

1. Implemented algorithm

The algorithm implemented in Matlab to remove superficial contributions from fNIRS data has been inspired by the publications of Saager and Berger, 2005 [1] and Scholkmann et al., 2014 [2].

[1]: <https://www.ncbi.nlm.nih.gov/pubmed/16211814>

[2]: <https://www.ncbi.nlm.nih.gov/pubmed/24622337>

Firstly, channels provided are identified as short- or long-channels according to the 3D inter-optode Euclidean distance. The user is expected to provide a threshold (in 'mm') so the script can identify automatically the short-channels (e.g. 10 mm).

Following that, the script estimates the position of each channel by computing the arithmetic mean of the respective source and detector. This information is later on used to automatically detect, for each long-channel, which is the closest short-channel to be used for its data correction.

The correction is therefore applied on a channel basis (i.e. individually for each long-channel) by computing two scalar products: (a) $\text{dot}(AS, AL)$, and (b) $\text{dot}(AS, AS)$; in which 'AS' stands for the Optical Density of the short-channel and 'AL' the Optical Density of the long-channel.

The corrected data is thus computed as follows: $AC = AL - \text{alfa} \cdot AS$, where $\text{alfa} = (a) / (b)$.

Please note that this short-channels correction was originally suggested to be applied on individual blocks of responses, i.e. separately for each trial of each condition of a given experimental design.

However, by doing so, the data structure might be affected, as individual block responses may end up with different baselines, and further statistical analysis with General Linear Model would not be possible.

Therefore, the implemented script provides a flexibility for the user to choose how the correction should be applied: either as the original on individual responses blocks, or rather by considering the whole time series of each channel as a whole. This can be set with the input argument 'task', whose values are '1' and '0', respectively.

Further input arguments concern the length of the window to be considered for the correction. Similar as for a block average calculation, the user may provide the baseline ('blen') as the period prior trial onset, as well as, the whole duration of the trial and rest ('offset'). Both are in seconds.

The algorithm has been implemented in two different versions, each of them to work exclusively with a major fNIRS analysis tool, namely Homer and nirs-toolbox (AnalyzeIR). Please choose the appropriate version depending on the intended tool for your data analysis. The following sections outline the most relevant steps to use the algorithm in each of them.

2. Use in Homer (v2_3_10202017)

In addition to the script **hmrSSR.m**, which performs the short-separation regression as described in section (1), two further scripts are provided: **procStreamReg.m** and **procStreamRegHelp.m**

These scripts are originally available in Homer2.3 and have been slightly modified to allow for proper adjustment of the processing pipeline in Homer, as well as to display its description.

Therefore, following steps need to be performed before launching Homer:

1) Include **hmrSSR.m** in the root of Homer2.3:

...\homer2_src_v2_3_10202017\hmrSSR.m

2) Replace **procStreamReg.m**, which is located under following path:

...\homer2_src_v2_3_10202017\PACKAGES\EasyNIRS\Processing\procStreamReg.m

3) Replace **procStreamRegHelp.m** in the same path:

..\homer2_src_v2_3_10202017\PACKAGES\EasyNIRS\Processing\procStreamRegHelp.m

Then, please launch Matlab and set the path to all folders and subfolders of Homer (i.e. usual setup of Matlab to work with the development version of Homer).

You may then launch Homer by typing **Homer2_UI** in the Matlab command window.

Now, you will need to set up the processing stream to add hmrSSR. On the top menu bar, please access Tools → Process Stream GUI. This will launch the procStreamGUI window.

An example of reasonable processing pipeline is shown on Figure 1.

