utilities

Russell O'Connor

November 13, 2015

1 crossings2edt

The `crossings2edt` program generates an `.edt` file from a `.chan` file, based on positive-going crossings of a specified threshold. This can be useful if there is one large, clean unit on the channel. One spike is placed in the `.edt` file for each place in the `.chan` file where there is a sample below threshold followed by zero or more samples at threshold, followed by a sample above threshold. The spike time is the time of the above-threshold sample rounded to the nearest .1 ms. Running the program with no arguments gives this usage message:

```
usage: crossings2edt whatever.chan threshold unit_code
```

`whatever.chan` is the input `.chan` file. The output `.edt` file will have the same base name, but with an `.edt` extension. Any existing file by that name will be overwritten without warning. `threshold` is a number between -32768 and 32767 chosen by the user, typically by reading it off the waveform display (using `waveform.tcl`) as the fifth number displayed at the top, when the cursor is positioned at what looks like an appropriate threshold level. `unit_code` is the cell ID that will be used for the spike train in the `.edt` file.

2 crossings2pos

The `crossings2pos` program generates a `.pos` file with spike times for use with the spikesorter, from a `.chan` file, based on positive-going crossings of a specified threshold. This can be useful if there is one large, clean unit on the channel. Running it with no arguments gives this usage message:

```
usage: crossings2pos [-m THR2] POS CHAN THR CLUSTER... > OUTPOS
```

The program is intended to be used with a `POS` file that has been generated by the spikesorter from `CHAN`, but for which simple threshold crossings would have done a better job. `THR` is a number between -32768 and 32767 chosen by the user, typically by reading it off the waveform display (using `waveform.tcl`) as the fifth number displayed at the top, when the cursor is positioned at what looks like an appropriate threshold level. `CLUSTER...` is a space-separated list of cluster numbers in `POS` that will be left out of `OUTPOS`. The first number in the list will be re-used in `OUTPOS` as the cluster number for the threshold-crossing spikes. If the old `.pos` file is deleted or renamed or moved, the new `.pos` file can be given the name of the old one, and then it can be viewed in the spikesorter. The waveform overlay for the new cluster will be wrong, as will scatterplots that include it, but everything else should work.

If a second threshold is specified with the `-t` option, a spike will not be placed at a positive-going crossing of the first threshold unless the signal in CHAN goes back below the first threshold without going above the second threshold.

An example session might look like this:

```
cd 2004-01-25_001
crossings2pos 2004-01-08_001_11.pos 2004-01-08_001_11.chan 28000 1 2 3 4 5 > new.pos
mkdir -p save
mv -i 2004-01-08_001_11.pos save
mv new.pos 2004-01-08_001_11.pos
```

3 dmx

The `dmx` program is used by `spikesort_control_panel` to determine the channel labels and which ones are enabled, but it can also be used by itself on the command line.

Invoked as

```
dmx DMXFILE > OUTFILE
```

where DMXFILE is a DataMAX header or recording file, it will translate the header to text and write it to OUTFILE.

Invoked as

```
dmx DMXFILE labels
```

it will print a list of the channel labels to the screen.

Invoked as

```
dmx DMXFILE enabled
```

it will print a list of zeroes and ones to the screen, indicating which channels are enabled (1 = enabled).

If DMXFILE is a recordong file, invoking it as

```
dmx DMXFILE N
```

will extract channel N (the first channel is channel 1) from DMXFILE and write it to a `.chan` file in the current directory. The name of the `.chan` file will be the

same as DMXFILE with any extension deleted, and with a two-digit channel number and the `.chan` extension appended. Andy existing file by that name will be overwritten without warning.

4 dmx_split

The `dmx_split` programs splits a DataMAX recording into `.chan` files. When invoked without arguments, it gives this usage message:

```
usage: dmx_split dmx_file_name
```

`dmx_split` creates two directories named `clean` and `split` in the current directory if they don't already exist. For each enabled channel, it will create a `.chan` file in the `split` directory. The name of the `.chan` file will be the same as `dmx_file_name` with periods changed to underscores, and with an underscore, a two-digit channel number, and the `.chan` extension appended. The first channel is channel 1 (not 0). If any of the `.chan` files already exist, `dmx_split` will ask once whether to overwrite them. If the answer is no, it will exit immediately and no `.chan` files will have been written.

