ratio equals 1. Yet the difference between (13) and the analogous expression in the state space of *Y* is sometimes misleading to draw conclusions on the directionality of the interaction.

4) L index

Recently, Chicharro and Andrzejak [16] proposed an improved version of all the indices above, which includes a first step in which the distances between vectors is the reconstructed state spaces are first transformed (rescaled) using their ranks. Thus, the index is defined as:

$$M^{(k)}(X \mid Y) = \frac{1}{N}\sum_{i=1}^{N}\frac{G_i(X) - G_i^{(k)}(X \mid Y)}{G_i(X) - G_i^{(k)}(X)} \qquad (14)$$

Here, $G_i^{(k)}(X) = (k+1)/2$ (resp. $G_i(X)=N/2$) is the average rank distances between any X_i and its *k* nearest neighbours (resp. the (N-1) remain vectors). The normalization procedure, which entails a first step consisting in the sorting of all the distances, ensures that these two distances are constant for all *i* and every signal, as they only depend on *N* and *k.* The only remaining distance, which is the one affected by the degree of FC between the signals, is $G_i^{(k)}(X \mid Y)$.

This index allows a reliable estimation of the directionality of the interaction, and, as in the case of (13), is also normalized between 0 and 1 for independent (resp. identical) signals. Yet, it is the most computationally intensive of all the GS indices, as it requires the sorting of the distances between each reference vector and all the remaining ones.

5) Synchronization likelihood (SL)

Apart from the indices described above, which are all included in the libraries of the platform, there is another one, which is often used in the context of brain FC from EEG data: this is the so-called *synchronization likelihood (SL).* [38], [39]. This index sacrifices the ability to provide information about directionality (as is the case, e.g., of the L index) in return for greater robustness against the features of the individual signals (although, as we have seen, the L index also shares this robustness). SL, which is described in detail in the original publication [38], has been, however, recently implemented in an optimal, computational very efficient way, which already uses portable C and OpenMP as well as CUDA to allow its estimation in heterogeneous multicore CPU/GPU architectures [40]. After testing the performance of this implementation, we verified that it is already suitable as a library to be used for the platform (it is open source code), yet as it is third-party software, we did not include it as a part of this deliverable.

4.3. Mutual information

Mutual information (MI) is, from a theoretical point of view, the best option to determine the degree of FC between two signals in the most general way. In fact, whereas the calculation of the PS and GS indices require previous transformation of the signal as pre-processing steps, in the case of MI it is only necessary to estimate the individual (marginal) and the joint entropies of the data. As we will see, however, this apparently very simple concept is not without its complications.

Formally, MI for a discrete pair of random variables (signals) X and Y is defined as:

$$MI(X,Y) = \sum_{y_i \in Y} \sum_{x_i \in X} p(x,y) \log\left(\frac{p(x,y)}{p(x)p(y)} \right) \quad (15)$$

where p(x,y), p(x) and p(y) are the joint probability function and the marginal ones, respectively. For independent processes, the joint probability factorizes as the product of both marginal probabilities, the logarithm and therefore also the MI equals zero. If, on the other hand, the processes are not independent, but there is *any kind of statistical dependence between them*, the joint probability is higher than the product of the marginal ones and MI is greater than zero. Alternatively, MI can be also defined as a combination of entropies, namely:

$$MI(X,Y) = H(X) + H(Y) - H(X,Y) \quad (16)$$

Either way, the practical estimation of *MI* and its application in the context of FC / network patterns estimation present two problems

1) Estimation of the probability from data

This is possibly the main difficulty. The naïve estimation of the probabilities, which is based on the binning of the ranges of *X* and *Y,* and implicitly assumes that the relative frequency of a given value estimated from this binning equals its probability. Although it is the fastest implementation [5], as it is based on the individual and the joint histogram, it is also known to produced biased estimations of *MI* [9], [10], because such equality is only valid in the limit of infinite data samples (the well-known Law of Large Numbers). Even sophisticated binning strategies, such as using bins of unequal size to maximize the individual entropies, do not eliminate this bias in practical applications.

Therefore, more elaborated estimations of the probabilities are called for, if we are to eliminate such bias. After extensive searching in the literature and testing of different options, we decided to follow the implementation of Kraskov and colleagues [10], which, according to the latest theoretical as well as practical results, has demonstrated to be the most adequate one for practical applications, showing statistical properties than excel those of more modern and sophisticated estimations of correlation such as the *maximal information coefficient* (MIC). It is based in an estimation of MI that makes use of the *k*-nearest neighbours statistics for the estimation of the entropies (see [10] for details). This approach has proven very robust and appropriate for time series analysis.

2) Normalization of MI

As commented above, one problem with MI is that, whereas it is (theoretically at least) zero for completely independent data, its value is not bounded in the case such independence does not hold. Instead, its upper limit for completely dependent signals pretty much depend on the individual entropies of each data (that is exactly the problem that the MIC was supposed to solve). This implies that a value of MI of, say, 0.3 for two signals may in fact result from a higher correlation than a value of 0.25 from another two (possibly more complex) ones. This issue may

become a serious one if we are to compare the degree of FC in two different situations / populations based solely on MI.

Different normalization procedures are possible. Among them, two possibilities are common, which make use of the individual entropies as normalizing factors. They are the *symmetric uncertainty* [41], defined as:

$$U(X,Y) = 2\frac{MI(X,Y)}{H(X)+H(Y)} \qquad (17)$$

and the total correlation

$$\frac{MI(X,Y)}{\min[\mathrm{H(X)},\mathrm{H(Y)}]} \qquad (18)$$

which are both normalized to 1.

In order to allow the calculation of any of these two normalized versions, the corresponding library in the platform estimates not only (16) but also the individual entropies of each of the signals considered.

Finally, it is worth noting that we also tested the feasibility of including in the platform the abovementioned MIC index [7], [42], which is normalized per definition and allegedly presents also very good statistical properties as an associated measure. Yet, on the one hand, the latest results seem to suggest that MIC is not better than Kraskov's MI implementation [7]. And, on the other hand, even the fastest implementations of MIC[43] are orders of magnitude slower than the MI implementation we have obtained, which render this former index as unsuitable for (close to) real-time estimations of FC patterns.

5. Practical implementation and simulations

5.1. PS indices

As it turns out, PS indices as such are the easiest to implement in practice. Yet the need for different pre-processing steps (filtering and phase extraction) complicates matters. We will first briefly review the currently available versions (open source), which we reviewed previously to the implementation of ours, and will then show the results of the speed-up we obtained with our implementations in the computer simulations performed using the HW platform described in section 3.1.

5.1.1 Currently available versions

To the best of our knowledge there are three different open source implementations of the PS indices.

a) DAMOCO.

Rosenblum and colleagues have developed a MATLAB® toolbox for Data Analysis with Models of Coupled Oscillators (DAMOCO, http://www.stat.physik.uni-potsdam.de/~mros/damoco2.html). This toolbox includes the implementation of PLV, filtering and Hilbert transform. They are all in plain MATLAB code and not even optimized for multichannel data. PLI and wPLI are not included, and not estimation of the significance of the index is available either.

b) Field Trip

The popular toolbox for EEG/MEG analysis (http://www.fieldtriptoolbox.org/) includes filtering and phase estimation scripts, as well as estimators of the three PS indices (but not of their significance). Again, these implementations are not optimized, and more importantly, the calculation requires that the data are in FieldTrip data format, which obviously further constrains its usefulness in third-part applications.

c) EEGLab

As in the case of FieldTrip, this toolbox for EEG analysis (http://sccn.ucsd.edu/eeglab/) includes filtering and phase estimations scripts, as well as estimators of PLV and PLI (with no estimation of their significance). But again, the same issues as in the case of FieldTrip, including those regarding the need to work with data in EEGLab format.

Thus, we used the implementations above mainly to check and validate our results.

5.1.2 Results of the simulations

Zero-Phase Filtering

The first one of the software libraries from this part of the platform consists of an optimized implementation of FIR zero-phase band-pass filter, with the following input / output parameters.

```
%% zero phase filtering
% USE: trial_filtfilt(filter,data,mode)
%
% filter= row vector with filter denominator coefficients (b)
% data = column vector with data to be filtered
% mode = mode for FFT scheduling
%        mode = 0 -> execute fastest but suboptimal.
%        mode = 1 -> execute fast but in a suboptimal algorithm.
```

```
%     mode = 2 -> execute slower the first time, but consider possible faster
algorithms.
%       mode = 3 -> execute slowest the first time, but consider the fastest
algorithm.
```

The order of the filter is estimated adaptively from the number of samples in the times series as one fifth of the data. Besides, previous to the filtering, the data are mirror padded at both sides, to reduce edge effects. Filtering is then carried out efficiently in the frequency domain after applying the FFT to both the data and the filter itself. The FFT mode, which is introduced as a parameter, allows deciding whether it is necessary to carry out an initial (i.e., at the beginning of each MATLAB session) search for the optimal FFT algorithms (which is especially useful should the length of the data be not a power of 2). The function also estimates the total number of CPU cores (threads) available before executing the main calculation routine to optimize performance.

The call of this function is as follows:

y_filtered=trial_filtfilt(b,data,mode);

where the filtered data is included in the output variable *y_filtered*

Figure 3 shows the results of the simulation of the total time spent in the filtering process as a function of the number of channels[5] / data samples. Besides, Figure 4 shows a comparison of the time spent by our implementation and that spent by the (already optimized) implementation of the function filtfilt from the Digital Signal Processing toolbox of MATLAB. Note the speed-up (more than one order or magnitude) for a typical data segment of 64 sensors/ 10^3 points (2 seconds sampled at 0.5 kHz), for a total computational time of less than 10^{-3} s.

[5] Henceforth we will use the terms channel and sensor interchangeably to denote number of simultaneous variables to study FC. In practice, they may be either scalp EEG channels, MEG sensors or reconstructed neural sources/brain areas.

