VLF Receiver Toolkit - Notes

For version 0.9j of vlfrx-tools

Index

Installation

Ubuntu

Installation is very easy on Ubuntu/Debian systems. Just run the following apt-get commands (prefix with sudo if you haven't enabled the root login):

```
apt-get install libasound2-dev
apt-get install libvorbis-dev
apt-get install libflac-dev
apt-get install libx11-dev
apt-get install libpng12-dev
apt-get install libxpm-dev
apt-get install libncurses5-dev
apt-get install libforms2
apt-get install libforms-dev
apt-get install libshout3-dev
apt-get install libsamplerate0-dev
apt-get install libfftw3-dev
apt-get install sox
apt-get install gnuplot
```

That takes care of all the packages that the toolkit depends on. Then install the toolkit with the instructions below.

If you're not running Ubuntu or some other flavour of Linux that uses apt-get, then see the section Prerequisite Packages below.

Raspberry Pi

Using Raspberry Pi for audio A/D with a USB sound dongle is not recommended due to fundamental and apparently unsolvable problems with the USB sub-system. USB isochronous data packets are silently and frequently dropped (even when the CPU is lightly loaded) and this renders the USB device useless for reliable signal capture.

Currently (2018) the best option for capture on the Raspberry Pi are the 'Audio Injector' (stereo) and 'Octo' (6 input channels) audio interfaces made by Flatmax Studios. These cards use I2S/DMA for data transfer. They still produce the occasional read error (which vtcard recovers from) but are quite usable - certainly far better than USB.

Installation is straightforward on the RPi. If running the Raspbian operating system, use the apt-get commands listed for Ubuntu above. Then continue with the installation instructions below.

Installation

Download the tgz file and unpack. cd down into the package directory and run

```
./configure
```

Configure will report if any prerequisite packages are missing. Some of these can be ignored given suitable options to `configure`.

On the Raspberry Pi, if you are using Raspbian or Archlinux you should enable the use of hardware floating point, configure with

```
./configure --with-hardfloat
```

The command `./configure --help` will show all the configuration options. When configuration is successful, continue with the installation,

```
make
make install
```

Programs will be installed in /usr/local/bin.

If you get a make error on OpenBSD, you may need to do

```
LDFLAGS="-lpthread -lspeex -lz -lossaudio"
export LDFLAGS
```

and then repeat the configure and make install.

If make reports an undefined OV_ECTL_COUPLING_GET then you must upgrade your vorbis library libvorbis to version 1.3.2 or later, http://www.xiph.org/vorbis/.

Prerequisite Packages

You can run `./configure` first to see which packages, if any, are missing. Maybe you don't need them and can turn off their requirement with configuration options mentioned below. All the prerequisites are easy to obtain and install.

alsa/oss
The operating system will probably already have ALSA installed and configured. Older systems, or non-linux, or those few cards that don't have working ALSA support but do have 4Front OSS support, may require installation of OSS.

Some systems such as Ubuntu don't install development files. If you find that the package configures itself for OSS but you know you have ALSA drivers installed, try

```
apt-get install libasound2-dev
```

The package can be compiled without soundcard support if required. This may be appropriate if the installation is purely for post-processing of data and is not required to read from a soundcard. In this case, configure the package with `--without-soundcard` and program `vtcard` will not be compiled.

fftw
Download from http://www.fftw.org/ and install.

libvorbis
Required only for program `vtvorbis`. Disable this requirement with `--without-vorbis`. Download from http://www.xiph.org/vorbis/ and install. Ubuntu users can just do

```
apt-get install libvorbis-dev
```

Ubuntu may also need

```
apt-get install libshout3-dev
```

libFLAC
Required only for program `vtflac`. Disable this requirement with `--without-flac`. To install flac, download from http://flac.sourceforge.net/download.html, or on Ubuntu, do

```
apt-get install libflac-dev
```

xforms
This is used to make control panels for some of the programs. Download from http://xforms-toolkit.org/ and install. Ubuntu users may have to also do

```
apt-get install libjpeg62
apt-get install libjpeg62-dev
```

libsamplerate
Required for sample rate conversion `vtresample`. In the unlikely event that you don't need this, configure with `--without-resample`. Otherwise, download from http://www.mega-nerd.com/SRC/ and install.

ncurses
This is used by `vtstat` to report the state of a stream. You will probably already have the libraries installed, but some Linux installations don't include the ncurses.h header file. On Ubuntu you may have to do

```
apt-get install libncurses5-dev
```

Otherwise, disable the requirement (and program `vtstat`) with configure option `--without-curses`.

libshout
This package is used by `vtvorbis -u` to uplink data to an icecast server. You can bypass this requirement with `--without-shout`, or download from http://www.icecast.org/download.php and install.