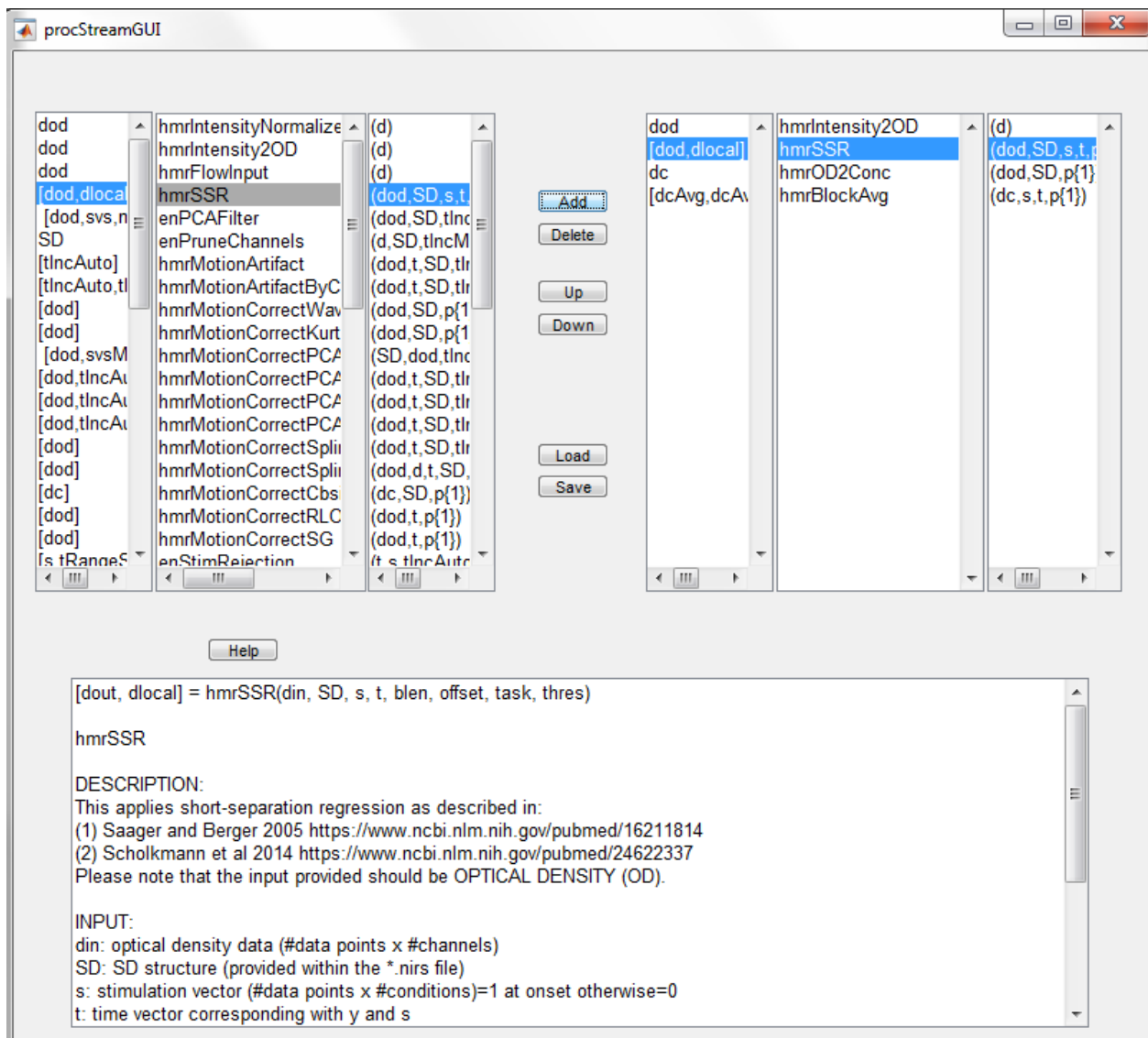
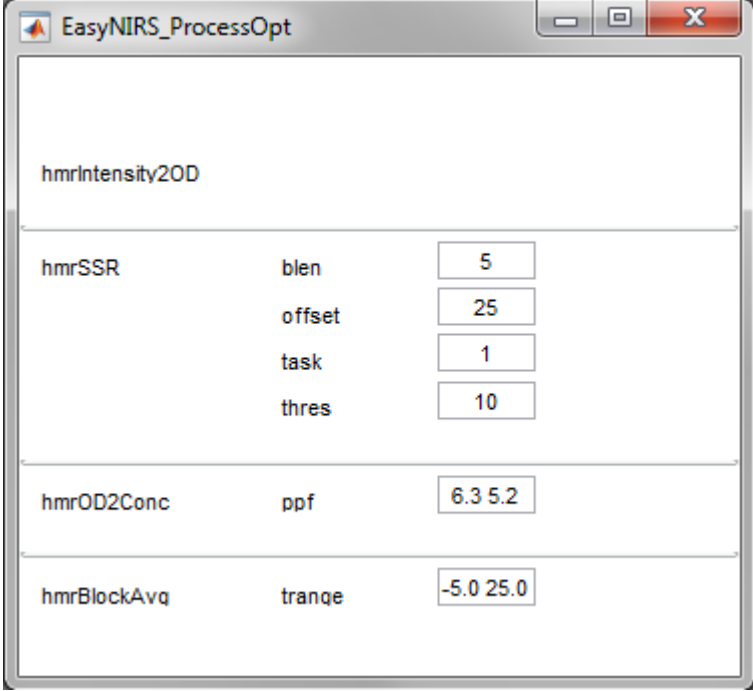