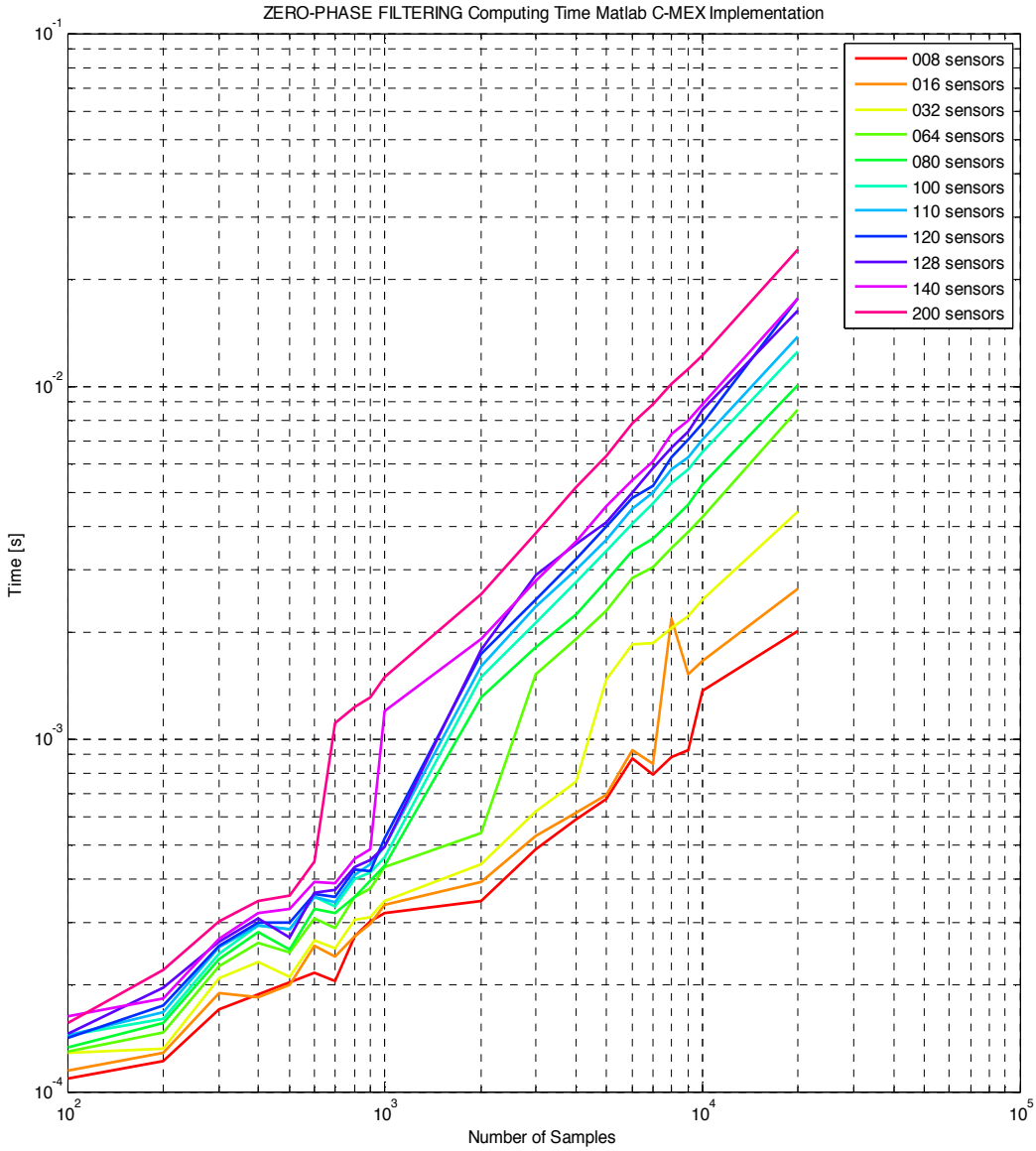

Figure 3: Calculation time of our filtering library for the HW platform described in section 3.1

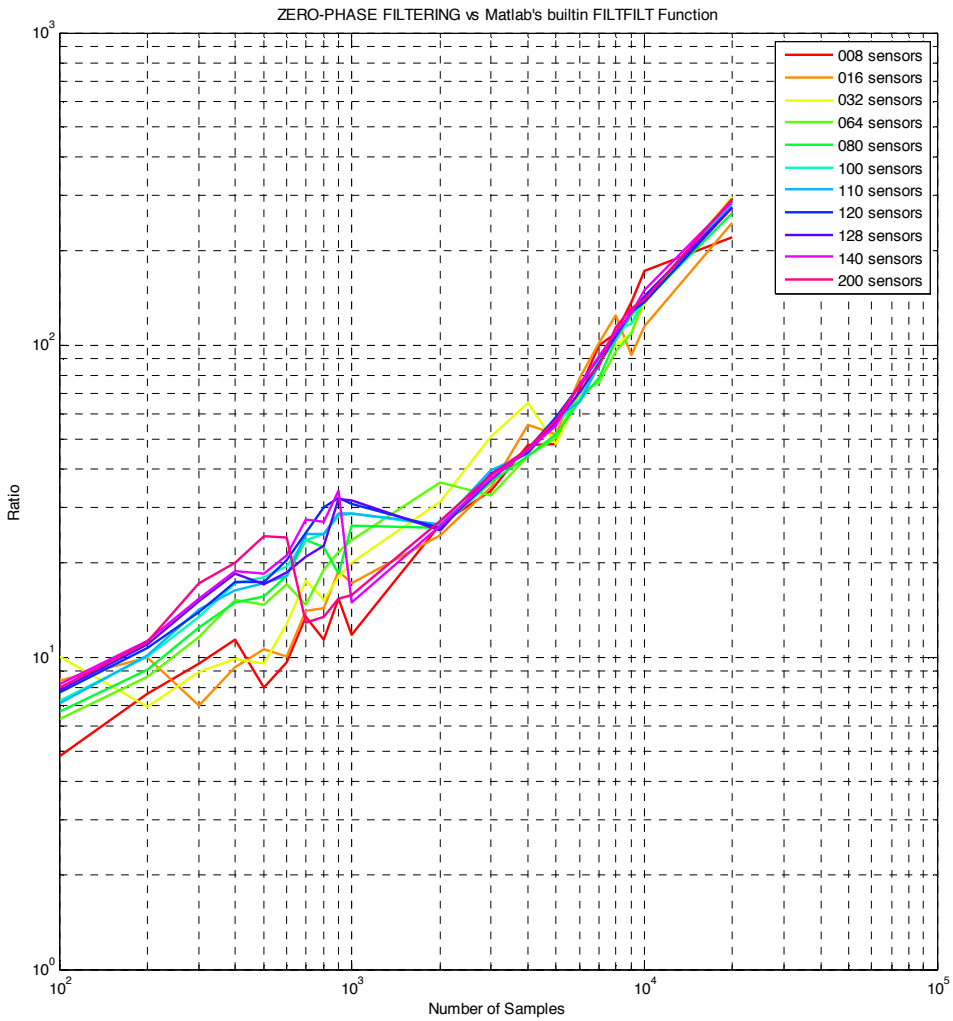

Figure 4: Speed-up of our filtering implementation, as compared to MATLAB function filtfilt in the DSP, for different values of data length and number of channels

Phase-based Functional Connectivity indices

PLV. PLI. wPLI are all computed (along with the estimated significance of the PLV) within the MEX function directly invoked from the MATLAB environment.

```
%[plv,pval_plv,pli,wpli]=function trial_pl(data,samples_elim,mode)
% data = eeg data (sensors by columns).
% samples_to_discard = samples to discard at the beginning and samples to
%                      discard at the end of the phase signals.
%        mode = 0 -> execute fastest but suboptimal.
```

```
%        mode = 1 -> execute fast but in a suboptimal algorithm.
%         mode = 2 -> execute slower the first time, but consider possible
faster algorithms.
%       mode = 3 -> execute slowest the first time, but consider the fastest
algorithm.
```

An additional input parameter here is the number of data samples to discard, to prevent edge effects in the estimation of the HT, if any, to affect the calculation of the indices. The function uses an FFT-based algorithm to estimate the HT, after mirror padding the data, so that discarding of data should be used only if serious edge effect are suspected.

The call of the function is:

[plv,pval_plv,pli,wpli]=tial_pl(data,samples_elim,mode)

For the calculation of the indices, we considered different versions to compare their performance

- Vectorized MATLAB® code using the BSX function
- The former one plus the Parallel Computing Toolbox (PCT) to exploit all available CPUs[6]
- GPU implementation using the GPUarray function included in the PCT toolbox
- MATLAB wrapper of a C-MEX compiled version (single CPU)
- MATLAB wrapper of a C-MEX compiled version using OpenMP (multiple CPU)

The following figures show the results of the simulations for different combinations of data length and number of sensor/channels in each case. As can be seen, the multicore MEX version compiled from the portable C function using OpenMP is the fastest one, so that calculation are accelerated by a factor of x20 as compared to the fastest pure MATLAB implementation, for the same example as above (i.e., 64 channels and 1000 data samples). It is noteworthy that, as shown in Figure 7, calculation time on the GPU device is much slower than in the CPU (even the pure MATLAB® versions) and independent of the number of data samples, which is due to the fact that the main bottleneck in GPU-based calculations is the time required to transfer the data from system's to GPU's memory, which is limited by the bandwidth of the PCI bus. Also, by comparing Fig. 5 with Fig. 6, another interesting result is apparent, namely that the use of PCT to increase the number of workers produces actually an increase in computational time for the pure MATLAB vectorised code, due to the necessary data transfer from/to the workers even with sliced variables.

As shown in Figure 10, the whole procedure of filtering, phase calculation and PS indices estimation (including significance of the PLV) for 64 channels and 10^3 samples for the HW

[6] From MATLAB versión 2013 on, the number of local workers in a MATLAB pool to be used with the PCT is no longer limited.

platform used is less than $5x10^{-3}$ s., a speed up of almost two orders of magnitude compared to the fastest MATLAB implementation.

