Real-time neural signals decoding onto off-the-shelf DSP processors for neuroprosthetic applications

Danilo Pani, *Member, IEEE,* Gianluca Barabino, *Member, IEEE,* Luca Citi, *Member, IEEE,* Paolo Meloni, *Member, IEEE,* Stanisa Raspopovic, *Member, IEEE,* Silvestro Micera, *Senior Member, IEEE,* and Luigi Raffo *Member, IEEE,*

***Abstract*—The control of upper limb neuroprostheses through the peripheral nervous system (PNS) can allow restoring motor functions in amputees. At present, the important aspect of the real-time implementation of neural decoding algorithms on embedded systems has been often overlooked, notwithstanding the impact that limited hardware resources have on the efficiency/effectiveness of any given algorithm. Present study is addressing the optimization of a template matching based algorithm for PNS signals decoding that is a milestone for its real-time, full implementation onto a floating-point Digital Signal Processor (DSP).**

The proposed optimized real-time algorithm achieves up to 96% of correct classification on real PNS signals acquired through LIFE electrodes on animals, and can correctly sort spikes of a synthetic cortical dataset with sufficiently uncorrelated spike morphologies (93% average correct classification) comparably to the results obtained with top spike sorter (94% on average on the same dataset). The power consumption enables more than 24 hours processing at the maximum load, and latency model has been derived to enable a fair performance assessment. The final embodiment demonstrates the real-time performance onto a low-power off-the-shelf DSP, opening to experiments exploiting the efferent signals to control a motor neuroprosthesis.

***Index Terms*—Neural prosthesis, Real-time systems, Digital signal processing chips, Embedded software, Spike sorting**

I. INTRODUCTION

ACTIVE upper limb prostheses exploit external energy sources to give rise to specific movements interpreting the patient's intention, which is extracted from a physiological signal as Electroencephalogram (EEG), Electromyogram (EMG) and Electroneurogram (ENG). Despite EMG-controlled prostheses already entered the clinical practice, ENG ones, which are necessarily invasive, requiring to access the Peripheral Nervous System (PNS) [1], are still experimental. ENG is potentially able to provide more information than EMG to control more degrees of freedom (unless approaches such as targeted reinnervation [2] are used), at the expenses of a significantly higher computational complexity [3]. In order to control a motor neuroprosthesis, the decoding algorithm must be implemented onto an embedded signal processing platform, characterized by limited resources, low operating frequency and tight real-time bounds to fulfil. This aspect is often overlooked, assuming it as a second-order issue, with obvious consequences on the validity of the achieved results in a real scenario.

The aim of this paper is to present the real-time, full implementation onto a floating-point Digital Signal Processor (DSP), of a PNS signal decoding algorithm based on spike sorting and classification. To the best of our knowledge, this is the first time a completely embedded solution for PNS signals decoding, with a power profile compatible with a wearable implementation, is presented and evaluated. Different optimization aspects are discussed, from the simplest steps needed to switch from the off-line to an on-line implementation, to the algorithm modifications aimed at achieving the smallest memory footprint and improved code efficiency. DSPs are presented as valuable targets for on-line PNS signal decoding, being able to effectively perform the required functions with all the flexibility typical of microcoded implementations. Compared to the very preliminary porting exploration [4], the current solution enables the development of a completely stand-alone neuroprosthesis, re-trainable without external computers.

The remainder of this paper is organized as follows. In Sect. II, the background information on both the aim of this work and the chosen original algorithm is provided. Sect. III deals with the algorithm optimization and changes applied to achieve real-time performance onto an off-the-shelf DSPs. Porting details are provided in Sect. IV. The datasets used for testing are illustrated in Sect. V, whereas the achieved results are presented in Sect. VI. Conclusions are given in Sect. VII.

II. MOTIVATION AND BACKGROUND

Neural signal decoding is a critical signal processing stage required by CNS- and PNS-based neuroprostheses, mainly based on spike sorting algorithms to analyse the single neuron activity from extra-cellular recordings [5]. Thanks to the evolution of the neuronal interface at the PNS level with intrafascicular electrodes characterized by a good selectivity

D. Pani, G. Barabino, P. Meloni and L. Raffo are with the DIEE, Dept. of Electrical and Electronic Engineering, University of Cagliari, 09123 Cagliari, Italy {danilo.pani, gianluca.barabino, paolo.meloni, luigi}@diee.unica.it

L. Citi is with the School of Computer Science and Electronic Engineering, University of Essex, Colchester, United Kingdom lciti@essex.ac.uk

S. Raspopovic, S. Micera are with the Translational Neural Engineering Laboratory, Center for Neuroprosthetics and Institute of Bioengineering, School of Engineering, Ecole Polytechnique Federale de Lausanne, Lausanne, Switzerland and Scuola Superiore Sant'Anna, Pisa, Italy {stanisa.raspopovic, silvestro.micera}@epfl.ch

The research leading to these results has received funding by the Region of Sardinia in the ELoRA project (Fundamental Research Programme, L.R. 7/2007, grant agreement CRP-60544), by the European Commission in the NEBIAS project (FP7, FET Proactive, grant agreement 611687) and by the Italian Ministry of Health in the NEMESIS project (Young Researchers, grant agreement 064/GR-2009-1591615).

Manuscript received December 19, 2014; revised August 29, 2015.

[6], in principle it is possible to perform decoding on the PNS with techniques similar to those exploited for the CNS. However, at cortical level, Signal-to-Noise Ratio (SNR) is usually good enough to clearly recognize the action potentials, whereas on the peripheral nerves their amplitude is typically lower and so the SNR [7].

There is a huge number of techniques for CNS signals decoding [8]. Some of them undoubtedly have clear on-line vocation, as for those based on Kalman filtering [9], and have been successfully exploited on human subjects [10]. Other techniques, more specific for PNS signals and based on spike sorting and classification, are more challenging form a computational perspective [7]. Taking into account the final goal of this work, in this paper one of these algorithms has been selected [11]. Originally tested on the animal model to decode afferences, i.e. the tactile/proprioceptive stimuli provided on the leg where the neural interface is implanted, is has been successfully exploited for off-line decoding of efferences on a human amputee [12].

Despite the chosen algorithm had potentialities to be adopted in an on-line scenario, this is not true for other better algorithms such as one of the top state-of-the-art ones (SPC, [13]). This algorithm consists of the following processing steps:

- a 4^{th} order non-causal bidirectional elliptic IIR bandpass filtering between 300Hz and 3kHz;
- SD using a fixed amplitude threshold based on the signal variance;
- SS based on superparamagnetic clustering (SPC).

The algorithm assumes a refractory period of about 2 ms, not including any strategy for the overlapping spikes. For each detected spike, limited to 64 samples at a sampling frequency of 24 kHz by a time windowing, the algorithm uses as features for the classifier the wavelet transform coefficients that meet the criterion of Kolmogorov-Smirnof normality. As last stage, the algorithm includes the unsupervised SPC, automatically selecting through a Monte Carlo simulation the temperature parameter value, exploited to influence the cluster size and modifiable by the user in order to really achieve the best performance. This algorithm has been considered here as golden standard for performance comparisons.