Figure 1: Example of processing pipeline with short-channels correction as set up in the procStreamGUI.

As explained in section 1, the algorithm is supposed to work on **optical density** data. Therefore, the pipeline first converts the raw data intensity to optical density. It is then followed by the short-separation correction and the conversion to concentration changes (hmrSSR and hmrOD2Conc). Finally, we also include hmrBlockAvg to allow for easy-to-interpret block average visualization.

Once the pipeline has been defined, please click on “Save” and add it to the current processing stream. On the Homer main window, please now click on “Options” to set up the parameters for the processing stream.

Figure 2 shows an example of parameters setup for a finger tapping dataset based on block average design with 15 s activation and 10 s rest.



The screenshot shows a window titled "EasyNIRS_ProcessOpt" with a scrollable list of parameters. The parameters are organized into sections: "hmrIntensity2OD", "hmrSSR", "hmrOD2Conc", and "hmrBlockAvg". Each section contains a parameter name, a description, and a value in a text box.

Parameter	Description	Value
hmrIntensity2OD		
hmrSSR	blen	5
	offset	25
	task	1
	thres	10
hmrOD2Conc		
	ppf	6.3 5.2
hmrBlockAvg		
	tranqe	-5.0 25.0

Figure 2: Example of parameters setup with short-channels correction and block average visualization.

The baseline prior onset (blen) has been set to 5 s and the window length has been set to 25 s (sum of activation and rest period). We have set up 'task' to 1, as we would like to apply the correction for each block of response individually. Finally, the threshold for inter-optode distance to automatically detect short-distance channels (thres) has been set to 10 mm.

Once the parameters have been set up as desired, please close the EasyNIRS_ProcessOpt window and click on “Run” in the Homer main window. This will process the data and generate the results.

Upon processing completion, one may access the block average results by selecting the desired channels on the 2D layout, and then, in the “Plot” menu, choose “Conc” (concentration changes), “HbO” and “HbR” (data to be plotted) and “show Run HRF” (to display the block average). You may also select the trigger marker of interest (e.g. 1 or 15) from the drop-down menu.

To illustrate the potential of the short-distance correction, displayed below is an example of left-hand finger tapping responses (HbO and HbR) for processing pipelines with and without SSR.

The data illustrate the responses from each hemisphere. The green curve (right hemisphere) presents a more prominent hemodynamic response for the left-hand finger tapping in the case with SSR.