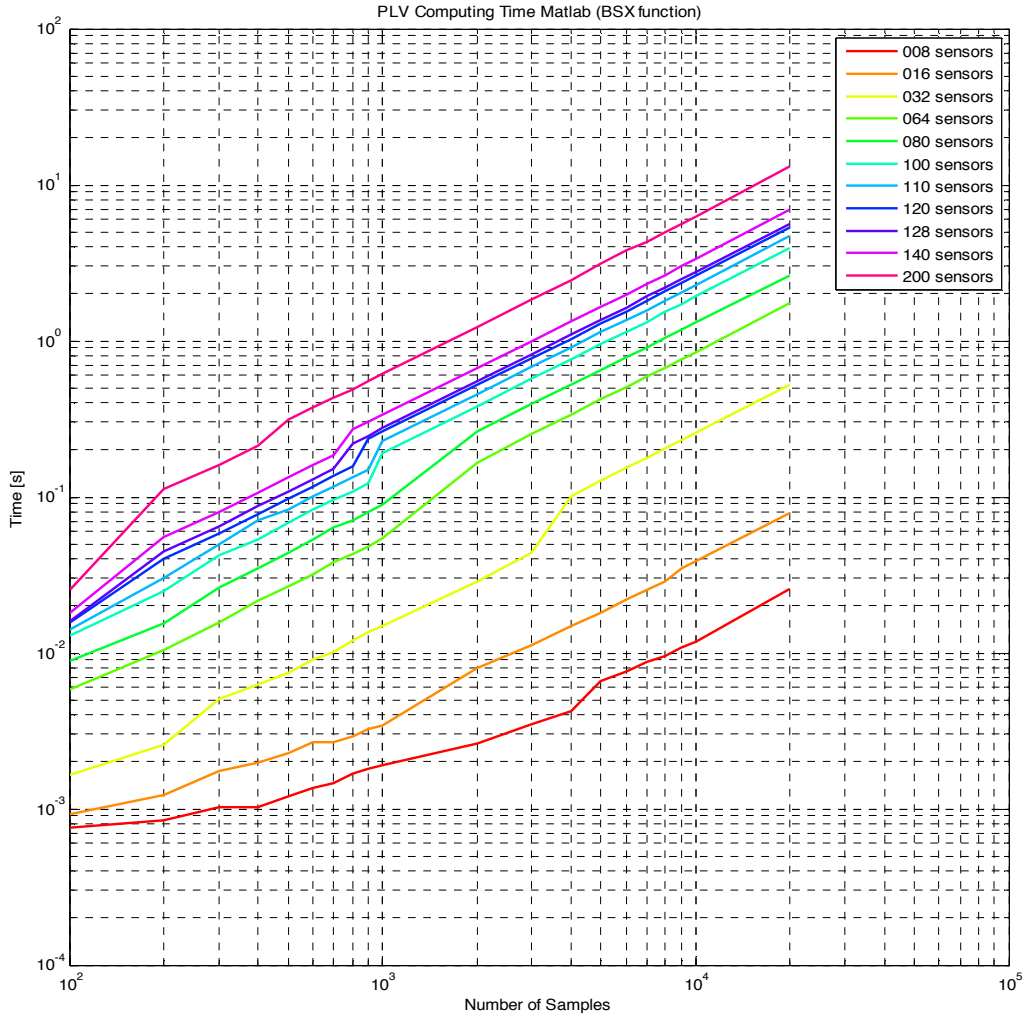

Figure 5: Calculation time of the "pure" MATLAB vectorised implementation of the *trial_pl* library as a function of the number of channel/sensors and number of samples.

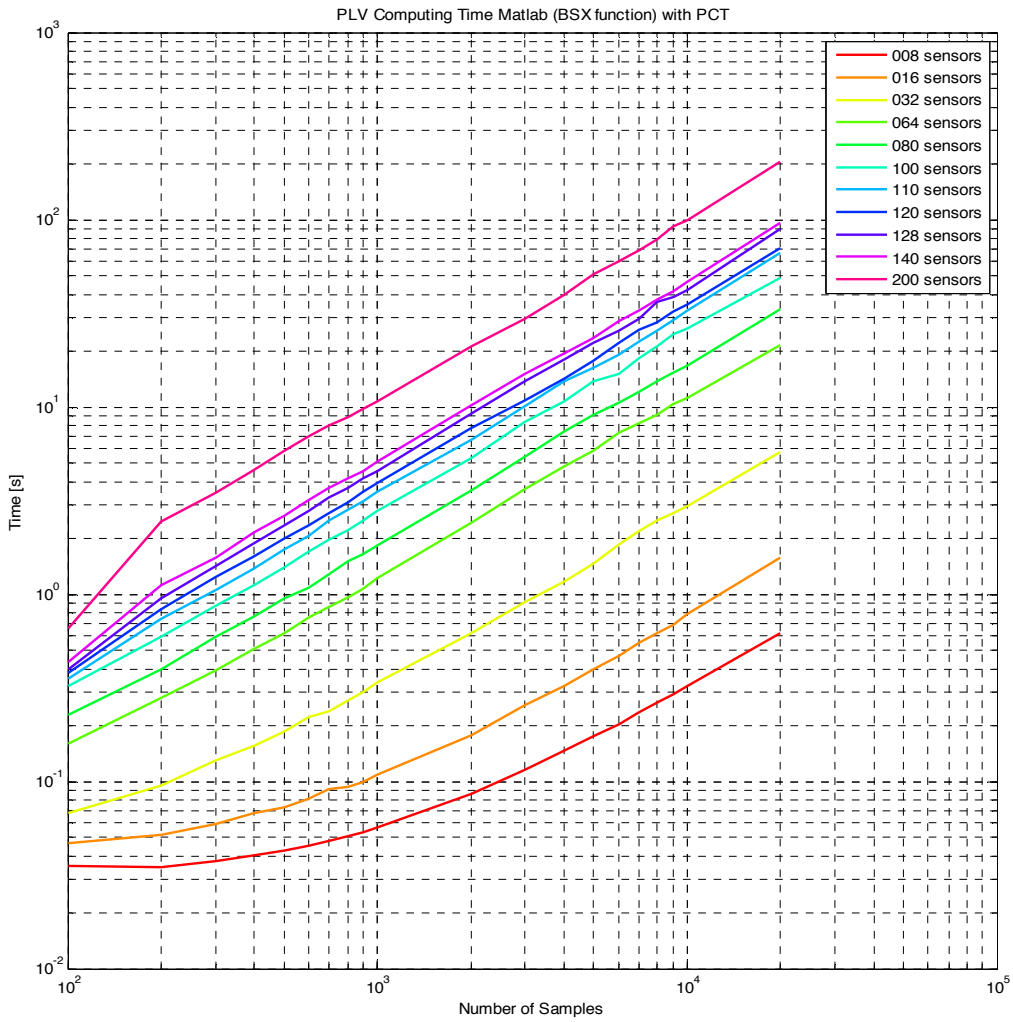

Figure 6: Calculation time of the MATLAB PCT implementation to calculate the PS indices, which makes use of all available CPUs as MATLAB workers, as a function of the number of channel/sensors and number of samples. Note that, as compared to the version without PCT, typical times are actually higher here, as commented in the text (see also Table 1).

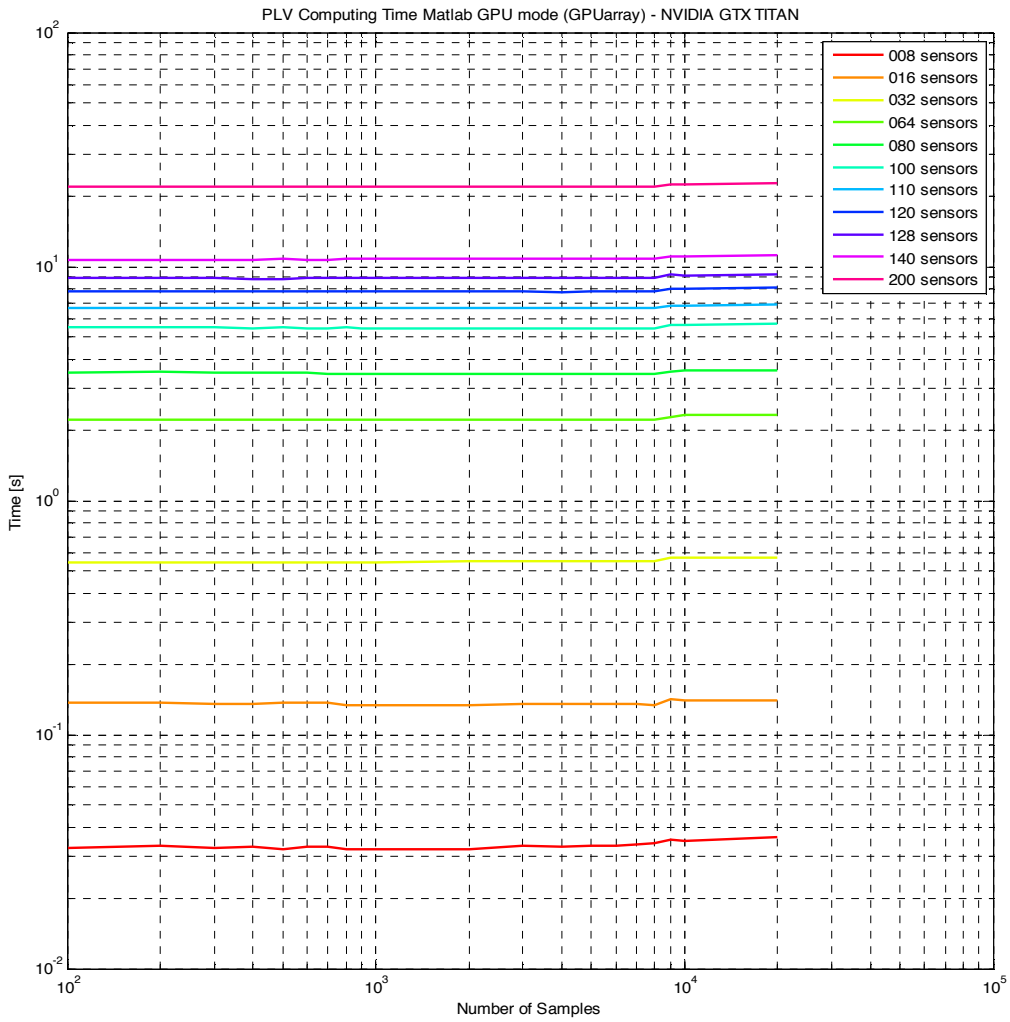

Figure 7: Calculation time, as a function of the number of channel/sensors and number of samples, of the MATLAB implementation to calculate the PS indices using the GPUarray function included in the PCT. It allows estimating the indices on the GPU devices instead of the CPU using pure MATLAB code. Note that the times are almost independent of the number of samples, but much higher than those for both the MATLAB vectorised and the C-MEX versions, for typical numbers of data samples (i.e., up to than 10^4).