A. Original formulation of the chosen algorithm

The chosen algorithm [11] consists of four stages: wavelet denoising (WD), spike detection (SD), spike sorting by template matching (SS) for feature extraction, and classification (CL). The signal is assumed to be sampled at 12 kHz. This is in line with the current literature, stating that 10 kHz is the optimal sampling frequency for similar recordings [14].

WD is performed exploiting a translation-invariant wavelet transform with *Symmlet 7* mother wavelet, three decomposition levels, hard thresholding of the details and complete removal of the approximation. The latter choice limits the signal bandwidth from 750Hz to 6 kHz. The minimax threshold method is used, providing a scaling factor for the standard deviation of the noise σ_n, computed through the median absolute deviation, over 45 s of quiescent signal.

SD is based on a fixed threshold, computed as three times the standard deviation of the samples in such time interval.

SS is performed by template matching based on the Pearson's correlation coefficient. Every new detected spike is extracted (variable support length) and compared by cross-correlation with the existing templates. After the spike and a template have been aligned on the highest correlation value, when: (I) the correlation value is higher than a fixed threshold $th_c = 0.9$, (II) the ratio between the mean square difference of the spike and template, and the power of the template, is less than 0.5, then the spike matches the template. The best matching template, t_i, (exhibiting the highest correlation, M_c) is selected. During the Tuning Phase (TPh), the templates are created when at least one of the above conditions cannot be satisfied, and the best matching template is updated by synchronized averaging every time all the conditions are met. At the end of TPh, the templates with a reciprocal correlation higher than th_c undergo a weighted synchronized averaging. Then, only the most used n_{feat} templates are retained for the SPh. After training, during Steady Phase (SPh), i.e. the same procedure is performed without any template update.

A pattern to be recognized by the classifier is a $1 \times n_{feat}$ array of features where every element represents the percentage of spikes (in a $L-$sample long window of the input signal, lasting t_L seconds, where t_L ranges from three to six seconds) with a waveform highly similar to one of the n_{feat} templates available from the SS stage. Classification is performed by a $\nu-$ Support Vector Machine (SVM) with radial basis function (RBF). To solve the multi-class problem (each class represent a decoded stimulus) one-against-one approach was used.

III. Optimizations for real-time performance

Before making any change to the algorithm, the architectural target needs to be identified. Application Specific Integrated Circuits (ASIC) and Field Programmable Gate Arrays (FPGA) represent an efficient solution but require a complex design and exhibit limited flexibility. In fact, ASICs cannot be modified after the production, whereas even though FPGAs can be reconfigured, this requires more effort than a simple program revision. Conversely, microprogrammed implementations are more flexible but less efficient, due to the general-purpose processor architecture. In this case, DSPs represent a valuable target, being embedded processors expressly designed for advanced signal processing. As any embedded processors, DSPs presents limited resources and programming issue to be considered. Nevertheless, when appropriately programmed, they could easily outperform mainstream desktop processors on signal processing tasks [15] or achieve comparable results with dramatically better power consumption performance [16].

Once identified the architectural target, as a broad family of devices such as DSPs, the algorithm needs to be adapted for the on-line execution, and the code *ported* on the selected specific device. Despite porting is a rather specific task depending on the chosen processor, also algorithms modifications to enable on-line effective and efficient processing depend on the chosen device.

1) Block-on-line processing: In order to cope with the conflicting requirements of processing and classification, a block-on-line approach has been chosen for the on-line implementation. The input is a stream at $f_s = 12$ kHz (signal samples) and the output is a class assigned to a classifier sample (hereafter called *pattern*) at $f_p = 4Hz$, which is similar to other works in the literature [17]. Furthermore, such a value, which can be modified, entails a fair delay and a reasonable refresh rate for the prosthesis commands. An additional trigger signal $@f_s$ is processed to mark the underlying neural activity for training purposes.

To achieve a good spikeness for the pattern, intended as the number of spikes per window, the observation window for features computing should be large enough to contain several spikes and should "slide" on the input signal with a frame rate r adequate to follow local variations in the signal in real time. Given t_L, the frame rate r defines the time instants t_n in which the patterns will be extracted to be processed by the classifier, i.e. $t_n = n\frac{t_L}{r}$ with $n = 1, 2, \cdots$. It is possible to choose $L = b\_len \times r$, so that b_len is the number of new signal samples, forming a $block$, needed to push on the window (the window slides with an overlap of $1 - r^{-1}$, e.g. 75% for $r = 4$). Since only the classifier requires to operate on the whole window, a "virtual" sliding window is implemented in the code for the first three stages (WD, SD, SS), the last one being in charge of preserving a memory of the spike sorting results for the latest r blocks (updated block-wise) to compute the features for a single pattern. This choice reduces the memory requirements and the computational redundancies. Noticeably, r is a user-defined parameter.

To provide an overview of the proposed algorithm in its on-line version, Fig. 1 describes with a pseudocode the different processing parts executed every time a new block of b_len input samples is available. The different conditions responsible for the transitions between the states of the algorithm can be automatically set by the code at run-time (e.g. whenever an established number of spikes has been processed, after a specific amount of time, in response to an external trigger).

The sequence of the macro-states of the algorithm changes over time in order to achieve the automatic tuning of the various stages before the normal operation:

1) WD with threshold tuning in loop with SD;
2) WD and SD with established thresholds, followed by SS in TPh;
3) templates reduction and definition of the final set of n_{feat} templates;
4) WD and SD with established thresholds, followed by SS in SPh;
5) CL creation and training;
6) WD and SD with established thresholds, followed by SS in SPh and then by CL (pattern recognition).

//WD
Wavelet decomposition (N_{scales} scales)
if 1^{st} block **then**
  Evaluate the parameters to compute the WD threshold, for this block
  Assign them to all the r_σ elements required to compute the WD threshold for a whole window
  Compute the WD thresholds at the different scales
else if $tuning\ threshold$ **then**
  Evaluate the parameters to compute the threshold, for this block
end if
for $i = 1$ to N_{scales} **do**
  Perform thresholding at scale 2^i
end for
Wavelet recomposition
//SD
Spike detection (different approaches)
//SS
Crosscorrelate incoming spike with existing templates (if any)
if $template\ creation$ mode **then**
  if $max(correlation) > th_c$ **then**
    Update corresponding template by synchronized averaging
  else
    Create a new template from incoming spike
  end if
else if passing from $template\ creation$ mode to $sorting$ mode **then**
  Perform templates merge and reduction to n_{feat} templates
else if $sorting$ mode **then**
  if $max(correlation) > th_c$ **then**
    Periodically update corresponding template by synchronized averaging
    Update templates count for feature creation of current pattern
  end if
end if
if $low\ spikeness$ AND $tuning\ threshold$ **then**
  Compute the thresholds at the different scales
  Update the WD threshold buffers
end if
//CL
if $sorting$ mode **then**
  if $training\ classifier$ **then**
    Classifier training
  else
    Classify the current pattern
  end if
end if

Fig. 1. Main loop of the on-line algorithm, for every new block of data (b_len samples).