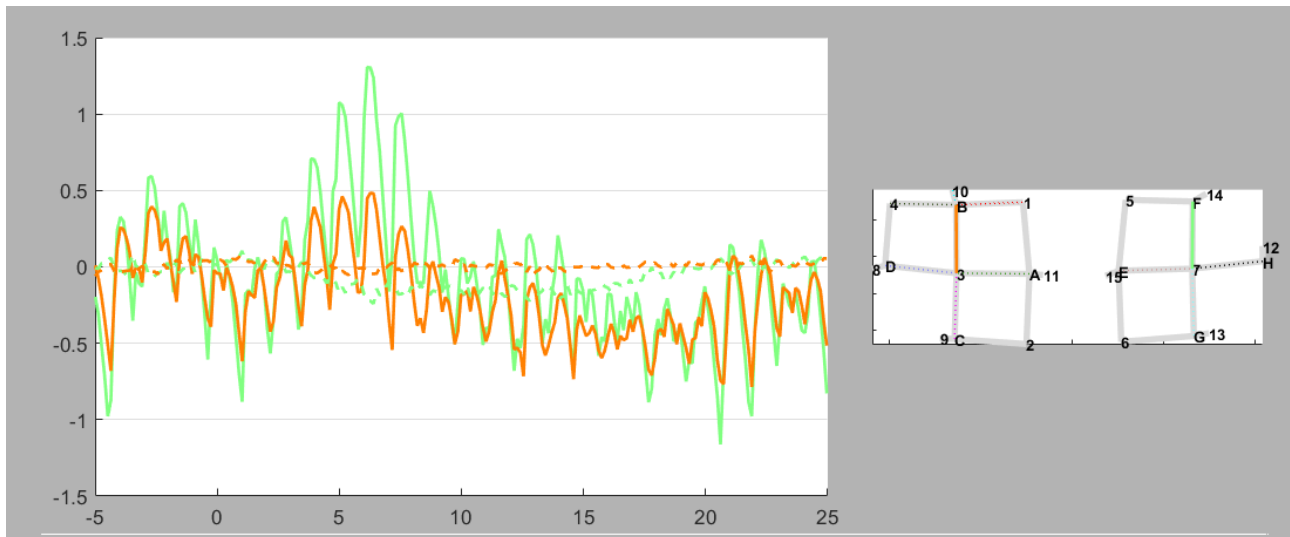


Figure 3: **Left**-finger tapping responses on left (orange) and right (green) hemisphere **without** SSR.

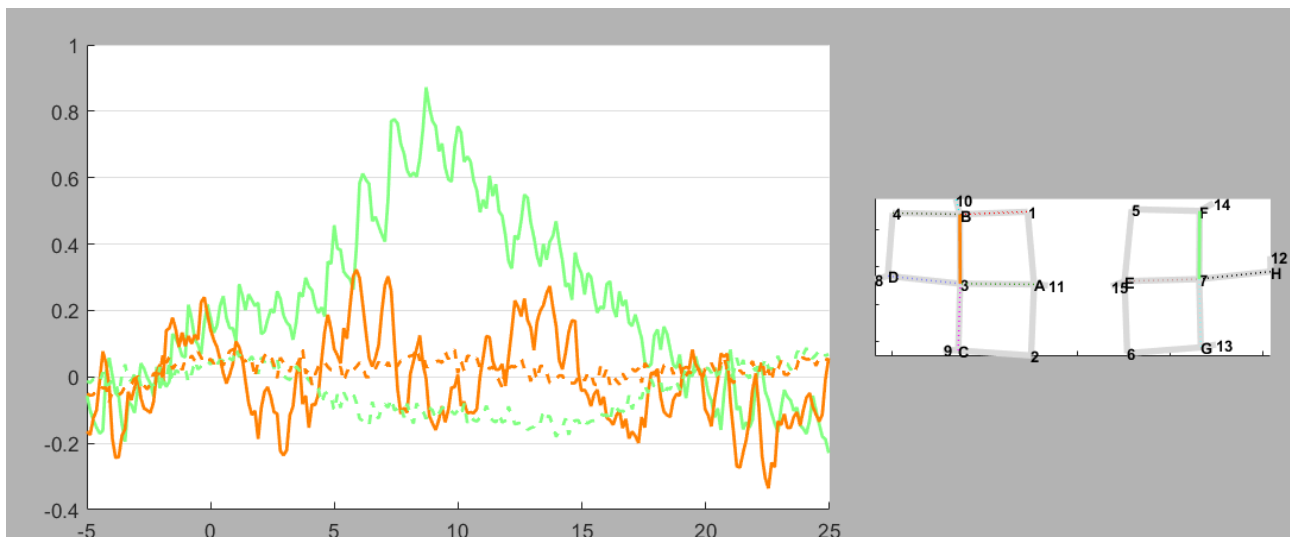


Figure 4: **Left**-finger tapping responses on left (orange) and right (green) hemisphere **with** SSR.

3. Use in nirs-toolbox (dev version: 2018-11-01)

Three scripts are provided for the nirs-toolbox, as outlined below for following purposes:

- 1) **ntbxSSR.m** → the implemented script, to be placed in: `\nirs-toolbox\+nirs\+modules`
- 2) **vector2event.m** → to replace the original: `\nirs-toolbox\+nirs\+design\vector2event.m`
- 3) **pipeline_ntbx.m** → example of command lines to process data and visualize results

In particular, (2) will be necessary in case you are loading a file with *.nirs extension (e.g. data recorded with NSP2). It comments out the estimation of responses duration, and rather initializes all entries with value '1'. This is a similar procedure that takes place for other NIRx files (loadNIRx.m). We will rather use another function available to set up the stimulus duration.

