Resolution of Superpositions in EMG Signals using Belief Propagation: Results for the Known Constituent Problem

Volker M. Koch[1,2], Kevin C. McGill[2], and Hans-Andrea Loeliger[1]
[1] Signal and Information Processing Laboratory, ETH Zurich, 8092 Zurich, Switzerland
[2] Rehabilitation R&D Center, VA Palo Alto Health Care System, Palo Alto, CA 94304, U.S.A.
koch@isi.ee.ethz.ch, mcgill@va51.stanford.edu, loeliger@isi.ee.ethz.ch

***Abstract*— The problem of resolving superpositions in electromyographic (EMG) signals is considered. The shapes of the motor unit action potentials that make up each superposition are assumed to be known a-priori (known constituent problem). Two different and novel belief propagation algorithms have been developed to solve this problem. These algorithms and simulation results are presented in this paper.**

***Index Terms*— EMG signal decomposition, factor graphs, belief propagation**

I. Introduction

EMG signals: Muscle contraction produces electrical activity that can be measured and analyzed. An electromyographic (EMG) signal consists of contributions from several sources (motor units) which discharge repeatedly producing spikes called motor unit action potentials (MUAPs). Four MUAPs are shown at the bottom of Fig. 8. A block diagram as in Fig. 1 can be used to model and to simulate EMG signals: source i emits a discrete-time binary signal $X_i \triangleq (X_{i,1}, X_{i,2}, X_{i,3}, \ldots)$ with $X_{i,k} \in \{0,1\}$. If $X_{i,k} = 1$, we say that source i "fires" at time k. Each electrode picks up a noisy and filtered superposition of these source signals. For example, electrode j picks up the EMG signal $Y_j \triangleq (Y_{j,1}, Y_{j,2}, Y_{j,3}, \ldots)$ with

$$Y_{j,k} = \sum_{i=1}^{N_{\mathrm{src}}} \sum_{\ell=0}^{M_i} X_{i,k-\ell} \cdot h_{i,j,\ell} + W_{j,k}, \qquad (1)$$

where $h_{i,j,\ell} \in \mathbb{R}$ is the ℓ-th filter coefficients expressing the waveform of MUAP i in channel (electrode) j, N_{src} is the number of sources, M_i is the order of the FIR filter corresponding to source i, and $W_j \triangleq (W_{j,1}, W_{j,2}, W_{j,3}, \ldots)$ is additive white Gaussian noise (AWGN) for channel j.

Problem statement: Given the block diagram in Fig. 1, decomposing a multi-channel EMG signal $Y \triangleq (Y_1, Y_2, \ldots)$ means estimating the binary source signals X_i. Many decomposition algorithms have been proposed. However, many practical tools are limited to a few superimposed MUAPs, e.g., only two in [1].

This paper: This paper considers EMG signals consisting of a single superposition as shown in Fig. 8. We want to resolve such superpositions given all filter coefficients $h_{i,j,\ell}$ and assuming that each source fires exactly one time (known constituent problem). For this, we present two new EMG signal decomposition algorithms, which are extended versions of previously outlined algorithms in [2][3]. As in these papers, our new algorithms are based on graphical models (factor graphs [4]). Non-optimal message-passing (belief propagation) algorithms run on these factor graphs to resolve the superpositions by calculating messages that are sent along the edges of the factor graph. The edges correspond to variables.

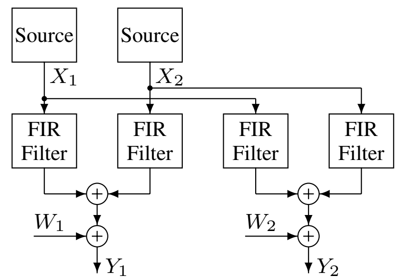

Fig. 1. Model of EMG signal generation for two sources and two channels

II. Methodology

Factor graph: Based on the block diagram in Fig. 1, we constructed a factor graph. Fig. 2 shows one section of this factor graph. The complete factor graph has one such section for each discrete time k. Note the similarities between the block diagram and the factor graph. We explain the individual nodes of the factor graph below.

Belief propagation algorithm and messages: We use the sum-product rule [4] to calculate an approximation of the marginal conditional probability distributions of the binary source signals $X \triangleq (X_1, X_2, \ldots)$.

For continuous variables, the messages that would ideally be used are scaled continuous probability density functions. Since general continuous functions can not be stored in memory, they have to be represented/approximated in some way. For example, they can be approximated by joint Gaussian density functions. In this case, the message can consist of only a mean vector and a covariance matrix. Alternatively, continuous messages can be represented by discrete messages (by discretizing variables) or lists of samples/weights.