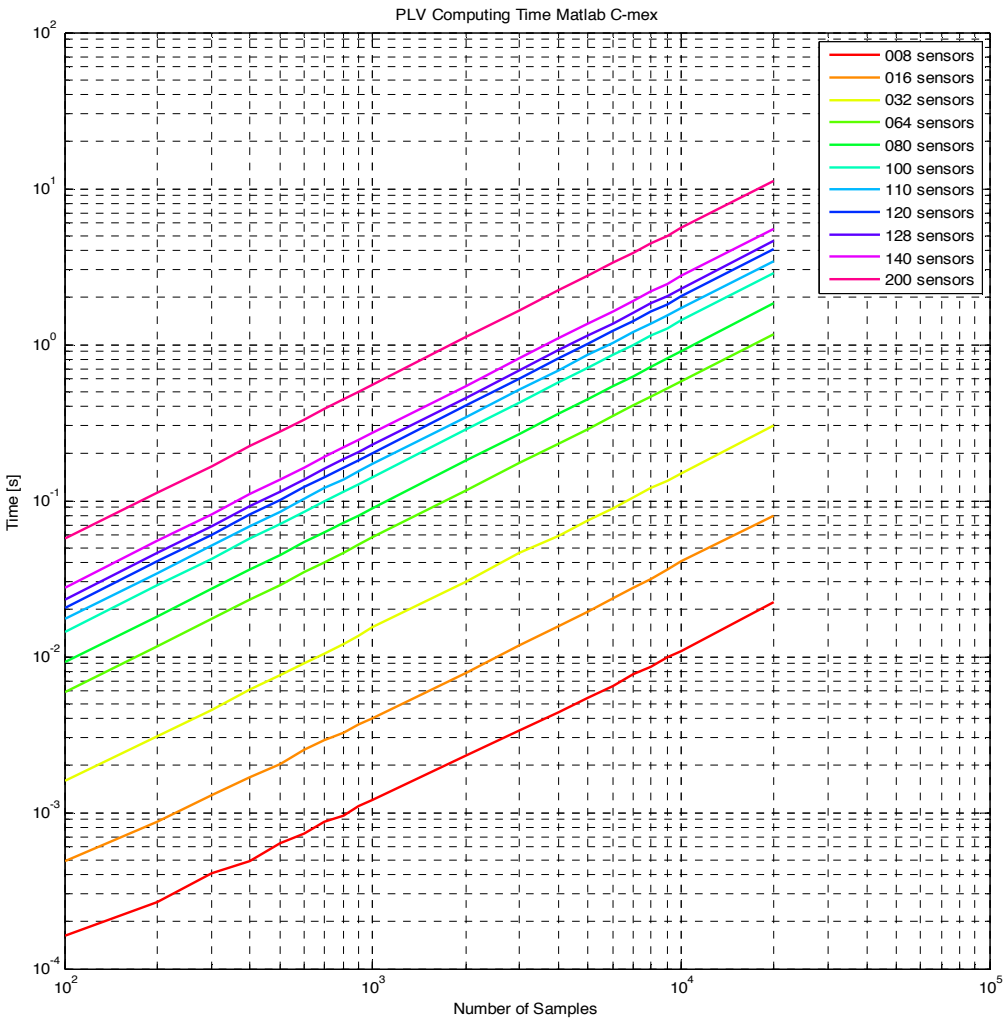

Figure 8: Calculation time, as a function of the number of channel/sensors and number of samples, of the implementation using a MEX compiled version of the portable C code on a single CPU.

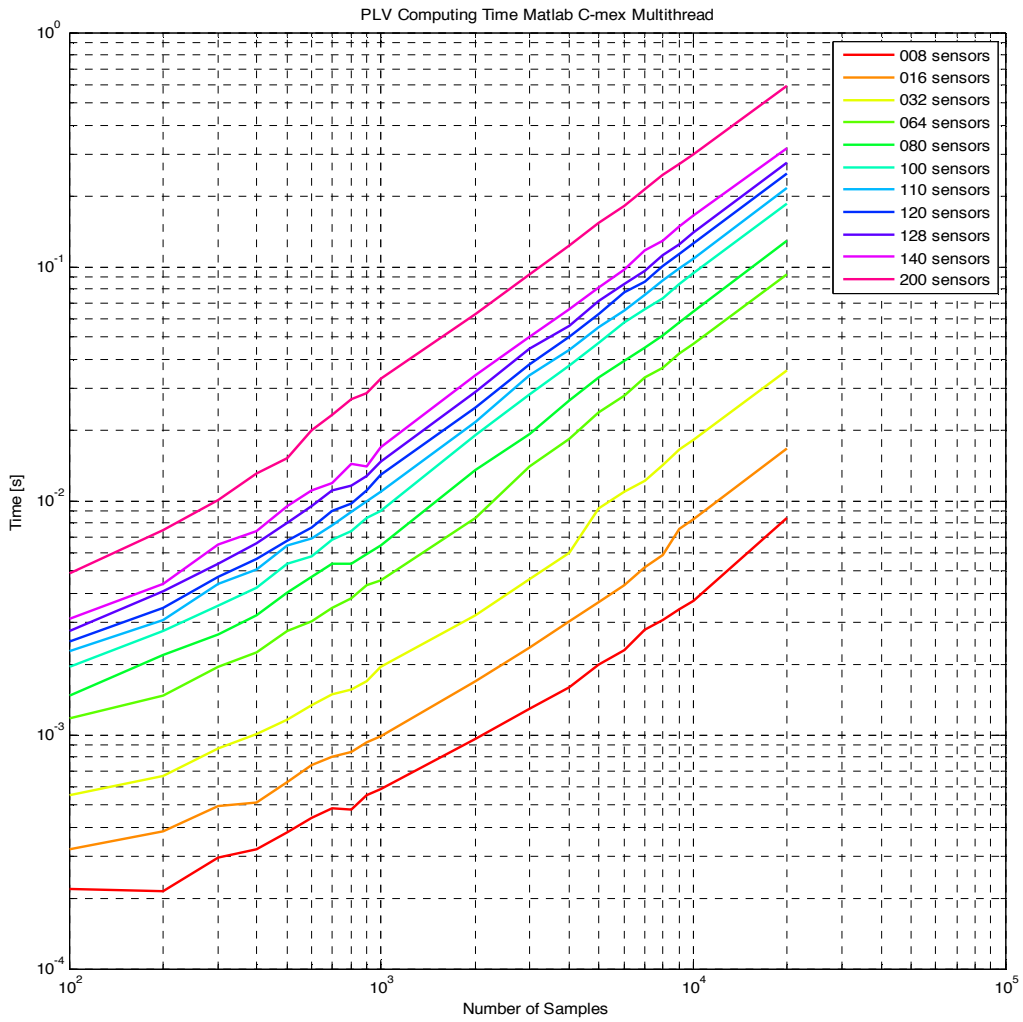

Figure 9: Calculation time, as a function of the number of channel/sensors and number of samples, of the implementation using an OpenMP MEX compiled version of the portable C code using all the CPUs available.

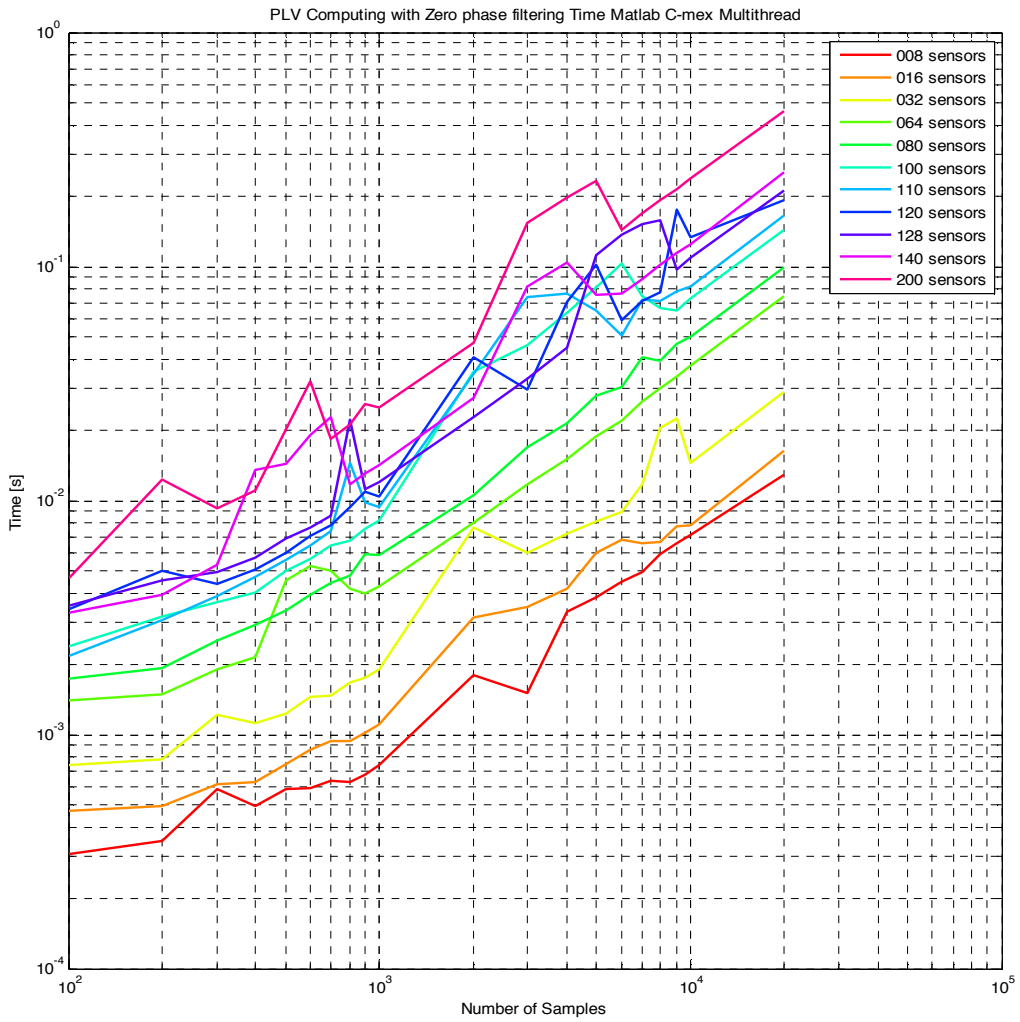

Figure 10: Total computational time of the whole process of PS indices calculation using the multicore C implementation for filtering, indices calculation and significance estimation for PLV.

Table 1 provides also concrete figures of the execution times associated to PS indices calculation with the different implementations (64 electrodes /2x10³ channels). As it turns out, the vectorised MATLAB implementation is faster than the one using normal code combined with the PCT. Also, it is quite clear that the GPU implementation is very inefficient. Instead, the multithread implementation allows for a speed-up in the calculation of the indices alone of x20.

Implementation	Execution Time [s]	Speed up
MATLAB (BSX)	0.178	-
MATLAB PCT	2.49	0.07
MATLAB GPU	3.21	0.0039
C-MEX single thread	0.12	x1.4
C-MEX multithread	0.0087	**x20**

Table 1: Execution times and corresponding speed-up (as compared to the fastest MATLAB implementation, first row) of the different tested implementations to calculate the PS indices (64 channels / $2x10^3$ samples) with the HW platform described in section 3 (in bold, speed-up of the fastest version)

5.2. GS indices S,H,M and L

5.2.1 Currently available versions

As far as we are aware, there are only two open-source implementations (one of our own) of the GS indices considered. One is the original implementation included supplementary material in the paper of Chicarro & Andrzejak [16]. The second one is our implementation as included in the HERMES toolbox [17]. None of them are optimized for performance and neither take advantage of the existence of multicore / GPU devices, if any, in the HW architecture.