The pipeline_ntbx presents a set of commands that can be called in different blocks. Most of them use functions already available within the development version of nirs-toolbox, as downloaded in November 1st, 2018. Only (1) needs to be added to enable the short-distance correction.

Below we break down each block of commands and provide a brief explanation:

(A) Firstly, the user shall be prompted to select the *.hdr or *.nirs file corresponding to the dataset to be analyzed. Please note that this is optional, as one may also type in this information, or rather provide the path to a folder with several datasets to be analyzed at once. In this tutorial, for the sake of simplicity, we rather go through the single-dataset scenario.

```
%%% Prompt to select path and file
[filename, pathname] = uigetfile({'*.hdr'; '*.nirs'}, 'Please select the .nirs
or .hdr file of your experiment.');
```

(B) Once the data_path has been identified, the program identifies the available file extension to call the respective function. Note that nirs-toolbox provides several different functions to load different types of data. If you would like to load other data types, please check the folder `\+nirs\+io`

```
%% Load Data (individual dataset)
[~,~,ext] = fileparts(data_path);
if strcmp(ext, '.nirs')
    raw = nirs.io.loadDotNirs(data_path);
elseif strcmp(ext, '.hdr')
    raw = nirs.io.loadNIRx(data_path, false);
else
    display(['Unexpected file extension: ' ext])
end
```

(C) As previously explained, there is an specific call to edit the stimulus duration for each condition. This can be provided as an array whose length should match the number of trigger markers. Below is an example of two conditions (two markers) with 12 s of response time each. Please make sure to adapt 'stim_dur' according to your experimental design.

```
%% Edit stimulus duration
stim_dur = [12 12]; %array with duration of each condition
raw_edit = nirs.design.change_stimulus_duration(raw, [], stim_dur);
```

(D) You may also rename the conditions to facilitate data interpretation later on. In the example below, the conditions had been automatically named by the nirs-toolbox as 'stim_channelX'. In case your dataset presented other names after loading, please make sure to adapt the fields on the left. Those on the right represent the resulting names.

```
%% Rename stimulus
job = nirs.modules.RenameStims();
job.listOfChanges = {
    'stim_channel1', 'left_tap';
    'stim_channel2', 'right_tap';
};
raw_edit = job.run(raw_edit);
```

(E) Now we proceed with the first processing step. As explained before, the implemented algorithm expects optical density as input data. Thus, please make sure to convert the data accordingly.

```
%% Compute Optical Density
job = nirs.modules.OpticalDensity();
OD = job.run(raw_edit);
```

(F) Once optical density data are available, we provide these as input to our SSR function. Please note that the call consists of “nirs.modules.ntbxSSR”, because it is supposed to be located under \+nirs\+modules. The first argument expected is the Data class of nirs-toolbox with optical density data. The following arguments are baseline, offset (in addition to duration) and the 'task' flag. The latter is herein set to '0' to allow for further General Linear Model computation. This means that the whole time series of each channel will be corrected at once, instead of applying the correction for each block of response separately. This is an alternative to the original method description.

```
%% Short-channels correction
blen = 5; % baseline (pre onset)
offset = 5; % offset (post duration)
task = 0; % flag for separate blocks
OD_edit = nirs.modules.ntbxSSR(OD, blen, offset, task);
```

(G) As the data have been previously converted to optical density, it is still necessary to compute the concentration changes before moving to the statistics. Use the BeerLambertLaw module for it.

```
%% Compute Concentration Changes
job = nirs.modules.BeerLambertLaw(); % it uses PPF = 0.1
hb = job.run(OD_edit); % unit: uM (i.e. 10-6 mmol/L)
```

(H) Now, we make use of the AR-IRLS module available in the toolbox for the GLM. Essentially, the algorithm provides a flexible order for the autoregressive model to better adapt to the features of fNIRS data (e.g. autocorrelation with higher sampling rate). Literature reference can be found here: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3756568/>

```
%% Individual Stats: AR-IRLS
job = nirs.modules.AR_IRLS();
job.verbose = true;
job.basis = Dictionary();
job.basis('default') = nirs.design.basis.Canonical();
SubjStats=job.run(hb);
```