In this paper, we present results for two algorithms: one with discrete messages only (runs on the factor graph in Fig. 2), and another one with discrete messages for the variables X and S, and joint-Gaussian messages for the

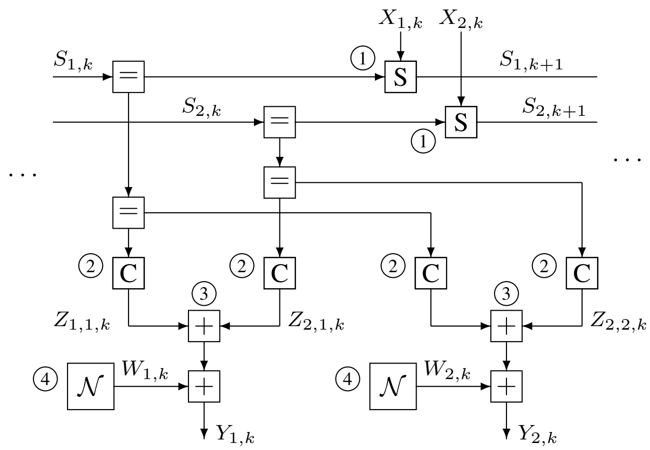

Fig. 2. Time-k section of a factor graph for the discrete case

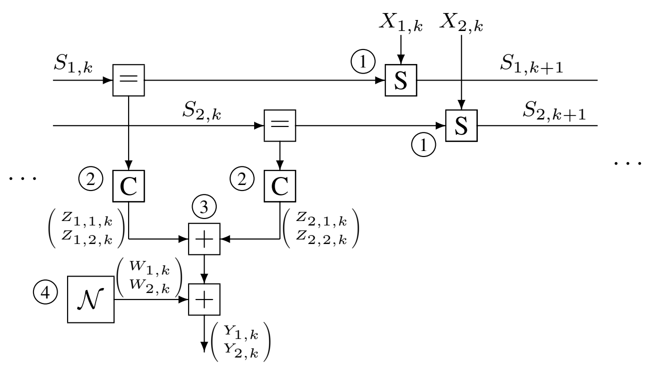

Fig. 3. Time-k section of a factor graph for the joint-Gaussian case

variables below the coefficient nodes ② (runs on the factor graph in Fig. 3).

① **Source nodes:** The boxes labeled ① in Fig. 2 and Fig. 3 represent the state transition probabilities $p(s_{i,k+1}|s_{i,k})$, which are defined by the finite state model in Fig. 4. This

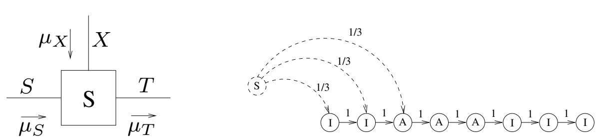

Fig. 4. Left: source node; right: finite state model of one source

finite state model is basically a shift register with idle states Ⓘ and active states Ⓐ. State Ⓢ is the initial start state at time "$k = -1$", which is only used for simulating signals; it is not used in the decomposition algorithm. The first $M_i + 1$ (length of the MUAPs of source i) active states in Fig. 4 correspond to the taps of the FIR filter that models MUAP i. We expect the MUAPs to lie completely within the EMG signal Y. Therefore, at $k = 0$, the system can be in one of the left idle states Ⓘ or in the first (left) active state. At $k = 1$, it can be in the second left idle states or in one of the first two active states, etc. The message update rules are derived and implemented in a straightforward fashion (in accordance with the sum-product rule) given this model. When calculating message μ_T in Fig. 4, μ_X is assumed to be uniform, which is equivalent to having no incoming message on the edge corresponding to X.

② **Coefficient nodes:** The boxes labeled ② in Fig. 2[1] represent the coefficient nodes (see also Fig. 5). They translate between state variables S and MUAP values Z. For this, we interpolate the MUAP waveforms over a fine grid of S using cubic splines. For easy numeric integration, we then assume that each segment of the MUAP between the "fine-grain" values (on the previously mentioned fine grid) of S can be approximated by a straight line segment.