A. Wavelet denoising optimization

For orthogonal wavelets, exploiting a dyadic discretization, WD can be implemented using a trellis of quadrature mirror FIR filters, a low-pass ($H(z)$)/high-pass ($G(z)$) pair for analysis and their mirrored versions for synthesis. The analysis produces the approximation signal ($a_{2^i}(z)$, low pass), passed to the next level, and the detail one ($d_{2^i}(z)$, high pass), retained for thresholding. Since every low-pass filter in the analysis trellis halves the Nyquist frequency of the approximation signal, it is usually downsampled for both computational and memory efficiency (the same filters can used in all the stages). Such a scheme is not translation invariant so in [11] it is combined with cycle spinning to overcome this limit. This approach is not suited for real-time processing because of the computational cost and the intrinsic off-line nature of the procedure. An *à trous* approach, which oversamples at each scale the analysis filters rather than downsampling the associated streams, has been rather adopted.

Furthermore, the *Symlet 7* mother wavelet, chosen to be similar to a typical action potential [14] to exploit matched filter properties in the processing, presents a large time support. This has reflections on both computational complexity and delay, compared to shorter wavelets such as the *Haar* one which will be alternatively considered.

Moreover, similarly to other works like [13], the WD-filtered signal bandwidth could be narrowed to 375Hz - 3kHz by adding one level of WD and cleaning the first detail. Remarkably, the upper cut-off frequency is still larger than 2 kHz, which is known to contain the most of the physiological information [14]. With four levels, a latency of 196 samples can be experienced with the *Symlet 7* whereas only 16 samples are required by the *Haar*. This affects the memory footprint too, since between analysis and synthesis, the detail signals need to be delayed to obtain the correct alignment in time. This is achieved writing with the correct offset in the recomposition filters buffer, taking into account the time lag as $t_{2^i} = t_{2^{i+1}} + lat_{2^{i+1}}$, where:

$$lat_{2^i} = N_{G_i(z)} - (N_{G_i(z)} \% 2) \tag{1}$$

$N_{G_i(z)}$ being the length of the $G_i(z)$ filter and % the reminder of the integer division.

The minimax method for threshold computing presents a dependency from the length N of the signal, which is misleading in on-line processing where the window length is extremely short. A fixed precomputed scaling factor equal to 3.9 has been used, corresponding to the value computed in [11]. Moreover, the sample standard deviation can be more efficiently used in place of the median absolute deviation since, in an on-line integrated approach, σ_n can be updated only when no spikes are detected in the signal. In order to evaluate σ_n over a frame of samples larger than b_len, i.e. $b\_len \times r_\sigma$, optimizing the latency, it is possible to compute for every incoming block of new b_len samples (at each scale i) their squared sum (all the detail coefficients are zero-mean) $s_{j,2^i} = \sum_{n=1}^{b\_len} d_{2^i}^2[n]$, so that for every new block of samples the last r_σ partial squared sums can be summed up, multiplied by $(b\_len \times r_\sigma - 1)^{-1}$ and then the squared root computed:

$$\sigma_{n_{2^i}} = \sqrt{\frac{1}{b\_len \times r_\sigma - 1} \sum_{j=1}^{r_\sigma} s_{j,2^i}} \tag{2}$$

B. Spike detection optimization

Rather than a simple amplitude threshold, another one based on a Nonlinear Energy Operator (NEO) [18] can be adopted to improve the performance, obtaining a pre-emphasis of the spikes:

$$\eta\{\hat{x}[n]\} = \hat{x}^2[n] - \hat{x}[n+1] \cdot \hat{x}[n-1] \tag{3}$$

The threshold is then obtained as a scaled version of the mean value of the NEO in a predefined time interval:

$$Thr = C \cdot \frac{1}{N} \sum_{n=1}^{N} \eta\{\hat{x}[n]\} \tag{4}$$

where C is empirically chosen. Tests performed on synthetic datasets [19] showed that the NEO is generally less sensitive to the choice of the amplitude threshold of the SD stage, and SD is more accurate and with limited computational complexity.

Only the detected spikes whose support is shorter than a parameter len (expressing a time value in terms of number of samples) are analysed. During the TPh, larger spikes are discarded whereas during SPh a smaller chunk of samples is possibly extracted looking at the threshold crossing only. This allows recognizing spikes embedded into bursts but does not address the problem of the overlapping spikes.

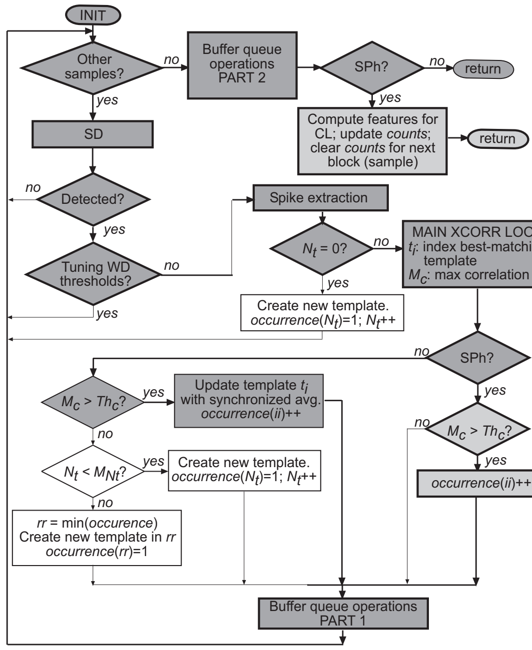


Fig. 2. The spike detection and sorting algorithms.

C. Spike sorting optimization

The SS is a correlation-based algorithm (Fig. 2), which must be able to work in real time both when the templates are being created (TPh) and when they are stable (SPh). When the WD thresholds are being computed, SS is skipped.

During the TPh, the algorithm can store up to M_{N_t} waveforms in a *templates matrix*. Such a structure is stored in a linear array, $len \times M_{N_t}$ words long, row major, for faster access on a DSP. An infinite template matrix is emulated during TPh by overwriting the template with the lowest occurrence when a new one needs to be stored and the matrix is full. To speed up the computation of the cross-correlations with the Pearson's coefficient, the templates are standardized to have zero mean and unitary variance.

During SPh, the SS acts as a feature extractor for the classifier, which operates downstream. A periodic template update can be allowed. Despite a new pattern is generated every 0.25s, it takes into account the ENG activity over $L = b\_len \times r$ samples. A matrix has been used to store row by row the occurrences of each template in each of the r blocks composing an $L-$wide sliding window. Such a matrix is updated column-wise at every new block and a pattern for the classifier is a feature vector obtained summing up by columns the occurrences of each template and dividing all the elements by the sum of the spikes occurrence in that window (the *spikeness* index). Such an approach reduces the processing time at the expenses of a slightly larger memory footprint. Before training, the patterns undergo a normalization process.