5.2.2 Results of the simulations

Generalized Synchronization

The function implementing the calculation of the GS indices, and its corresponding input/output parameters, is defined below.

```
%% Estimation of GS indices
% [S,H,M,L]=function trial_shml(data_1,emb_dim,tau,k,w,states_eff_step)
% data = eeg data (sensors by columns).
% emb_dim = embedding dimension
% tau = time lag for embedding
% k = number of neighbours to consider
% w = window correction for neighbour finding
% states_eff_step = state-space down sampling to consider when calculating
distances
```

Figures 11 to 13 show the results of the simulations for different combination of number of channels/samples. GPU implementation is not shown, as it is again inefficient. Thus, only C-MEX (single and multicore implementations) as well as pure MATLAB code one are shown. The results of simulations shown correspond to the following set of parameters: emb_dim=1; tau=1; k=6; num_threads=16; states_eff_step=1; w=1. Yet the results do not change significantly (especially in terms of relative speeding of the parallel C-MEX as compared to the MATLAB code) for different combinations of them.

Note that the pure MATLAB implementation, even after vectorization and using the optimized MEX function psdist2.m included in the Statistical Toolbox, is not very efficient, and the computational time for more than 8 sensors is high. By comparing Figure 13 with Figure 11, it is clear that the OpenMP C-MEX compiled version, which makes use of all available CPUs, significantly reduced this computational time. Thus, e.g, for 8 sensors / 10^3 samples, it decreases from 25 (Fig. 11) to $9x10^{-2}$ s (Fig. 13), i.e., a speed up factor of x277 (more than two orders of magnitude). For the fastest implementation, it takes about 10 seconds to estimate all the GS indices for the typical configuration considered before of 64 sensors / 10^3 samples, which is clearly much slower than the PS indices. This is to be expected, because the calculation of GS indices is computationally very expensive, as it involves the estimation and sorting of $O(N^2)$ distances in *m* dimensional spaces.

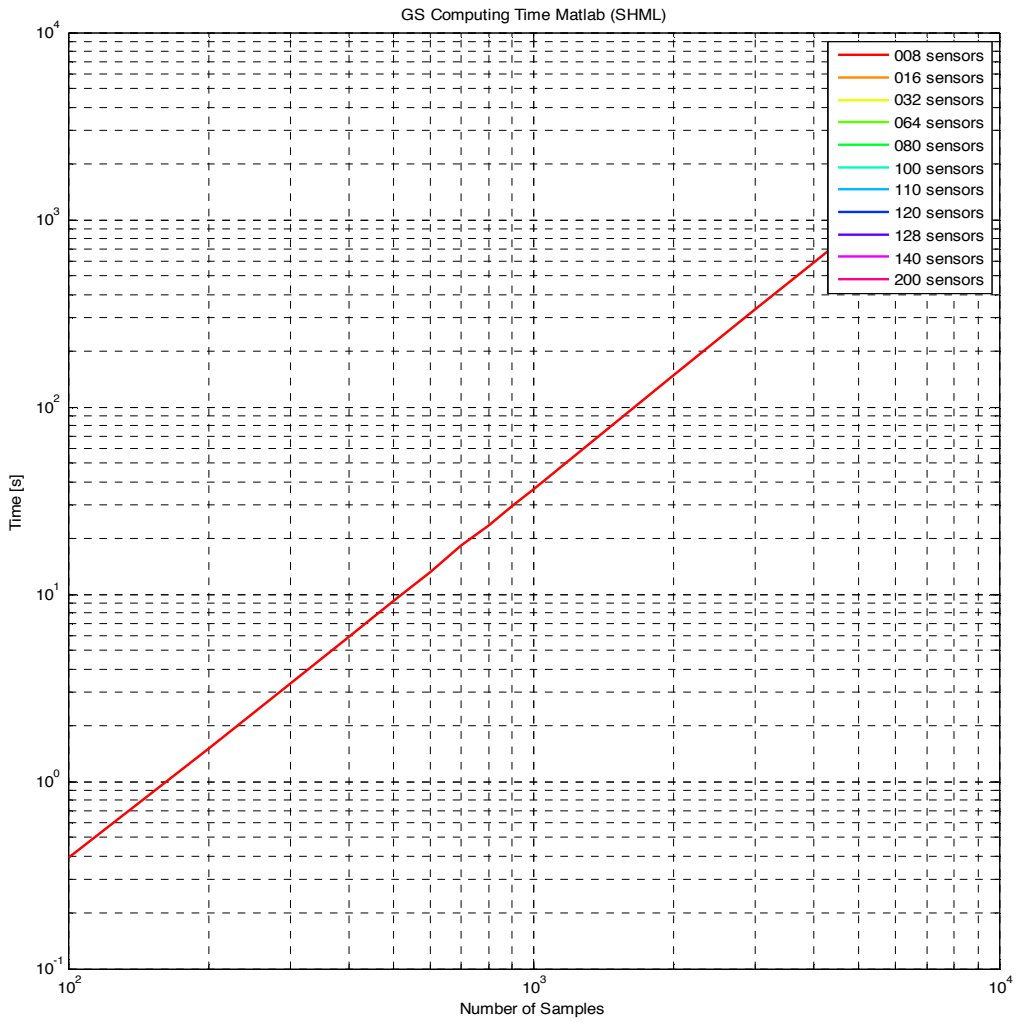

Figure 11: Total computational time of the process of GS indices using the pure MATLAB implementation with the set of parameters indicated in the text. Note that, due to the inefficiency of MATLAB, only the time for the simulation with 8 sensors is included.

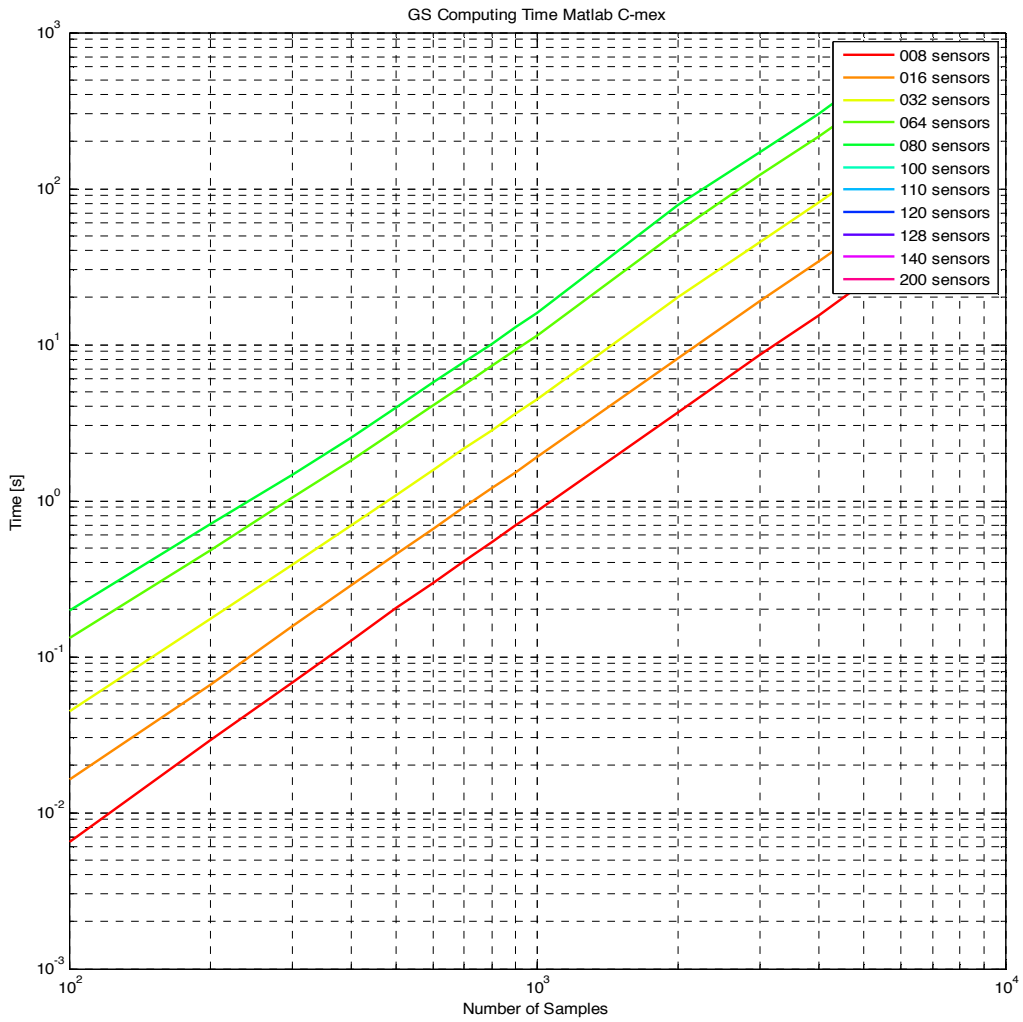

Figure 12: Total computational time of the process of GS indices using a C-MEX compiled implementation of the function on a single CPU

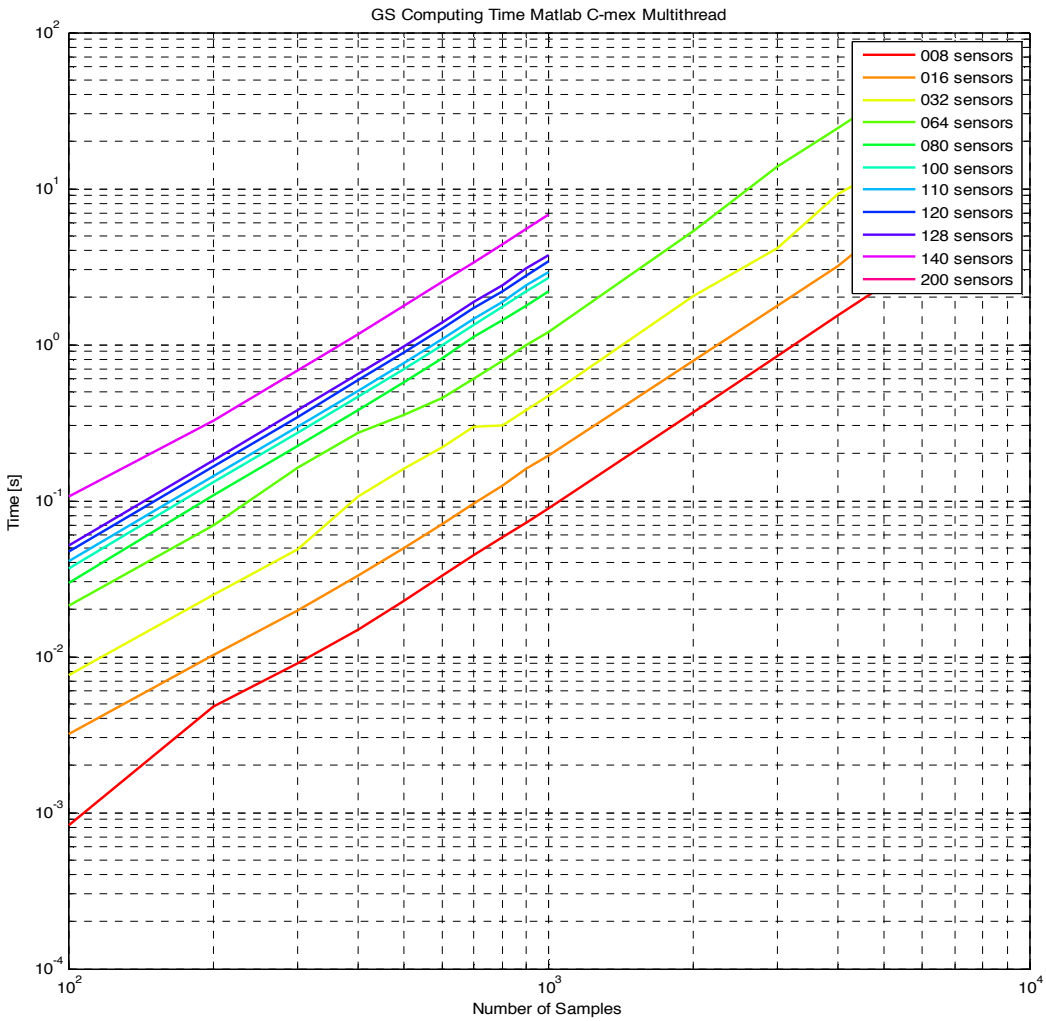

Figure 13: Total computational time of the process of GS indices using a C-MEX implementation of the function compiled using OpenMP and working on all the CPUs available