(I) Finally, we may visualize the results of a t-test based on the outputs of the AR-IRLS. We create a simple contrast to assess the different of right- and left-hand finger tapping and assess as a t-test. Using the draw() method, we also specify a threshold of q-values to be lower than 0.05.

```
%% Visualize Results
c = [-1 1]; % contrast: right - left
ContrastSubj = SubjStats.ttest(c); % compute t-test
ContrastSubj.draw('tstat', [-10 10], 'q < 0.05') % apply threshold
```

Following the pipeline described above with a finger-tapping dataset collected with 16 sources and 30 detectors (16 of which have been used for short-channels), we illustrate below the statistical results of HbO and HbR data as obtained for two cases: (a) without SSR; and (b) with SSR.

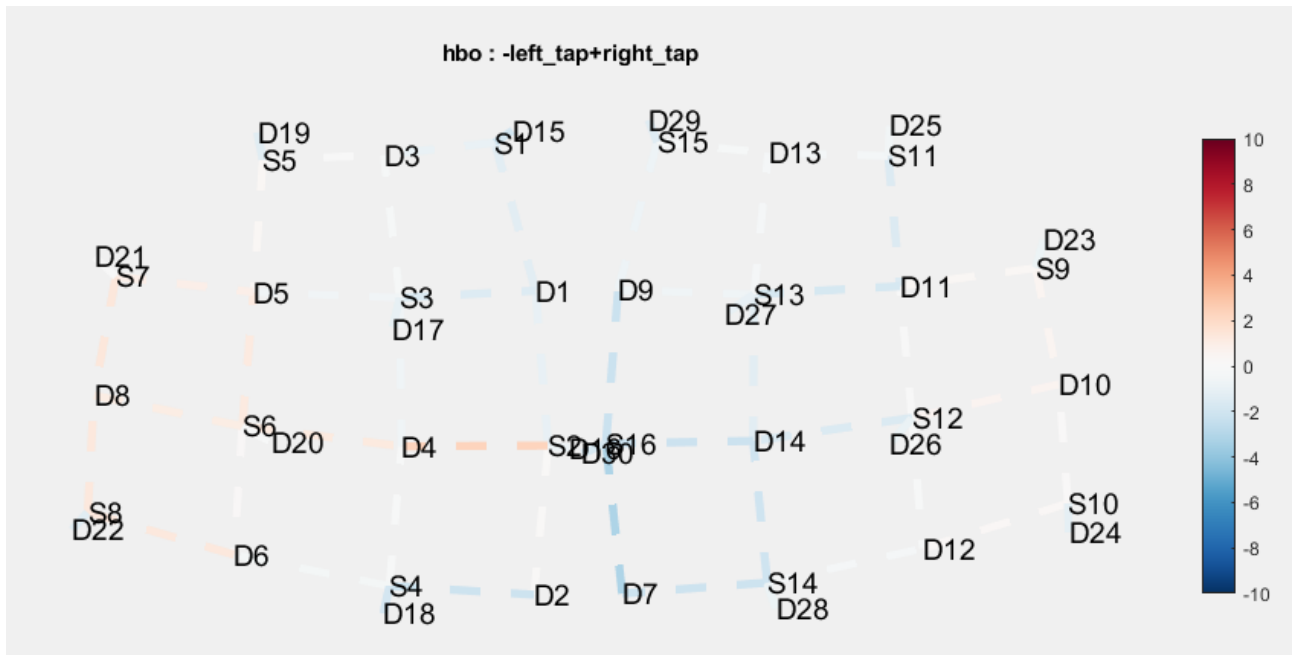


Figure 5: HbO results with a threshold of $(q < 0.05)$ for $(\text{right_tap} - \text{left_tap})$ **without** SSR.