D. Classification optimization

Compared to the work presented in [4], the whole classifier has been recoded, and the training has been also embedded into the DSP firmware. A binary trigger has been exploited for on-line labelling during training. In an interactive training scenario, labelling can be performed exploiting GPIO pins of the processor, by manually selecting the performed (or desired) movement among a given set, or automatically (in case of rigidly structured and synchronized training procedure). To avoid calling the classifier in case of low ENG activity, when the trigger flag is active the pattern is passed to the classifier only if:

$$s \geq \mu_s + \frac{1}{2}\sigma_s \tag{5}$$

where μ_s and σ_s represent respectively the sample mean and standard deviation of the spikeness index s computed over the whole signal which will be used for training.

The soft-margin version of SVM classifier (C-SVC) with RBF kernel has been preferred to the original $\nu-$SVM, because of its easier optimization. The optimal values to be assigned to the various parameters to maximize the results in terms of classification accuracy and processing latency have been experimentally selected ($\gamma = 1$ for the RBF, cost $C = 1$ for the C-SVC, tolerance of the termination criterion equal to 0.1). The multi-class problem can be better solved by means of a "One-vs-The rest" approach rather than a "One-vs-One", since the number of classifier models to train is reduced to the number of classes.

E. Algorithm versions

Despite some of the modifications do not introduce differences in terms of algorithm effectiveness, some others could. To this aim, different versions of the algorithms have been developed and tested, changing some aspects related to the first three stages (regardless the chosen mother wavelet for WD).

- V1, similar to [4] and then to the original algorithm (WD with 3 levels, removing a_{2^3}, bandwidth 750Hz-6kHz, SS on the output of the WD stage) but the NEO has been introduced in the SD;
- V2, a modified version of V1 (WD with 3 levels, removing a_{2^3}, bandwidth 750Hz-6kHz, NEO for SD) but the SS is performed on the output of a 300Hz-3kHz band-pass filter as in [13], using WD only for the SD;
- V3, a modified version of V2 (NEO for SD, WD only for the SD, SS on the band-pass filtered signal and not on WD output) but exploiting a WD with 4 levels, removing a_{2^4} and also d_{2^1} leading to a bandwidth of 375Hz-3kHz which is close to [13];
- V4, similar to V1 (NEO for SD, SS on the output of the WD stage) but, as in V3, the WD has 4 levels, removing a_{2^4} and also d_{2^1} leading to a bandwidth of 375Hz-3kHz.

IV. Algorithm porting on a DSP platform

Despite the presented optimizations could be applicable to any architectural target, they have been studied to achieve the highest performance on DSP platforms. For instance, the choice of block-on-line processing rather than sample-by-sample processing, is beneficial for DSPs because of the presence of specialized hardware/software modules tacking advantage from block processing (Direct Memory Access (DMA), address generators, specialized software libraries, etc.).

As target for this study, we chose a TMS320C6748 floating-point DSP, hosted on the OMAP-L138 low-power applications processor by Texas Instruments, also including an ARM core and several shared resources. The ARM core can be exploited to perform some processing tasks or as host processor, easily managing the communication with external acquisition units, I/O peripherals such as a small detachable LCD display or a storage unit. The DSP can run up to 465MHz and presents a Very Long Instruction Word (VLIW) architectural model enabling the execution of up to eight arithmetic operations (mixed fixed- and floating-point) in parallel. The efficiency of the implementation on this kind of processor strongly depends on the VLIW code density: the best results can be obtained only by means of optimized libraries and an adequate coding technique, also respectful of the processor memory organization.

The chosen DSP presents a memory hierarchy organized in four levels. Level 1 (L1D and L1P, 32kB each) includes two memories configurable either as RAM or cache (data and instructions). Level 2 (L2, 256kB) is a RAM that can be configured partly or totally as second level cache. L1 and L2 are internal to the DSP core. Level 3 (L3, 128kB) is an on-chip shared memory, external to the DSP core, on the same clock domain of L1 and L2 and theoretically with the same bandwidth. Nevertheless L3 pays additional latency due to competing traffic, prioritization, and use of shared resources. Level 4 (L4) is the off-chip (external) memory. Potentially bigger, it introduces a penalty of several clock cycles compared to the others for the access. In our implementation, all the code sections working on the data at 12 kHz, and the related variables, have been placed in L2 (configured as all-RAM) along with the on-line CL code at 4Hz, whereas both the code and the variables used for the CL training have been placed in L4.

The application code has been written in C. All the parameters requiring an on-line tuning are automatically set up by the algorithm at run-time without any interaction with the user. The minimal set of functions of the C++ LIBSVM library [20], required by the implemented classifier, have been translated in C with limited memory footprint. All the advanced coding practices for this kind of platform, along with a careful memory allocation, have been exploited, and the code has been compiled with the highest optimization (-o3). Some optimizations, which can be enabled or not, limit the portability of the C code to the processors of the same family. For instance, some data movements are managed by the Enhanced Direct Memory Access (EDMA3) peripheral in order to execute them in parallel with the CPU processing. Advanced single-precision DSP library functions can be enabled at compile time to improve the VLIW code density, thus leading to a better parallelism exploitation, which in turn means that more instructions can be performed in parallel on the different issue slots of the processor. Some of these functions, e.g. the *DSPF_sp_fir_gen()*, used in the WD stage for the FIR filtering, take advantage of the chosen block-on-line approach, allowing to reach very high performance compared to a sample-by-sample approach. Another library, MATHLIB, has been used to speed up the processing of some scalar math operations.

V. Datasets exploited for testing

As stated before, different versions of the on-line algorithm have been proposed, along with a different mother wavelet selection (*Haar* rather than *Symlet 7*. In order to evaluate their effectiveness relying on a ground truth, limitedly to the first three stages of the algorithm, we exploited a synthetic dataset [13]. It includes artificial mixtures of real spikes (from three neurons), extracted from neocortex and basal ganglia, and additive noise representing the contribution from other far neurons. It can be also used also to test SS algorithms for PNS signals decoding, when the action potentials can still be identified because of a good SNR.

The synthetic dataset [13] includes different signals (Easy1, Easy2, Difficult1, Difficult2) with increasing levels of complexity. The complexity is related to the similarity in the morphology of the action potentials of the three neurons, revealed by the Pearson's correlation coefficient (Easy1 containing spikes correlated up to 0.8, the other datasets having higher correlation). Since the proposed algorithm is based on template matching, the results achieved on the Easy1 dataset shall be considered with the greatest attention. The background noise level is defined in terms of its standard deviation and it ranges from 0.05 to 0.40. The average firing rate is 20Hz, and the number of spikes per neuron is about 3000 in each one-minute signal.

In order to test the CL stage too, we exploited real afferent signals from the PNS, kindly provided by Prof. Xavier Navarro and his team (Universitat Autònoma de Barcelona), acquired through Longitudinal Intrafascicular Electrodes (LIFE) in the sciatic nerve of rats [21] and recorded according to the protocol used in [22]. Five classes of sensory events can be identified on segments of the available signal, i.e. touch sensation elicited over four different areas of the rat limb (A to D) stimulated with Von Frey filaments, and flexion movement performed with animal hind limb.

VI. Results

The evaluation of the proposed algorithm and its implementation is presented in this section in terms of effectiveness (quality of the different versions) and efficiency (in the light of a real-time implementation). Effectiveness analysis will lead to the selection of the best version to be analysed in terms of efficiency, measured in terms of processing capabilities and power consumption.