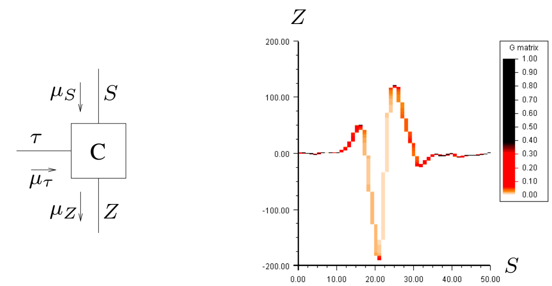

Fig. 5. Left: coefficient node, which corresponds to an FIR filter in the block diagram of Fig. 1; right: pre-calculated G matrix

Without interpolation, not all integer values Z in the interval from the lowest to the highest value in the interpolated MUAP will receive a probability mass. Therefore, this interpolation is especially important when dealing with non-integer shifts, which is equivalent to a change of the sampling phase τ.

The message update rule for μ_Z in Fig. 5 can be derived from the sum-product rule [4] assuming a uniform distribution of the phase variable τ, and by using a pre-calculated matrix G, which corresponds to $p(z|s)$. Fig. 5 shows the matrix G for the first MUAP in Fig. 8. The resulting update rule is

$$\mu_Z(z) \propto \sum_s G(z,s)\mu_S(s) \tag{2}$$

Note the similarity to

$$p(z) = \sum_s p(z,s) = \sum_s p(z|s)p(s). \tag{3}$$

The message update rule for $p_s(s)$ in upward direction is derived similarly.

③ **Addition nodes:** The equation $Z = X + Y$ describes the addition node in Fig. 6. Using the sum-product rule [4], μ_Z

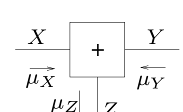

Fig. 6. Addition node

becomes a convolution of the two incoming messages:

$$\mu_Z(z) \propto \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} \delta(z-x-y)\cdot\mu_X(x)\cdot\mu_Y(y)\,\mathrm{d}x\,\mathrm{d}y \tag{4}$$

$$= \int_{-\infty}^{+\infty} \mu_X(z-y)\cdot\mu_Y(y)\,\mathrm{d}y. \tag{5}$$

[1] Here we omit an explanation of the corresponding nodes in Fig. 3.

Given joint Gaussian input messages $\mu_X(x) \propto \mathcal{N}(m_X, V_X)$ and $\mu_Y(y) \propto \mathcal{N}(m_Y, V_Y)$, where m_X and m_Y are mean vectors and V_X and V_Y are covariance matrices, the message update rule for Z becomes

$$\mu_Z(z) \propto \mathcal{N}(m_X + m_Y, V_X + V_Y). \tag{6}$$

Given discrete messages, the update rule is

$$\mu_Z(z) \propto \sum_y \mu_X(z-y) \cdot \mu_Y(y). \tag{7}$$

The update rules for X and Y are derived accordingly [4].

④ **Noise nodes:** For discrete messages, the message update rule for message μ_Z in Fig. 7 is:

$$\mu_Z(z) \propto \frac{1}{\sqrt{2\pi}\sigma} \cdot \exp\left(-\frac{(z-m)^2}{2\sigma^2}\right), \tag{8}$$

where m is a sample value of the EMG signal (given in μ_Y) and σ the standard deviation of the zero-mean additive white Gaussian noise. For the joint Gaussian case, the mean vector of the message is set to $(Y_{1,k}, Y_{2,k}, \ldots)$ and the covariance matrix is a diagonal matrix with the noise variances in the diagonal.

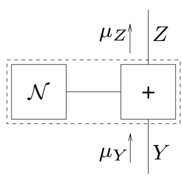

Fig. 7. Noise node

Message update schedule: The order of message calculations is defined by an update schedule. This is an important degree of freedom. In the algorithms presented here, messages are calculated first in forward direction (k increases), then in backward direction, and finally the marginal conditional probability distributions of the binary source signals X are approximated using the sum-product rule.

III. Results

We tested our algorithms with 100 simulated superpositions. The integer firing times were generated at random for each signal. The zero-mean additive white Gaussian noise had a standard deviation of 0.05 times the mean peak-peak amplitude of the noise-free MUAPs. Each superposition consisted of exactly four MUAPs. These MUAPs have similar shapes and they are also quite short compared to the MUAPs in our previous papers [2][3], which makes the decomposition problem more difficult. The four MUAPs and a sample superposition are shown in Fig. 8. Table I shows the decomposition results. The shapes of these MUAPs were given to the decomposition algorithms and it was assumed that each source fires exactly one time (known constituent problem).

IV. Discussion and Conclusion

In this paper we present novel approaches to EMG signal decomposition. In particular the joint-Gaussian message passing algorithm is described here for the first time.

The decomposition algorithm described in [5] is optimal in the sense that it finds the firing times that minimize the mean squared error between the reconstructed signal and the EMG signal. Thus it is able to decompose all of the 100 test signals correctly—even for the single channel case. In contrast, the message passing algorithms described in this paper are sub-optimal since the underlying factor graphs have loops. Loopy belief propagation algorithms are iterative algorithms, which might not converge to the correct solution.