Again, table 2 provides concrete figures for the execution time, which shows the excellent speed-up obtained with the fastest implementation as compared to the currently existing MATLAB ones for a typical number of sensors and data channels.

Implementation	Execution Time [s]	Speed up
MATLAB	>1900	-
MATLAB PCT	>1900	-
C-MEX single thread	51.67	x36
C-MEX multithread	3.08	**x616**

Table 2: As in table 1 but for the GS indices

5.3. Mutual information

5.3.1 Currently available versions

There are different open source implementations of MI, most of them based on the naïve approach discussed above that involves the estimation of probabilities (marginal and joint ones) using binning strategies (for a recent example, see [5]). Yet, as commented, the approach based on k-nearest neighbours statistics as described in [10] and recently reviewed in [7] is the most suitable one in our framework. Thus, we focused our attention, as starting points, on two efficient implementations of this strategy, already in C/C++.

a) TIM

This is a cross-platform open-source C++ library for the estimation of information-theoretic measures from continuous-valued time series. It is available at:
http://www.cs.tut.fi/~timhome/tim/tim.htm

b) MILCA

This is the implementation of the authors of ref. [10], available at:

http://www.ucl.ac.uk/ion/departments/sobell/Research/RLemon/MILCA/MILCA
in C++ code.

This turned out to be the fastest and most accurate one, and use therefore used as a benchmark against which to compare our implementation.

5.3.2 Results of the simulations

Mutual Information

The function implementing the calculation of MI indices, and its corresponding input/output parameters, is defined below.

```
% Estimation of the MI using k-nearest neighbours entropies
% mi=function trial_mi(data,emb_dim,tau,k);
% data = eeg data (sensors by columns).
% emb_dim = embedding dimensions to consider
% tau = time lag to consider for embedding
% k = number of neighbours to consider
```

The results of the simulations below correspond to the input parameters emb_dim=1; tau=1; k=6; for different combination of data length and sensors. Again, as in the case of GS indices, changes in the input parameters did not significantly change the results (either quantitatively r qualitatively). Again here, only our implementations based on portable C (single or multicore ones), which are the most efficient ones, are included (GPU suffers from the same problem in regard to data transfer from / to GPU memory, see Table 3, and are therefore not included in the comparisons below).

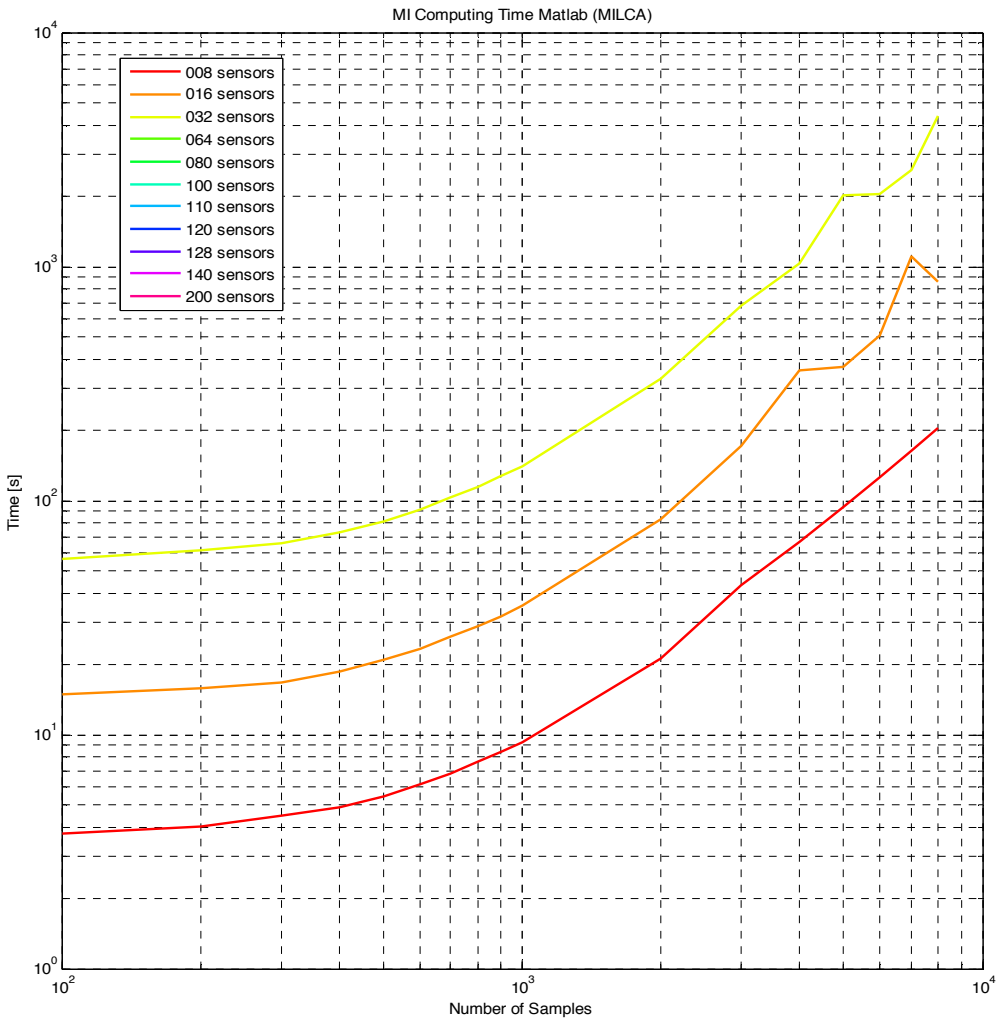

Figure 14: Computational time as a function of the number of samples / sensors for the MI estimated using the original MILCA implementation (see text for details) on a single CPU. Computational times for more than 32 sensors are very high (not shown in the plot).

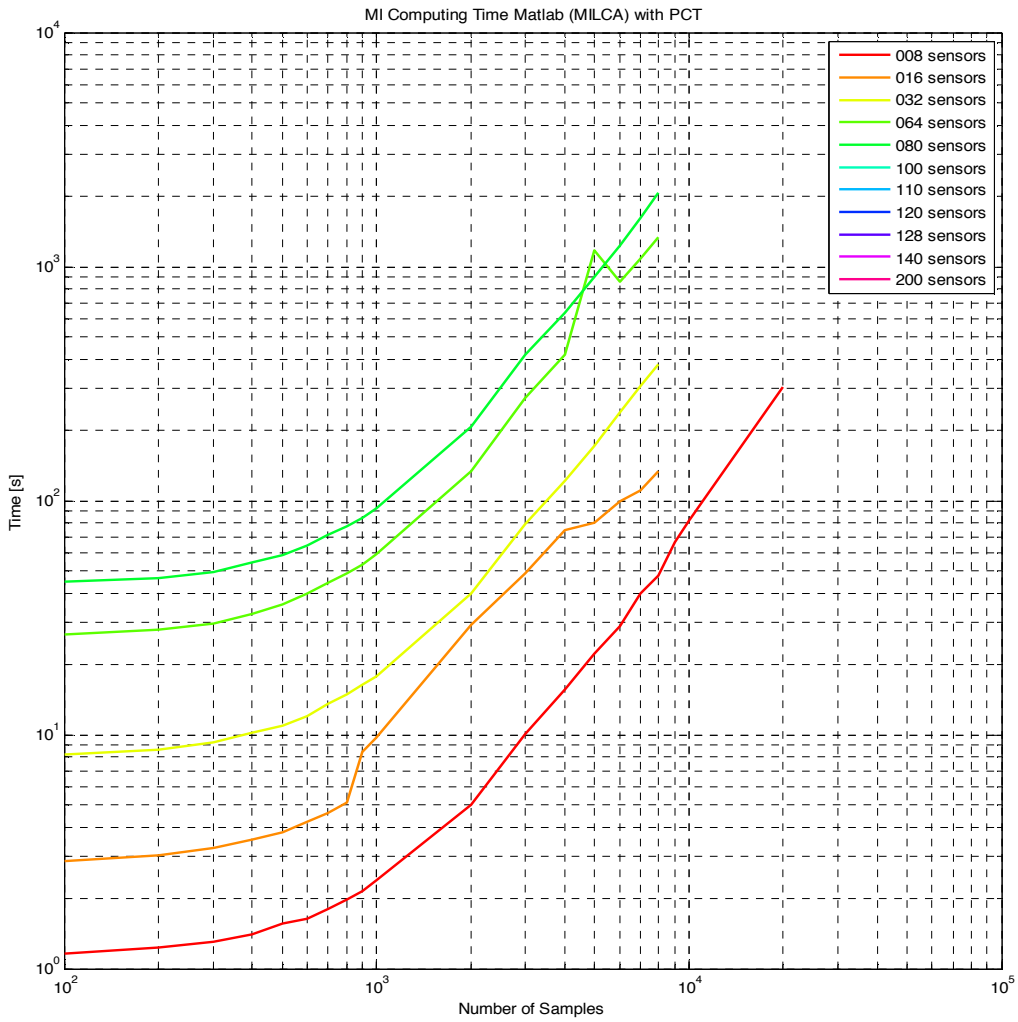

Figure 15: Same as in figure 14 but combining the original MILCA implementation with MATLAB PCT to make use of all available CPUs in the HW platform. Again, computational time for more than 80 sensors are very high (not shown in the plot)