A. Effectiveness analysis

PNS neural signals are usually affected by a low SNR [7]. From this perspective, WD represents an important pre-processing stage whose influence on the spike detection can be demonstrated by using the synthetic dataset. V1-V4 version, with both Symlet 7 and Haar wavelets, are compared with the SPC only in terms of spike detection figures of merit, i.e. True Positives (TP) rate (the percentage of true detected spikes over those present in the signal, Fig. 3) and False Positives (FP) per minute (Fig. 4).

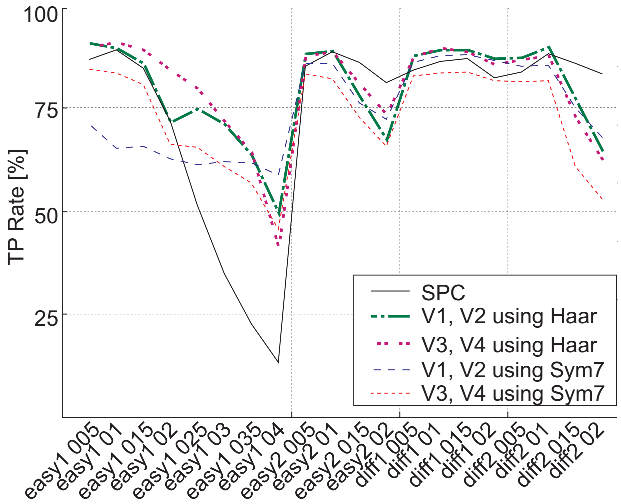


Fig. 3. Spike detection results in terms of TP rate using the Symlet 7 and the Haar wavelets compared to SPC.

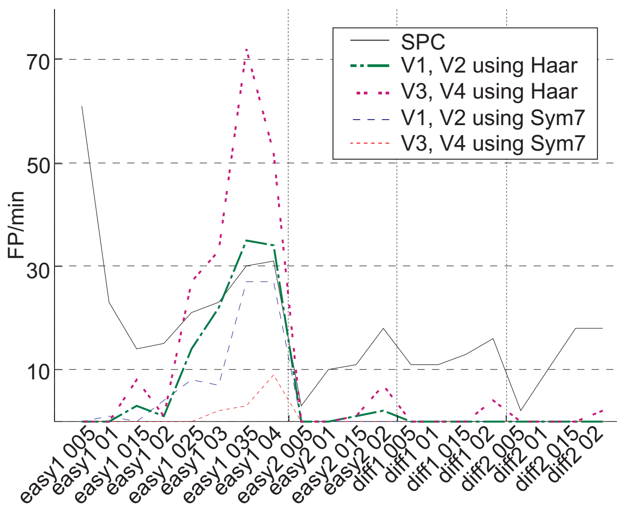


Fig. 4. Spike detection results in terms of FP per minute using the Symlet 7 and the Haar wavelets compared to SPC.

In both cases, the proposed SD approach, combining WD and NEO, performs better than the one of the SPC for the largest part of the signals, with isolated exceptions on the Easy1 database. The *Haar* wavelet reveals a higher TP rate than the *Symlet 7* but also a higher number of FP/min.

SS performance has been evaluated in terms of percentage of matching. From Tab. I we can see that the results seem to be strongly influenced by the noise level and are largely worse than those achievable off-line with the SPC for all the dataset but the Easy1 one, as expected. The best performance can be achieved with the V4 version of the algorithm and the *Haar* wavelet. Results are definitely worse for the versions of the

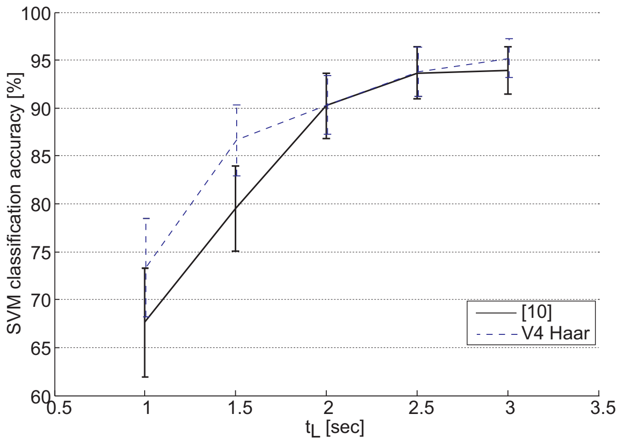


Fig. 5. SVM classifier accuracy (mean and standard deviation over 3000 random training/test pairs on the same dataset) with $len = 44$ samples and varying t_L, comparing the algorithm implemented in [4] with the current V4 version using the Haar wavelet.

on-line algorithm performing the SS on a band-pass filtered version of the signal rather than on the wavelet denoised one. This seems to be correlated to the limited effect of band-pass linear filtering on such signals when performing spike sorting by correlation methods. Again the *Haar* wavelet seems to be more effective than the *Symlet 7*. Remarkably, the SPC works with features obtained from wavelet analysis, so that the effect of band-pass filtering is superimposed to that of wavelet sub-band analysis.

Since all the V1-V4 versions of the on-line algorithm present the NEO as SD stage, it is worth to evaluate how much this stage influences both detection and sorting, compared to a solution based on a fixed threshold. Limiting our analysis to V4 with the *Haar* wavelet, the behaviour of the two solutions is similar, the NEO allowing a considerable reduction of the FP/min, up to 75%. All the more so, in terms of matching, as can be seen comparing SPC, *V4-H* and *V4-H abs* rows in Tab. I, the results exploiting the NEO in the spike detection are better than those achievable with a fixed threshold.

It is then worth to see whether such results reflect what happens on the real PNS signals. Tests have been performed evaluating the classification accuracy while varying the time frame for a pattern (t_L), comparing the algorithm implemented in [4] with the current V4 version (Fig. 5). Accuracy has been computed as average over 3000 different training/test partitions of the same dataset, randomly divided every time using 80% of samples for the training set and the rest for the test set. V4 is more robust to the variation of t_L, allowing to obtain a more efficient hardware implementation for the same classifier performance. Moreover, its accuracy is always higher, particularly for the most interesting low values of t_L, from a minimum of about 74% to a maximum of 96% with a standard deviation that decreases as t_L increases.

B. Efficiency analysis

The previous analysis reveals how, compared to the original algorithm and to its first tentative porting, the proposed modifications lead to improved effectiveness performance. In particular, the best version is V4, using the *Haar* wavelet. This solution has been then profiled in order to derive a latency model. This is a mathematical model, descending from cycle-accurate profiling, able to provide (with some approximations) the expected latency (CPU clock cycles) for a given code under different test conditions. Since the code optimizations, and consequently its performance, can be different if only a part of the code is compiled, in order to create a fair latency model the code was not modified (except for the parameters to tune) for the analysis of the different sections but the input data changed in order to be able to trigger specific behaviours. Compared to other techniques, this approach is very time consuming but allows pursuing accuracy.