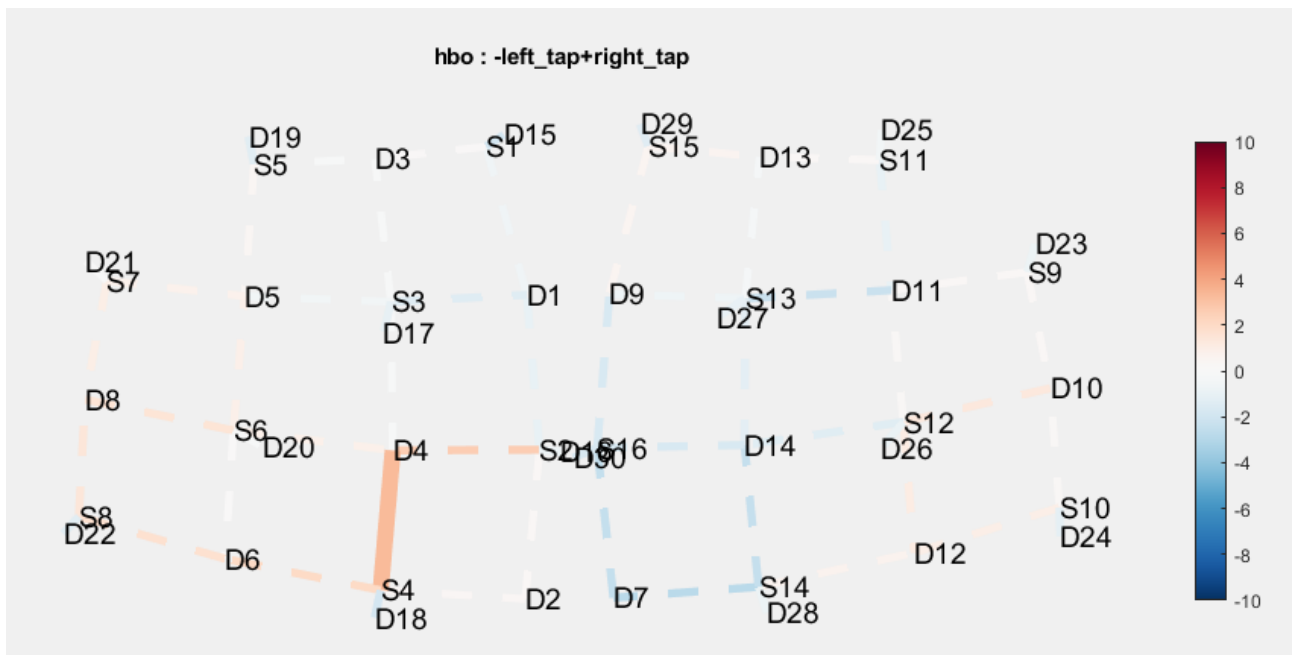
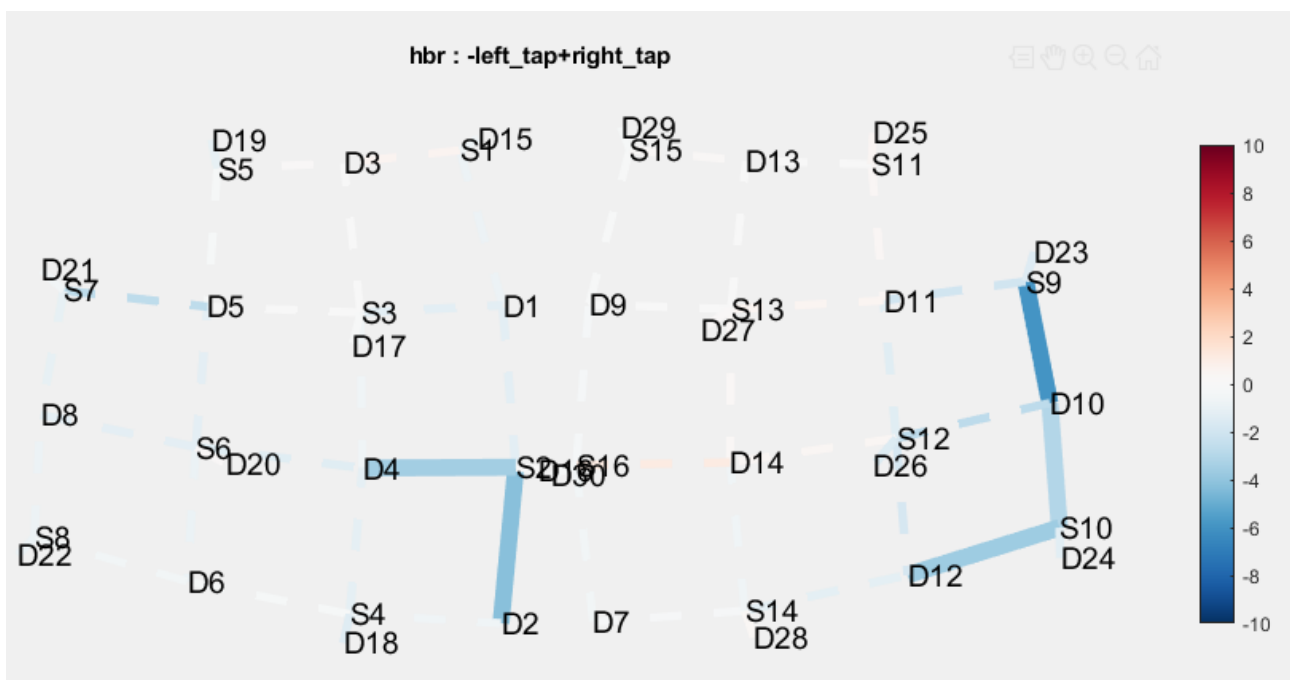
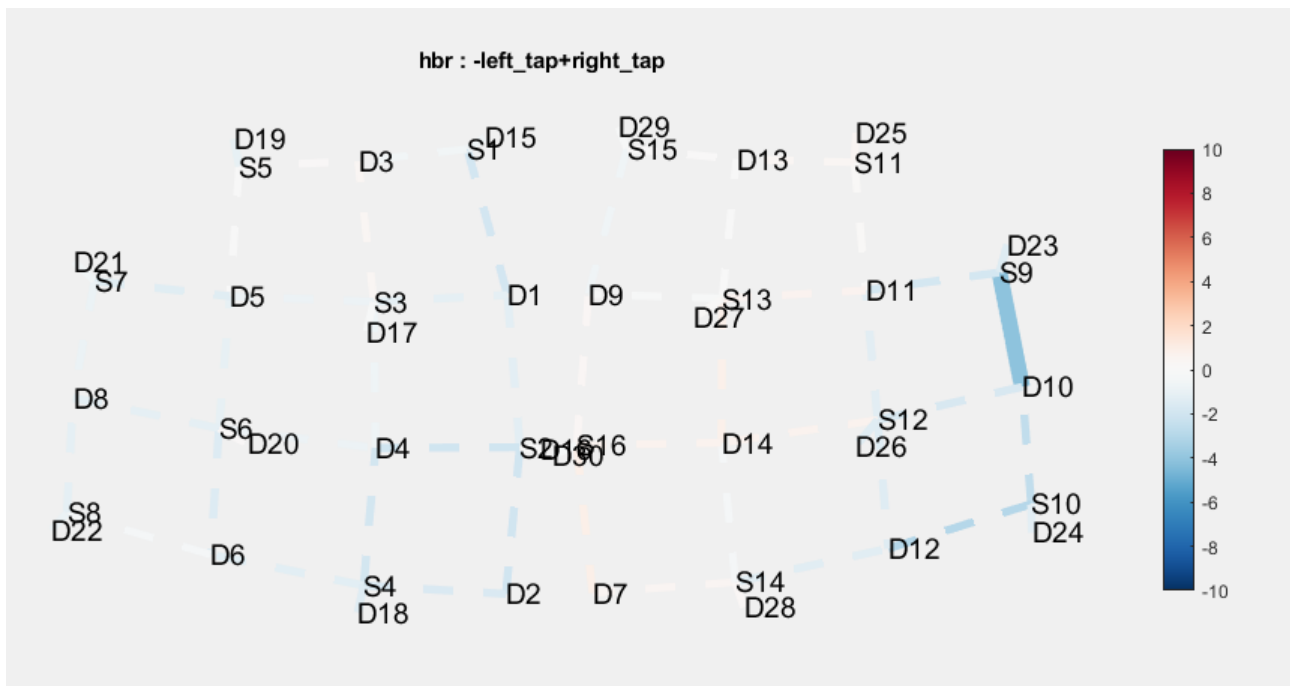


Figure 6: HbO results with a threshold of $(q < 0.05)$ for $(\text{right_tap} - \text{left_tap})$ **with** SSR.



- Please note that HbR results are expected to be negative, representing a decrease during activation.