It will also create a header file with the same name as `dmx_file_name`, but in the `clean` directory, containing just the header from `dmx_file_name`. If the file already exists, it will be overwritten without warning. If you answered no to overwriting `.chan` files as described above, the header file will still be overwritten if it exists, but the new file will be empty.

5 info_file.pl

The `info_file.pl` script reads a coordinate spreadsheet file and a `.nam` file and writes an "info" spreadsheet file. It is normally invoked automatically by `spikesort_control_panel` after the user clicks on the MERGE button, so the user doesn't need to know about it, but it can be invoked by itself from the command line if the user wants to recreate the info file without rerunning the merge. When invoked without arguments, it gives this usage message:

```
usage: info_file.pl datadir prefix

info_file.pl will look for a coordinate spreadsheet file at /oberon/experiments/prefix/prefix.xls
info_file.pl will look for a .nam file at datadir/prefix.nam
info_file.pl will  write  an info file at /oberon/experiments/prefix/prefix_info_orig.xls
```

The coordinate spreadsheet file must be in Excel 95-2003 format, and the output info file will be as well. The coordinate spreadsheet is created by hand. The `.nam` file is generated by `spikesort_control_panel` after the user clicks the MERGE button. (`spikesort_control_panel` calls `merge_edt` which calls `info_file.pl`.)

The `.nam` file is an ASCII text file with Linux-format line endings (line feed). It has one line per cell, with three items per line:

1. the digital channel number
2. the cell name assigned by `edt_merge`
3. the merged channel number assigned by `edt_merge`

`info_file.pl` requires the coordinate spreadsheet to have the following exact text in the indicated cells:

Cell C11: DIGITAL CHANNEL
Cell I11: A/P
Cell J11: R/L
Cell K11: DEPTH
Cell A3: EXPERIMENT:
Cell A5: RECORDING:
Cell A7: DATE:

There can be a second column of data with the same headers in the same rows, but starting in column P. If there is no second column, the header cells must be empty.

The data must start in row 12, and the background of the data cells must be colored. The row following the data must not be colored.

`info_file.pl` also requires the following information in the indicated cells:

Cell D48: the digital channel number of the reference electrode
Cell D50: the A/P coordinate of the reference electrode
Cell D51: the R/L coordinate of the reference electrode
Cell D52: the depth coordinate of the reference electrode

At USF, the coordinate spreadsheet is generated from a template in the Excel workbook found from Windows at `\\cisc3\experiments\templates\coordinate_template_workbook.xls` (or at

/oberon/experiments/templates/coordinate_template_workbook.xls from Linux). Instructions can be found at \\cisc3\experiments\templates\coordinate_template_instructions.doc.

The output info file for each experiment contains the name and date of the experiment; the recording number; the names, channel numbers, and coordinates of each cell; and places for the results of AA and STA testing. The data in the info file is read into `xanalysis` for display and, from there, is written out for import into a database (at USF, that's the GAIA database).

`info_file.pl` makes two copies of the output file, one with a name of the form *prefix*`_info_orig.xls` as in the usage message, and the other with a name of the form *prefix*`_info_curr.xls`. If *prefix*`_info_orig.xls` already exists, it is overwritten without warning. If *prefix*`_info_curr.xls` already exists, a warning is written to the command line and the old version is backed up to a file with a numbered extension, starting with *prefix*`_info_curr.xls.~1~`. No existing backups are overwritten or deleted. If AA or STA information was written to the old *prefix*`_info_curr.xls`, it is copied to both new files.

The two arguments to `info_file.pl` tell it where to find the input files and where to put the output file as shown in the usage message above. (The `/oberon/experiments` directory shown in the usage message is the path as seen from Linux. As seen from the Windows systems at USF, the same directory is `\\cisc3\experiments`.) It does not matter what the current working directory is when `info_file.pl` is invoked.

If the user wants to run `info_file.pl` without writing the output to the official directory, the user can specify a different directory as a third argument, and the output file will be written there instead.