The main issue in the creation of the latency model is the high number of branches that the algorithm can take during the different phases of its execution. Referring to Fig. 2, it is clear that different latencies will be experienced when only the SD is performed or when the system is creating the templates (TPh) or during normal operation (SPh). Even if we created a model for each possible combination, varying the proper parameters, only the two worst-case working conditions will be analysed hereafter. The worst case in the TPh is identified in Fig. 2 with the darkest shading, whereas the branches including the lighter shading are those related to the SPh. White blocks can be considered not part of any worst case scenario. Only the parts of the code that must be executed in real time have been evaluated for the latency model: the CL training and the templates reduction have been profiled apart.

From the code profiling, the WD for the V4 version of the algorithm, with the chosen window length, requires about 320kcycles with the *Haar* wavelet. Such a result cannot be compared with that reported in [4] because the current memory requirements are higher and then a larger use of the external memory is required, with the consequent latency penalties. Within the current framework, the implementation of the original 3-level WD stage with the *Symlet 7* wavelet would require about 640kcycles. Taking also into account the demonstrated superiority in terms of performance of the *Haar* wavelet compared to the *Symlet 7* one, the adoption of the former should be preferred. The final model during the TPh is described by the following relation:

$$\begin{aligned} c = [2s_{len} + 1009.2 + (911.2M_{N_t} + 20.2)\, len + \dots \\ -16520M_{N_t}]\, n_s + 514495 \end{aligned} \tag{6}$$

where c is the cycle count, s_{len} is the duration in samples of the spikes, M_{N_t} is the number of elements of the templates matrix, len is the template length in samples, n_s is the number of spikes in a b_len-sample window ($b\_len = 3000$ in our case). It is worth to note that the dependence from s_{len} is very small compared to the other parameters. For this reason, considering an average case of $s_{len} = 16.6$, as from the available signals, the final model during the TPh is:

$$\begin{aligned} c = [1042.4 + (911.2M_{N_t} + 20.2)\, len - 16520M_{N_t}]\, n_s + \dots \\ +514495 \end{aligned} \tag{7}$$

TABLE I
TEMPLATE MATCHING PERCENTAGE USING THE HAAR AND THE SYMLET 7 MOTHER WAVELET, ON THE SYNTHETIC DATASET [13].

	E1_005	**E1_01**	**E1_015**	**E1_02**	**E1_025**	**E1_03**	**E1_035**	**E1_04**	**E2_005**	**E2_01**	**E2_015**	**E2_02**	**D1_005**	**D1_01**	**D1_015**	**D1_02**	**D2_005**	**D2_01**	**D2_015**	**D2_02**
SPC	94	96	95	96	96	92	92	90	95	95	83	58	97	96	85	43	96	97	62	15
V1-H	66	69	71	64	85	72	89	80	32	28	34	15	15	38	28	36	54	51	55	59
V1-S7	63	59	77	61	73	74	72	72	60	58	43	42	26	47	43	30	49	49	41	50
V2-H	96	95	95	89	75	62	65	39	63	78	64	50	50	39	32	33	49	64	52	48
V2-S7	94	93	90	87	86	47	44	28	94	77	64	27	49	42	33	24	49	62	56	47
V3-H	96	96	95	87	85	67	61	46	82	79	52	46	45	39	36	22	48	52	51	48
V3-S7	96	96	93	85	78	47	52	35	62	77	67	41	44	48	33	30	48	63	49	54
V4-H	96	96	97	95	95	93	90	89	62	67	31	27	30	37	43	48	56	52	60	63
V4-S7	95	81	94	95	59	78	51	57	57	60	31	51	34	50	52	25	50	58	53	21
V4-H abs	96	96	96	96	95	92	88	83	60	41	14	39	28	37	31	36	57	44	53	63

Using this model, it is possible either to:

- know the latency for a given n_s when the maximum number of templates in this phase is fixed to M_{N_t} and every template length is len or
- invert the model considering the maximum number of available cycles (Real-Time Bound RTB, which is 116.25Mcycles when clocking the DSP at 465 MHz) so that, fixing the structural parameters, it is possible to know how many spikes can be analysed in real time.

The second choice leads to the following model:

$$n_s = \frac{RTB - 514495}{1042.4 + (911.2M_{N_t} + 20.2)\,len - 16520M_{N_t}} \quad (8)$$

Some of the implemented optimizations forbid the adoption of this model at a very fine granularity because, for instance, the parameter len must be a multiple of four. The set of curves in Fig. 6 (top chart) allows evaluating the maximum number of processable spikes per second, during TPh, as a function of M_{N_t}, for different values of len.

Following a similar reasoning during the SPh, the final latency model is the following, where n_{feat} is the number of templates after their fusion (i.e. in the SPh) and n_{SV} is the number of support vectors.

$$n_s = \frac{RTB - 280n_{feat} - 1133n_{SV} - 514981}{(911.2len - 16520)\,n_{feat} + 578.5} \quad (9)$$

The number of spikes per second in this case is depicted in Fig. 6 (bottom chart), where n_{SV} has been fixed to a typical value of 700. With 40 templates in TPh and 10 in the SPh, and exploiting the same len of the SPC algorithm for the synthetic database (32 samples at 12kHz), the on-line algorithm is able to process more than 890 spikes per second in TPh and more than 3500 in SPh, which represents an important result in terms of possible exploitation even in case of extension to multichannel recordings. In this case, the computational power can be distributed across the different channels with acceptable performance levels.

In order to be able to appreciate the distribution of the computational load across the different stages of the algorithm, a point in the parameters domain has been taken and the amount of clock cycles required to the processor to carry out the different stages of the algorithm is presented in Tab. II.

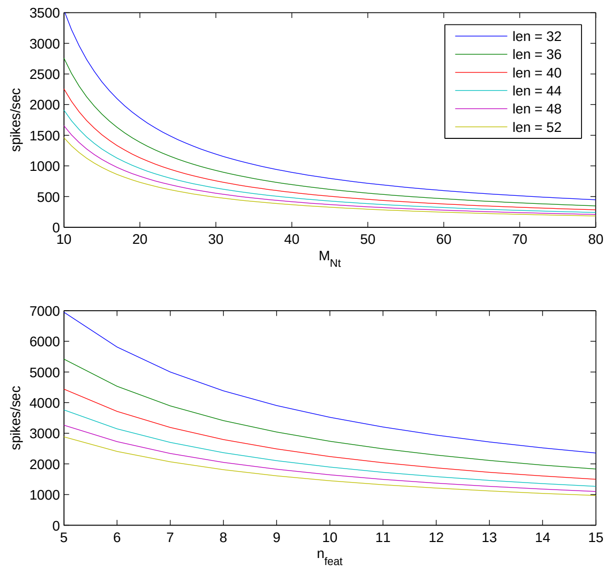


Fig. 6. The maximum of processable spikes per second in TPh (top) and SPh (bottom) as a function of M_{N_t} and n_{feat} respectively (from top to bottom, the curves are drawn for increasing values of len).