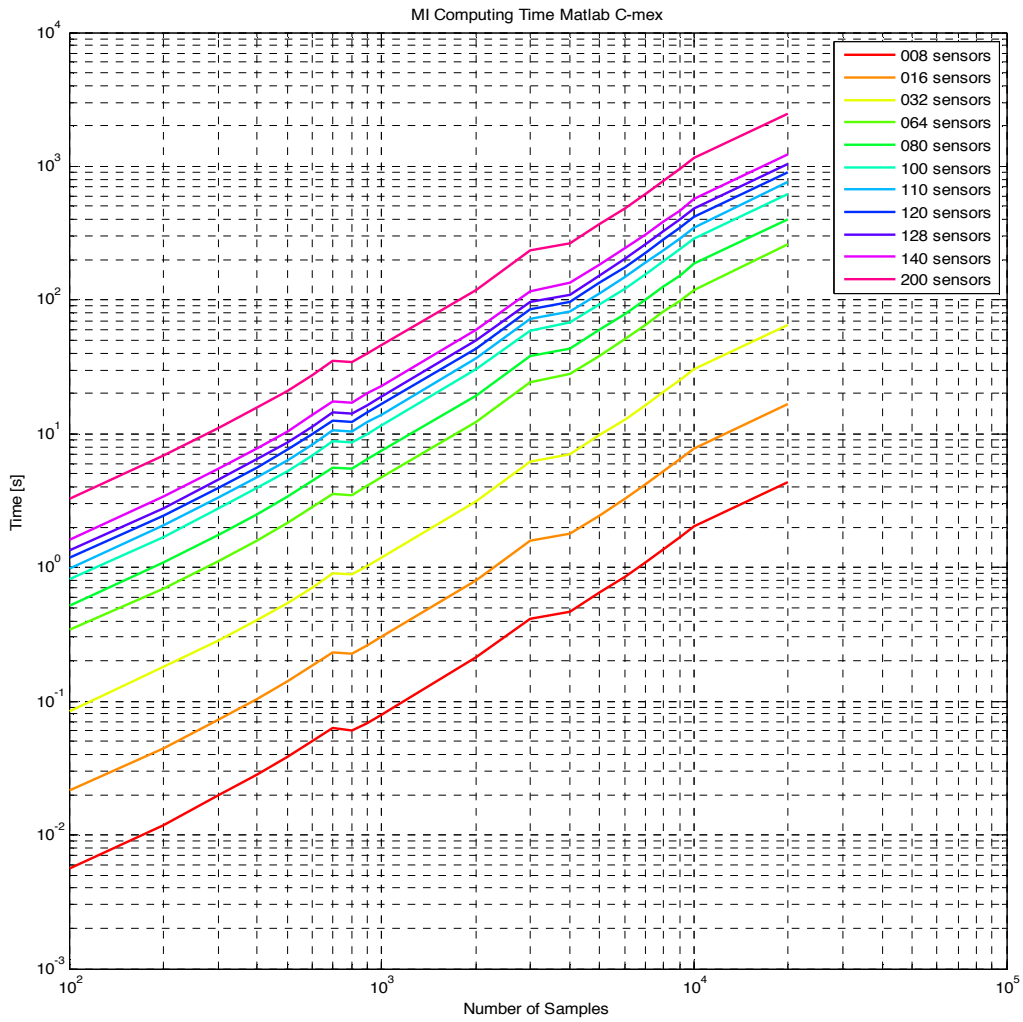

Figure 16: Computational time as a function of the number of samples / sensors for the MI estimated using our implementation of the k-nearest neighbours entropy estimation (see text for details) on a single CPU

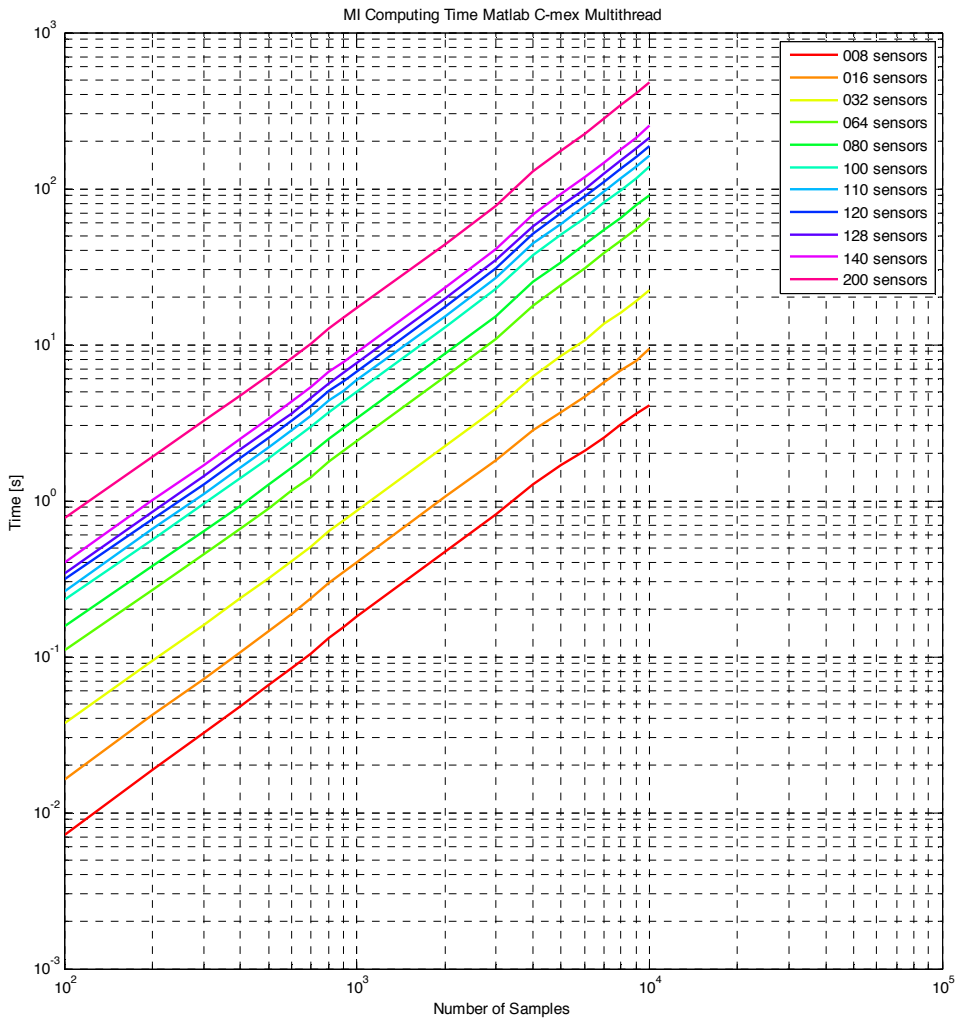

Figure 17: Same as in figure 16 but using our multithread C-MEX version of the function compiled using OpenMP

As it is apparent from the figures, the original MILCA implementation, despite being already in C++, gives rise to high computational times especially for more than 32 channels. The use of MATLAB® PCT improves the situation, but still produces computational times that preclude the use of this algorithm for applications where real time (or close to it) processing is needed. Instead, our implementation, especially in the case of the multithread version, produces a speed up of two order of magnitude. Indeed, for the same example as used in PS and GS indices (64 channels / 10^3 samples), it reduces the computational time, as compared to the MILCA PCT version, from 90 to 2 s (x45 speed up).

Implementation	Execution Time [s]	Speed up
MATLAB (MILCA)	1702	-
MATLAB (MILCA+PCT)	133	x12.7
MATLAB GPU	>1900	-
C-MEX single thread	12.36	x141
C-MEX multithread	4.41	**x385**

Table 3: As in tables 1 and 2 but for the MI implementations

6. Concluding section

6.1. Conclusion

The aim of this deliverable, included in WP3.3 of the Project, was, as remarked in the Introduction, the implementation of a computational platform to estimate network patterns of the creative brain from neurophysiological data. Given the objectives of the Project, such platform should allow the estimation of these patterns in a computationally efficient way, so that they can be analysed and used, within a recording session, to guide the stimulations protocols as well as to determine the effect, if any, of such protocols in brain activity. For this aim, currently available implementations are not suitable due to different reasons, as described in section 5. Instead, we have managed to produce a set of libraries, which perform optimally in the HW platform described in D2.3 by taking advantage of the multicore CPUs architecture combined with the speed associated to portable C implementations.

Besides, we have also checked that the GPU devices, while extremely efficient in the calculation time for a range of data length, cannot compete with the multicore CPUs implementations because of the time associated to the transfer of data from / to GPU memory. Thus, for instance, in the case of PS indices, even for the configuration with 4 GPU devices, which will likely reduce the data transfer time by a factor x2 as GPU related data transfer occurs through the PCIe standard port which has only two channels. So, time acceleration can only be increased in a factor of two. If 4 GPU devices are installed, each channel of PCIe will necessarily be shared by a pair of devices. The execution time as shown in Table 1 will be nowhere near to the ones obtained with the fastest C-based implementations.

Our libraries, which shape the intended platform, have achieved a speed-up ranging from x20-40 in the case of PS indices (which are the fastest ones) to x400-600 in the case of GS indices (the slowest ones) for a typical combination of sensors /samples. Most importantly, the execution times obtained in all cases allow for a real time[7] estimation of the network patterns, and in the case of PS indices, which are calculated extremely fast, it is even possible to obtain an estimation for the whole frequency range of an EEG/MEG signals (i.e., between 0.5 and 100 Hz) in bands of 1 Hz bandwidth in less than 1 s (i.e., one fourth of the time needed to record the data). Thus, we can say that we have achieved the goals of this WP.

6.2. Glossary

CPU	Central Processing Unit
GPU	Graphic Processing Unit
EEG	ElectroEncephaloGraphy
MEG	MagnetoEncephaloGraphy
FC	Functional Connectivity
EC	Effective Connectivity
PS	Phase Synchronization
GS	Generalized Synchronization
MI	Mutual Information
MIC	Maximal Information Coefficient
PCT	Parallel Computing Toolbox
HW	HardWare
SW	SoftWare

[7] Here, by real time estimation we mean that the time necessary to calculate the network patters is of the same order of magnitude as the time necessary to record the data. In fact, in the case of the GS indices (the most computationally demanding ones), for a typical sampling frequency of 500 Hz (0.5 kHz), the FC pattern of 4 seconds of data (2000 samples) from 64 EEG channels require 4.41 for its computation.

Bibliography

[1] K. J. Friston, "Functional and Effective Connectivity: A Review," *Brain Connect.*, vol. 1, no. 1, pp. 13–16, 2011.