6 integrate

The `integrate` program is used by `spikesort_control_panel` to process channels for which "I" is specified, but it can also be used by itself on the command line. Invoke it as

```
usage: integrate FILE.chan ANALOG_ID [CUTCODE CUT_EDT [flip]]
```

to rectify and integrate the signal in `FILE.chan` and write the result to `FILE.edt` with the specified `ANALOG_ID`. Also writes `FILE.bin`, which is a big-endian copy of `FILE.chan`. If `FILE.edt` or `FILE.bin` exists, it will be overwritten. Also appends an `I` to `FILE.status`.

If `CUTCODE` and `CUT_EDT` are specified, regions will be left out of the integrated signal in `FILE.edt`. The regions to be left out are specified by an event code in

an `.edt` file. `CUTCODE` is the id of the event code, and `CUT_EDT` is the name of the `.edt` file. The region between the first and second event of the given id is left out, and between the third and fourth, etc. In general, the region between each odd numbered event (start marker) and the following even numbered event (stop marker) is left out. If "flip" is specifed, the regions that would have been left out are kept, and vice-versa. The content of the left-out regions has no effect on the integrated signal.

Before each kept region, the integrator is initialized with the mean of the integrated signal, so there is little or no start-up ramp.

The samples of the integrated signal are 5 milliseconds apart, and the sampling clock continues to run during the left-out regions, so all intervals between samples are a multiple of 5 ms.

7 split_merge_abf

The `split_merge_abf` programs merges `.abf` files and then splits them into `.chan` files. When invoked without arguments, it gives this usage message:

```
usage: split_merge_abf whatever.abf [channel]...
```

The `whatever.abf` file must end in a `.abf` extension, but the `.abf` can be upper or lower case (or even mixed). The file is assumed to have a 2048 byte header, which is ignored, followed by consecutive interleaved 16-bit samples from 16 channels in channel order 1,9,5,13,2,10,6,14,3,11,7,15,4,12,8,16. There must be a multiple of 16 samples in the file. After processing the first `.abf` file it looks for another file with the same name except with an additional upper or lower case 'A' before the `.abf` extension (if there are both, it uses the one with the upper case). This file must also have a 2048 byte header, the same channel order, and a multiple of 16 samples, and the samples from this file are appended to those from the first file. The program then increments the ASCII value of the extra letter and looks for another file in the same way. This continues untils it fails to find the next file.

If no channel numbers are specified, all 16 channels will be extracted to `.chan` files. The file names will be the same as `whatever.abf`, except with the `.abf` extension replaced with a two-digit channel number followed by a `.chan` extension. Any existing files with the same names will be overwritten without warning. If channel numbers are specfied (in a space-separated list), only those channels will be extracted

8 tkss.tcl

Using `tkss.tcl`, the spike sorter can be run without using the `spikesort_control_panel`, simply by typing

```
tkss.tcl
```

at the command line. This brings up what the Spike Sorter User Guide call the "Spike Sorter Window". It will not dispatch spike sorts to other computers like the control panel does, but you can run multiple spikesorts in parallel on the same computer using the SPIKESORT button to start them one at a time. They will continue to run after you quit. You can kill all the spike sorts you are running by typing

```
killall spikesort
```

at the command line.

You can also split a DataMAX file or an `.abf` file into `chan` files using the SPLIT button. `tkss.tcl` will not show you raw data, but you can use `waveform.tcl` for that. It will not show you the cluster "diagram" either, but you can get that by typing

```
dot -Tps FILENAME.dot > FILENAME.ps;  gv -scale=2 FILENAME.ps
```

And it will not bring up the `.notes` file, but you can do that by typing

```
gedit FILENAME.notes
```

It will not invoke `integrate, digitize, rpls, cpls, trachpls, muph`, or `chan2hdt` for you, but those can all be run by hand. And it will not merge the `.edt` files for multiple channels into a single `.edt` file. You would have to do that by changing unit numbers with `tmover` or `sed` (figuring out yourself what they should be), and by merging the resulting `.edt` files with the `merge` utility.

But it does have all the functionality described under "*RESULTS*" in the Spike Sorter User Guide.