TABLE II
NUMBER OF CYCLES REQUIRED FOR THE EXECUTION OF THE FOUR PROCESSING STEPS ON A BLOCK OF 3000 SAMPLES. IN THIS CASE len = 40, M_{N_t} = 40 IN TPH, n_feat = 10 IN SPH, n_{SV} = 700 AND n_s = 100, WHICH IS COMPATIBLE WITH HAVING A TRAIN OF 1.4 MS SPIKES, SEPARATED BY A REFRACTORY PERIOD OF 1 MS.

	WD	**NEO+SD**	**SS**	**CL**
TPh	320437	136285	75443300	-
SPh	320437	136285	15072000	795900

In order to investigate the accuracy of the latency model in the steady state, actual profiling data was compared to the results obtained with the model for a different number of incoming spikes. The same parameters used for the algorithm evaluation on the real dataset (len = 40, n_{feat} = 10) was employed. The model prediction error was of 5% on average (standard deviation 0.58%). Remarkably, the model always overestimates the actual cycle count.

In terms of memory footprint, the real-time code requires only 39kB of memory and the data in L2 expressly instantiated in the code require 177kB of memory. Taking into account also the other code sections (used by the DSP/BIOS operating system of the DSP and its variables), the code on the internal RAM reaches 252kB of the available 256kB. The L4 memory usage reaches 580kB, including 259kB for the external data section of the CL and 26kB for the related code.

C. Power consumption analysis

An accurate estimate of the power consumption of the more demanding component in the system, the OMAP processor, can be achieved by using the spreadsheet available in [23]. Accuracy depends on the realism of the operating parameters used to fill in the spreadsheet. Compared to performing actual measurements on an EVM, this methods allows overriding the power consumption of the various hardware on-board components sinking current from the power source although they are not used. For this application, we chose a configuration where both cores (DSP and ARM) are enabled and the mDDR SDRAM, the EDMA3, PLL0 module, the SPI0 port and the USB1.1 port (useful as interfaces towards external devices, e.g. an acquisition unit) are also active. Percentage of utilization of the SPI0 port has been obtained by comparing the maximum bit rate for the port (50Mbps) with the expected one (12kHz, 8 channels, 16 bit per sample yield about 1Mbps). The utilization of the mDDR has been derived directly from the cycle accurate profiling of the application by counting the number of cycles spent to perform data movement from/to the L4 memory. A junction temperature of 50 °C has been used. The ARM core has been supposed to be under a typical workload. Different device frequencies and DSP loads have been tested in order to obtain the power consumption vs. the number of processed spikes, as shown in Fig. 7.

From this analysis it is possible to estimate which power source could be used in practice. For example, with a standard Li-Ion single cell battery (4.2V, 3000mAh capacity), supposing the system must process 400 spikes/sec (which is compatible with the duration of a spike), the projected duration of the battery would be 34 hours when clocking the DSP at 456MHz or 45 hours at 300 MHz. These numbers are compatible with a real scenario.

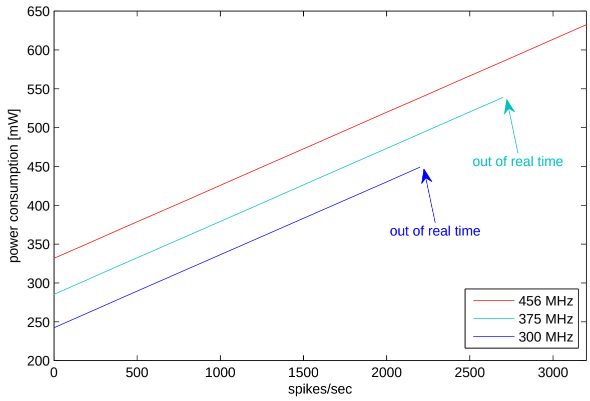


Fig. 7. Power consumption of the OMAP-L138 as a function of the number of processed spikes during the SPh phase (from top to bottom, the curves are drawn for decreasing values of operating frequency).

VII. Conclusions

This paper presents a complete study of the optimization of a state-of-the-art algorithm for PNS signals decoding, already exploited off-line on animals and humans, in order to obtain the best performance on a limited-resources embedded platform such as an off-the-shelf DSP. Compared to custom VLSI or FPGA implementations, the adoption of these highly efficient micro-programmed architectures leads to a greater flexibility. For the time being, this is particularly useful for closed-loop experiments when the signal processing algorithms need to be quickly adapted to previous experimental evidences. The proposed work also identifies improvements to the original algorithm able to guarantee the real-time performance with higher quality in the results compared to the original version.

Several tests have been performed both on synthetic neural datasets and on real afferent signals recorded in-vivo from rodents. The optimal version of the algorithm allows achieving an accuracy up to 96% in classification. Spikes sorting performance on a synthetic dataset is on average of 93% correct classification, which is comparable to the results obtained with a top spike sorter (94%) on the same dataset. Due to the template matching nature of the implemented algorithm, such results are limited to the Easy1 part of the dataset, taking into account that the others presents spikes with a correlation higher than 0.8.

The derived latency model allows the identification of the working point under different parameters setting. In a single-channel implementation, the algorithm is able to process 890 spikes per second when the unsupervised templates creation procedure is running (working on 40 templates), and up to 3500 after training (with 10 templates), in both cases with 2.7ms templates. Such numbers prompts the possible extension to a multi-channel scenario involving closed-loop real-time experiments including the complex phase of classifier training, now included in the same embedded framework so that also the training phase could be carried out without any external tool. Power consumption reveals performance compatible with a usage period longer than 24 hours, with a fanless device requiring about 600mW in the worst-case condition.

Acknowledgment

The authors gratefully thank Prof. Xavier Navarro and his team (Universitat Autònoma de Barcelona) for the collaboration in the real data acquisition in rats.

References

[1] S. Micera, P. M. Rossini, J. Rigosa, L. Citi, J. Carpaneto, S. Raspopovic, M. Tombini, C. Cipriani, G. Assenza, M. C. Carrozza, K.-P. Hoffmann, K. Yoshida, X. Navarro, and P. Dario, "Decoding of grasping information from neural signals recorded using peripheral intrafascicular interfaces," *Journal of NeuroEngineering and Rehabilitation*, vol. 8, no. 53, pp. 1038–1051, 2011.

[2] T. A. Kuiken, L. A. Miller, R. D. Lipschutz, B. A. Lock, K. Stubblefield, P. D. Marasco, P. Zhou, and G. A. Dumanian, "Targeted reinnervation for enhanced prosthetic arm function in a woman with a proximal amputation: a case study," *Lancet*, vol. 369, pp. 371–380, Feb. 2007.

[3] X. Jia, M. A. Koenig, X. Zhang, J. Zhang, T. Chen, and Z. Chen, "Residual motor signal in long-term human severed peripheral nerves and feasibility of neural signal-controlled artificial limb," *J Hand Surg Am*, vol. 32, pp. 657–666, 2007.

[4] D. Pani, F. Usai, L. Citi, and L. Raffo, "Real-time processing of tfLIFE neural signals on embedded dsp platforms: a case study," in *Proc. 5th International IEEE EMBS Conference on Neural Engineering*, 2011, pp. 44–47.

[5] H. Rey, C. Pedreira, and R. Q. Quiroga, "Past, present and future of spike sorting techniques," *Brain Research Bulletin*, 2015.