[2] J. D. Medaglia, M.-E. Lynall, and D. S. Bassett, "Cognitive Network Neuroscience.," *J. Cogn. Neurosci.*, pp. 1–21, 2015.

[3] E. Pereda, R. Q. Quiroga, and J. Bhattacharya, "Nonlinear multivariate analysis of neurophysiological signals," *Progress in Neurobiology*, vol. 77, no. 1–2. pp. 1–37, 2005.

[4] J. D. Bonita, L. C. C. Ambolode, B. M. Rosenberg, C. J. Cellucci, T. a a Watanabe, P. E. Rapp, and a. M. Albano, "Time domain measures of inter-channel EEG correlations: A comparison of linear, nonparametric and nonlinear measures," *Cogn. Neurodyn.*, vol. 8, no. 1, pp. 1–15, 2014.

[5] H. E. Wang, C. G. BÃ©nar, P. P. Quilichini, K. J. Friston, V. K. Jirsa, and C. Bernard, "A systematic framework for functional connectivity measures," *Front. Neurosci.*, vol. 8, Dec. 2014.

[6] C. E. Shannon and W. Weaver, *The Mathematical Theory of Information*. Urbana, Illinois: University Press, 1949.

[7] J. B. Kinney and G. S. Atwal, "Equitability, mutual information, and the maximal information coefficient.," *Proc. Natl. Acad. Sci. U. S. A.*, vol. 111, no. 9, pp. 3354–9, Mar. 2014.

[8] B. Pompe, P. Blidh, D. Hoyer, and M. Eiselt, "Using mutual information to measure coupling in the cardiorespiratory system.," *IEEE Eng. Med. Biol. Mag.*, vol. 17, no. 6, pp. 32–9, Jan. .

[9] K. Hlavackova-Schindler, M. Palus, M. Vejmelka, and J. Bhattacharya, "Causality detection based on information-theoretic approaches in time series analysis," *Phys. Reports-Review Sect. Phys. Lett.*, vol. 441, pp. 1–46, 2007.

[10] A. Kraskov, H. Stögbauer, and P. Grassberger, "Estimating mutual information," *Phys. Rev. E*, vol. 69, no. 6 2, p. 66138, 2004.

[11] D. Kugiumtzis and V. Kimiskidis, "Direct causal networks for the study of transcranial magnetic stimulation effects on focal epileptiform discharges," *Int. J. Neural Syst.*, 2014.

[12] R. Q. Quiroga, A. Kraskov, T. Kreuz, and P. Grassberger, "Performance of different synchronization measures in real data: A case study on electroencephalographic signals," *Phys. Rev. E*, vol. 65, no. 4 Pt 1, p. 41903, 2002.

[13] C. J. Stam, "Nonlinear dynamical analysis of EEG and MEG: Review of an emerging field," *Clin. Neurophysiol.*, vol. 116, no. 10, pp. 2266–2301, 2005.

[14] G. Sugihara, R. May, H. Ye, C. Hsieh, E. Deyle, M. Fogarty, and S. Munch, "Detecting causality in complex ecosystems.," *Science*, vol. 338, no. 6106, pp. 496–500, Oct. 2012.

[15] F. Takens, "Detecting strange attractors in turbulence," in *Lecture Notes in Mathematics*, 1981, vol. 898, no. 1, pp. 366–81.

[16] D. Chicharro and R. G. Andrzejak, "Reliable detection of directional couplings using rank statistics," *Phys. Rev. E*, vol. 80, no. 2, p. 26217, Aug. 2009.

[17] G. Niso, R. Bruña, E. Pereda, R. Gutiérrez, R. Bajo, F. Maestú, and F. Del-Pozo, "HERMES: towards an integrated toolbox to characterize functional and effective brain connectivity" *Neuroinformatics*, vol. 11, no. 4, pp. 405–34, Oct. 2013.

[18] P. Wollstadt, M. Martínez-Zarzuela, R. Vicente, F. J. Díaz-Pernas, and M. Wibral, "Efficient transfer entropy analysis of non-stationary neural time series.," *PLoS One*, vol. 9, no. 7, p. e102833, Jan. 2014.

[19] M. Vinck, R. Oostenveld, M. van Wingerden, F. Battaglia, and C. M. a Pennartz, "An improved index of phase-synchronization for electrophysiological data in the presence of volume-conduction, noise and sample-size bias.," *Neuroimage*, vol. 55, no. 4, pp. 1548–65, Apr. 2011.

[20] M. X. Cohen, "Effects of time lag and frequency matching on phase-based connectivity," *J. Neurosci. Methods*, Sep. 2014.

[21] S. Porz, M. Kiel, and K. Lehnertz, "Can spurious indications for phase synchronization due to superimposed signals be avoided?," *Chaos An Interdiscip. J. Nonlinear Sci.*, vol. 24, no. 3, p. 033112, Sep. 2014.

[22] M. G. Rosenblum, A. S. Pikovsky, and J. Kurths, "Phase synchronization of chaotic oscillators," *Phys. Rev. Lett.*, vol. 76, no. 11, pp. 1804–1807, 1996.

[23] P. Tass, M. G. Rosenblum, J. Weule, J. Kurths, A. Pikovsky, J. Volkmann, A. Schnitzler, and H. J. Freund, "Detection of n:m phase locking from noisy data: Application to magnetoencephalography," *Phys. Rev. Lett.*, vol. 81, no. 15, pp. 3291–3294, 1998.

[24] F. Mormann, K. Lehnertz, P. David, and C. E. Elger, "Mean phase coherence as a measure for phase synchronization and its application to the EEG of epilepsy patients," *Phys. D Nonlinear Phenom.*, vol. 144, no. 3–4, pp. 358–369, Oct. 2000.

[25] C. J. Stam, G. Nolte, and A. Daffertshofer, "Phase lag index: assessment of functional connectivity from multi channel EEG and MEG with diminished bias from common sources.," *Hum. Brain Mapp.*, vol. 28, no. 11, pp. 1178–93, Nov. 2007.

[26] R. Vicente, L. L. Gollo, C. R. Mirasso, I. Fischer, and G. Pipa, "Dynamical relaying can yield zero time lag neuronal synchrony despite long conduction delays," *Proc Natl Acad Sci U S A*, vol. 105, no. 44, pp. 17157–17162, 2008.

[27] M. Christodoulakis, A. Hadjipapas, E. S. Papathanasiou, M. Anastasiadou, S. S. Papacostas, and G. D. Mitsis, "On the Effect of Volume Conduction on Graph Theoretic Measures of Brain Networks in Epilepsy," in *Neuromethods*, Humana Press, 2014, pp. 1–28.

[28] M. C. Romano, M. Thiel, J. Kurths, K. Mergenthaler, and R. Engbert, "Hypothesis test for synchronization: twin surrogates revisited.," *Chaos*, vol. 19, p. 015108, 2009.

[29] M. Thiel, M. C. Romano, J. Kurths, M. Rolfs, and R. Kliegl, "Twin surrogates to test for complex synchronisation," *Europhys. Lett.*, vol. 75, no. 4, pp. 535–541, 2006.

[30] K. V Mardia and P. E. Jupp, *Directional Statistics*. John Wiley & Sons, 2000.

[31] D. Wilkie, "Rayleigh test for randomness of circular data," *Appl. Stutist*, vol. 32, pp. 311–2, 1983.

[32] Y. Benjamini and D. Yekutieli, "The control of the false discovery rate in multiple testing under dependency," *Ann. Stat.*, vol. 29, no. 4, pp. 1165–1188, 2001.

[33] H. Kantz and T. Schreiber, *Nonlinear Time Series Analysis*. Cambridge: Cambridge University Press, 2003.

[34] J. Arnhold, P. Grassberger, K. Lehnertz, and C. E. Elger, "A Robust Method for Detecting Interdependences: Application to Intracranially Recorded EEG," *Physica D*, vol. 134, no. 4, p. 29, Dec. 1999.

[35] A. Schmitz, "Measuring statistical dependence and coupling of subsystems," *Phys. Rev. E*, vol. 62, pp. 7508–7511, 2000.

[36] E. Pereda, R. Rial, A. Gamundi, and J. González, "Assessment of changing interdependencies between human electroencephalograms using nonlinear methods," *Physica D*, vol. 148, no. 1–2, pp. 147–158, 2001.

[37] R. G. Andrzejak, A. Kraskov, H. Stögbauer, H. Stogbauer, F. Mormann, and T. Kreuz, "Bivariate surrogate techniques: Necessity, strengths, and caveats," *Phys. Rev. E*, vol. 68, no. 6, p. 66202, 2003.

[38] C. J. Stam and B. W. Van Dijk, "Synchronization likelihood: An unbiased measure of generalized synchronization in multivariate data sets," *Phys. D Nonlinear Phenom.*, vol. 163, pp. 236–251, 2002.

[39] T. Montez, K. Linkenkaer-Hansen, B. W. van Dijk, and C. J. Stam, "Synchronization likelihood with explicit time-frequency priors.," *Neuroimage*, vol. 33, no. 4, pp. 1117–25, Dec. 2006.

[40] F. Rosales, A. García-Dopico, R. Bajo, and Á. Nevado, "An Efficient Implementation of the Synchronization Likelihood Algorithm for Functional Connectivity," *Neuroinformatics*, Dec. 2014.

[41] Ian H. Witten, Eibe Frank, "Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems) 2nd Ed., 2005

[42] D. N. Reshef, Y. A. Reshef, H. K. Finucane, S. R. Grossman, G. McVean, P. J. Turnbaugh, E. S. Lander, M. Mitzenmacher, and P. C. Sabeti, "Detecting Novel Associations in Large Data Sets," *Science*, vol. 334, no. 6062, pp. 1518–1524, Dec. 2011.

[43] D. Tang, M. Wang, W. Zheng, and H. Wang, "RapidMic: Rapid Computation of the Maximal Information Coefficient.," *Evol. Bioinform. Online*, vol. 10, pp. 11–6, Jan. 2014.

[44] M. C. Romano, M. Thiel, J. Kurths, I. Z. Kiss, and J. L. Hudson, "Detection of synchronization for non-phase-coherent and non-stationary data," *Europhys. Lett.*, vol. 71, no. 3, pp. 466–472, 2005.

[45] M. Le van Quyen, J. Foucher, J. P. Lachaux, E. Rodriguez, A. Lutz, J. Martinerie, and F. Varela, "Comparison of Hilbert transform and wavelet methods for the analysis of neuronal synchrony," *J. Neurosci. Methods*, vol. 111, p. 83, 2001.

[46] A. Bruns, "Fourier-, Hilbert- and wavelet-based signal analysis: are they really different approaches?," *J. Neurosci. Methods*, vol. 137, pp. 321–332, 2004.