For the 100 EMG signals decomposed for this paper, the algorithm in [5] is not only optimal, it is also faster than the new algorithms presented in this paper. However, the computational effort in the optimal algorithm from [5] increases exponentially with the length of the MUAPs and the number of MUAPs. In contrast, our new message passing algorithms show a roughly linearly growing computational demand with an increase in the number of sources, the number of channels, and the lengths of the MUAPs.

Given the inherent sub-optimality of our message-passing algorithms, the results are still promising. We also expect to obtain good results for the case of non-integer phase shifts, for the unknown constituent problem, and for signals with high levels of noise. The ability of our approach to decompose multiple channels (not only two, but any number of channels) simultaneously might yield good results in the case of multi-channel EMG signals recorded with surface electrode arrays.

V. Appendix: Java Package `factorgraph`

We developed a Java package called `factorgraph` to facilitate implementing message-passing algorithms that are based on factor graphs. This package is briefly described in this section.

One basic class is called `Node`. The package allows to define the factor graph topology by connecting nodes. Since every node has a node function, the second basic class is called `Function`. Its subclasses implement specific node functions for different message types, e.g., discrete messages, Gaussian messages, or joint-Gaussian messages. We have separate classes for nodes and its functions to separate the topology of the factor graph from the algorithm that runs on it. In this way we can easily change node functions without touching the topology of the factor graph. Nodes compute messages that are sent out of these nodes along the edges of the factor graph. Instances of the class `Schedule` define the order of

TABLE I

Decomposition results for 100 signals as in Fig. 8. The table shows the number of correctly decomposed signals and the average decomposition time in seconds per signal on a 2GHz Intel Pentium M processor.

	Correct	Time
Discrete messages, both channels	100	24
Discrete messages, only first channel	92	77
Discrete messages, only second channel	99	14.5
Joint-Gaussian messages, both channels	99	1.5