[6] J. Badia, T. Boretius, D. Andreu, C. Azevedo-Coste, T. Stieglitz, and X. Navarro, "Comparative analysis of transverse intrafascicular multichannel, longitudinal intrafascicular and multipolar cuff electrodes for the selective stimulation of nerve fascicles," *Journal of Neural Engineering*, vol. 8, no. 3, p. 036023, 2011.

[7] S. Micera, J. Carpaneto, and S. Raspopovic, "Control of hand prostheses using peripheral information," *Biomedical Engineering, IEEE Reviews in*, vol. 3, pp. 48–68, 2010.

[8] E. N. Brown, R. E. Kass, and P. P. Mitra, "Multiple neural spike train data analysis: State-of-the-art and future challenges," *Nature Neuroscience*, vol. 7, pp. 456–461, 2004.

[9] W. Wu and N. Hatsopoulos, "Real-time decoding of nonstationary neural activity in motor cortex," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 16, no. 3, pp. 213–222, June 2008.

[10] L. R. Hochberga, M. D. Serruya, G. M. Friehs, J. A. Mukand, M. Saleh, A. H. Caplan, A. Branner, D. Chen, R. D. Penn, , and J. P. Donoghue, "Neuronal ensemble control of prosthetic devices by a human with tetraplegia," *Nature*, vol. 442, pp. 164–171, 2006.

[11] L. Citi, J. Carpaneto, K. Yoshida, K.-P. Hoffmann, K. P. Koch, P. Dario, and S. Micera, "On the use of wavelet denoising and spike sorting techniques to process electroneurographic signals recorded using intraneural electrodes," *Journal of Neuroscience Methods*, vol. 172, pp. 294–302, 2008.

[12] P. Rossini, S. Micera, A. Benvenuto, J. Carpaneto, G. Cavallo, L. Citi, C. Cipriani, L. Denaro, V. Denaro, G. D. Pino, F. Ferreri, E. Guglielmelli, K. Hoffmann, S. Raspopovic, J. Rigosa, L. Rossini, M. Tombini, and P. Dario, "Double nerve intraneural interface implant on a human amputee for robotic hand control," *Clin. Neurophysiol.*, no. 121, pp. 777–883, May 2010.

[13] R. Q. Quiroga, Z. Nadasdy, and Y. Ben-Shaul, "Unsupervised spike detection and sorting with wavelets and superparamagnetic clustering," *Neural Comput.*, vol. 16, no. 8, pp. 1661–1687, Aug. 2004.

[14] A. Diedrich, W. Charoensuk, R. J. Brychta, A. C. Ertl, and R. Shiavi, "Analysis of raw microneurographic recordings based on wavelet denoising technique and classification algorithm: wavelet analysis in microneurography," *IEEE Transactions on Biomedical Engineering*, vol. 50, no. 1, pp. 41–50, Jan. 2003.

[15] D. Pani, A. Pani, and L. Raffo, "Real-time blind audio source separation: performance assessment on an advanced digital signal processor," *The Journal of Supercomputing*, vol. 70, no. 3, pp. 1555–1576, Dec. 2014.

[16] D. Pani, G. Barabino, and L. Raffo, "NInFEA: an embedded framework for the real-time evaluation of fetal ECG extraction algorithms," *Biomedizinische Technik/Biomedical Engineering*, vol. 58, no. 1, pp. 13–26, Dec. 2012.

[17] W. Wu, M. J. Black, Y. Gao, M. Serruya, A. Shaikhouni, J. P. Donoghue, and E. Bienenstock, "Neural decoding of cursor motion using a kalman filter," in *Advances in Neural Information Processing Systems 15*. MIT Press, 2002, pp. 117–124.

[18] J. Kaiser, "On a simple algorithm to calculate the 'energy' of a signal," in *Proc. International Conference on Acoustics, Speech, and Signal Processing, ICASSP-90*, vol. 1, Apr. 1990, pp. 381–384.

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

[20] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, pp. 27:1–27:27, 2011, software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[21] X. Navarro, N. Lago, M. Vivo, K. Yoshida, K. Koch, W. Poppendieck, and S. Micera, "Neurobiological evaluation of thin-film longitudinal intrafascicular electrodes as a peripheral nerve interface," in *IEEE 10th International Conference on Rehabilitation Robotics ICORR 2007*, Jun. 2007, pp. 643–649.

[22] S. Raspopovic, J. Carpaneto, E. Udina, X. Navarro, and S. Micera, "On the identification of sensory information from mixed nerves by using single-channel cuff electrodes," *J Neuroeng Rehabil.*, vol. 7, no. 17, Apr. 2010.

[23] "Omap-l138 power consumption summary," last accessed 10-Oct-2014. [Online]. Available: http://processors.wiki.ti.com/index.php/OMAP-L138_Power_Consumption_Summary\#Download

Danilo Pani, Ph.D., is non-tenure Assistant Professor in Biomedical Engineering at the Dept. Electrical and Electronic Engineering, University of Cagliari, Italy. Current main research topics are embedded real-time biomedical signal processing, wearable biomedical systems and tele-health.

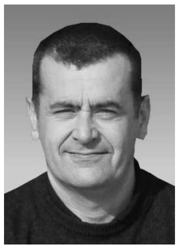
Gianluca Barabino, is Ph.D. student at the Dept. Electrical and Electronic Engineering of the University of Cagliari, Italy. His main research interests are in the field of high performance DSP architectures and algorithms, real-time neural signal processing algorithms and systems, tele-health.

Luca Citi received a PhD in biorobotics science and engineering at IMT and Scuola Superiore Sant'Anna, Italy. He worked as postdoctoral research fellow at MGH/Harvard Medical School, Boston, and was affiliated with MIT, Cambridge. He is currently a lecturer in computational intelligence at the University of Essex, Colchester, UK.

Paolo Meloni is currently assistant professor at the Department of Electrical and Electronic Engineering (DIEE) in the University of Cagliari. His research activity is mainly focused on the development of advanced digital systems, with special emphasis on the application-driven design and programming of multi-core on-chip architectures.

Stanisa Raspopovic, PH.D, is scientist at both the EPFL (Lausanne, Switzerland) and Scuola Superiore Sant'Anna (Pisa, Italy). His research interests include the bidirectional prosthetic control for amputees, hybrid modeling of electrical stimulation of nerves, epidural electrical stimulation for movement restoration in spinal cord injury and clinical translation of the animal experimentation.

Silvestro Micera, Ph.D. is Professor of Biomedical Engineering at the Scuola Superiore Sant'Anna (Pisa, Italy), and Associate Professor of Biomedical Engineering at the EPFL (Lausanne, Switzerland). His research interests include the development of neuroprostheses based on the use of implantable neural interfaces with the central and peripheral nervous system.



Luigi Raffo, Ph.D., is Full Professor of Electronics at the University of Cagliari, Italy, since 2006. He was President of the Course of Studies in Biomedical Engineering (2006 to 2012), and coordinator of several international projects. His research activity is in the field of embedded systems and biomedical signal processing.