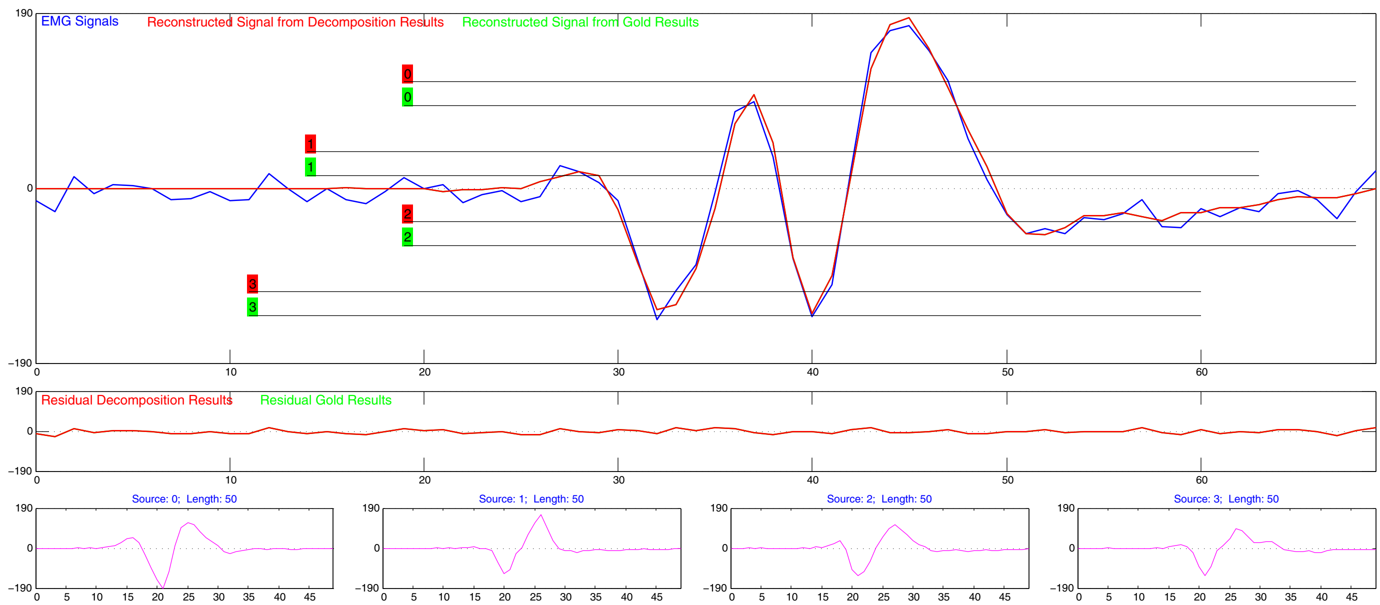

Fig. 8. Top: one channel of a two-channel simulated EMG signal as well as the reconstructed signal, which is based on the detected firing times; annotated with the detected (top) and gold (bottom) "firing times" and durations of the $N_{\mathrm{src}} = 4$ active sources; middle: difference signal between the EMG signal and the reconstructed signal; bottom: the four MUAPs used to simulate the 100 EMG signals. Remark: the reconstructed and residual signals based on the correct (=gold) firing times from the simulation (green) are hidden behind the corresponding signals for the detected firing times (red) since the firing times are identical

```
public void createGraph(){

    // Instantiate nodes
    nodeAdd       = new Node("nodeAdd", functionAdd);
    nodeTerminal0  = new Node("nodeTerminal0", functionTerminal0);
    nodeTerminal1  = new Node("nodeTerminal1", functionTerminal1);
    nodeTerminal2  = new Node("nodeTerminal2", functionTerminal2);

    // Establish connections between nodes
    Node.connect(nodeTerminal0, 0, nodeAdd, 0);
    Node.connect(nodeTerminal1, 0, nodeAdd, 1);
    Node.connect(nodeTerminal2, 0, nodeAdd, 2);

    // Define slice port numbers
    port = new Port[numberOfPortsMax];
    addPort(nodeAdd,0);
    addPort(nodeAdd,1);
    addPort(nodeAdd,2);

    // Define update schedule
    schedule0 = new Schedule(3,port,numberOfPorts);
    schedule0.addPortToSchedule(0);
    schedule0.addPortToSchedule(1);
    schedule0.addPortToSchedule(2);
}
```

Fig. 9. Source code that shows how a simple factor graph (with one add node and its terminal nodes) as well as a message update schedule is defined using the package `factorgraph`

message computations. To define such a schedule, the user assigns a unique number to each port in the factor graph. The class `Port` is used to do this. Then the schedule is just a sequence of port numbers. Fig. 9 contains a code snipped that shows how a very simple factor graph as well as an update schedule is defined.

The class `Message` deals with storing and processing various kinds of messages. The Java package is very flexible in that it can easily be extended to deal with arbitrary message formats.

When designing factor graphs for signal processing applications, like the ones in this paper, there are often many identical slices in the factor graph, one for each discrete time index. After implementing only one such slice in a subclass of the class called `FgOneTimeSlotAbstract`, objects of this class can then be instantiated for each slice. The single sections are connected to make up the full factor graph.

The Java package `factorgraph` is a general package. It provides building blocks that algorithm designers can use without having to derive every detail by themselves first (modular assembly principle). This helps researchers and developers to implement various signal processing or machine learning algorithms in a more or less straightforward way.

In summary, the `factorgraph` package is an object oriented, modular, open, reusable, and cross-platform toolbox. It facilitates the implementation of factor graphs and message-passing algorithms that run on such factor graphs. Its building blocks can simply be combined, which allows rapid prototyping. In addition, new building blocks can easily be included.

VI. Acknowledgement

This work was supported in part by the US National Institute of Neurological Disorders and Stroke (1 R01 NS051507).

References

[1] D. Zennaro, P. Wellig, V. M. Koch, G. S. Moschytz, and Th. Läubli, "A software package for the decomposition of long-term multichannel EMG signals using wavelet coefficients," *IEEE Trans. on Biomedical Engineering*, vol. 50, no. 1, pp. 58–69, January 2003.

[2] V. M. Koch and H.-A. Loeliger, "Decomposition of electromyographic signals by iterative message passing in a graphical model," in *26th Annual International Conference Engineering in Medicine and Biology Society (EMBC 2004)*, San Francisco, California, U.S.A., September 1-5 2004.

[3] V. M. Koch and H.-A. Loeliger, "EMG signal decomposition by loopy belief propagation," in *30th IEEE International Conference on Acoustics, Speech, and Signal Processing (IEEE ICASSP 2005)*, Philadelphia, PA, U.S.A., March 18-23 2005, pp. 397–400.

[4] H.-A. Loeliger, "An introduction to factor graphs," *IEEE Signal Proc. Mag.*, vol. 21, no. 1, pp. 28–41, January 2004.

[5] K. C. McGill, "Optimal resolution of superimposed action potentials," *IEEE Trans. on Biomedical Engineering*, vol. 49, no. 8, pp. 640–650, July 2002.