X11
Used along with xforms for interactive displays on some of the programs. These are optional. Remove with `--without-x11`; On Ubuntu you may have to do

```
apt-get install libx11-dev
apt-get install libpng12-dev
apt-get install libxpm-dev
```

Other Packages Some other packages that are very useful to have, are `Sox` and `gnuplot`. On Ubuntu, install these with

```
apt-get install sox
apt-get install gnuplot
```

Streams and Buffers

Signal streams

The programs in this package exchange data with one another using a simple packeted data format. Each packet is timestamped and contains sample rate calibration information. The data is streamable, meaning that a program can begin work in the middle of a stream and does not need to see the start. Signal streams can be exchanged through a choice of files, fifos, lock-free buffers, and network connections.

Stream timestamps are preserved through the toolkit programs and data storage and retrieval. Timestamp resolution is 1nS and the accuracy can be better than 100nS if a good PPS is available from a GPS.

Signal samples within the stream have a choice of formats: 8, 16, or 32 bit signed integers, or single or double precision floating point.

A stream can contain any number of signal channels and the samples are interleaved into frames.

Lock-free buffers

These are circular buffers established using shared memory and appear as virtual files under the directory /dev/shm or /run/shm. Each buffer has just a single input process but any number of processes can consume data from the buffer. The input process will never block waiting for consumers. Consumer processes will block and sleep until a packet of data is available. If a consumer process cannot keep up with the flow of data, the buffer will eventually overrun and consumers will be able to detect this and take appropriate action.

Lock free buffers are used where a stream needs to be split n-ways, or where upstream processing needs to be protected from downstream hold-ups.

Stream Naming

Stream files and named pipes use normal unix pathnames. Lock-free buffers use a buffer name which is prefixed with an '@' character to distinguish it from a pathname. Network connections begin with a '+' character.

Examples:-

```
/tmp/foo          A file or named pipe;
@bar              A lock-free buffer called 'bar';
 -                Standard input or standard output FIFOs;
+foo,41720        Send output by TCP/IP to port 41720 on host 'foo';
++foo,41720       As above but with persistent network connection;
+41720            Listen for input on TCP/IP port 41720;
++41720           As above, but listen again if connection dropped;
```

All streams names can be qualified with additional syntax. Input streams can specify which channels within them are to be used. The name is followed by a colon and a comma-separated list of channel numbers (counting from 1).

Examples:-

```
/tmp/foo        Use all channels from input /tmp/foo;
@bar:4          Use only the 4th channel;
/tmp/foo:1,3    Use first and third channels;
+51300:2        Listen on port 51300 and use the 2nd channel;
```

Output stream names may be qualified to specify a data format and, in the case of lock-free buffers, a buffer size.

```
/tmp/foo        Output to file or pipe /tmp/foo using 8 byte floating point;
/tmp/foo,i2     Use 2 byte signed integers instead;
@bar,i4         Create a buffer of 4 byte signed integers, default 10 seconds length;
@bar,20,i4      Create a 20 second buffer of 4 byte signed integers;
@bar,20         Create a 20 second buffer using default f8 format;
+bar,51300,i2   Send to port 51300 at hostname 'bar' using 2 byte signed integers;
```

The available data format options are:

```
f8  8 byte floating point (ie double precision);
f4  4 byte floating point (ie float);
i4  4 byte signed integer;
i2  2 byte signed integer;
i1  1 byte signed integer;
```

All channels within a stream will use the same data format. The default format is `f8`. When using integer data formats, the programs scale sample values so that +/-1.0 corresponds to the maximum value represented by the integer size. You must ensure that output signals from programs directed into integer formatted streams don't exceed +/-1.0.

Network connections can be made to be persistent by beginning the stream name with `++` instead of `+`, for example `++bar,51300`. In this case if a connection drops, or a connection fails to be established, the sending end will keep retrying and the receiving end will continue to listen. This prevents a processing pipeline from collapsing if a network connection has a problem, and is useful in start-up scripts when the order in which hosts are booting is not guaranteed.

General Notes

Commands

Many of the toolkit programs have a lot of command line options, too many to remember. All the programs will accept an option -? which outputs a usage message and option summary. Options and arguments follow the usual unix command line syntax of space separated fields and getopt conventions. Where some options need sub-arguments these are comma separated.

Messages

All of the programs will accept one or more `-v` options to indicate the level of verbosity of messages. With no `-v`, only significant events are reported. Using `-v` will give some additional info, `-vv` for even more detail, and `-vvv` is only useful for software debugging.

Messages will go to a logfile if one is specified (`-L`), and to the stderr stream unless the program is put into background with a `-B` option.

Log files

Many of the programs will produce a log file if given a `-L logfile` option. The `logfile` argument gives the pathname to the required log file. The log file will receive all the messages selected by the `-v` options. Log files are written with an open/append/close sequence for each message, therefore they can easily be removed or renamed for log rotation.

Network connections

Network connections specified by the `+` or `++` syntax are compatible with the netcat utility. For example

```
bar $   vtstat +41720
foo $   vtcat @source +bar,41720
```

has the same effect as

```
bar $   vtstat +41720
foo $   vtcat @source | nc bar 41720
```

or

```
bar $   nc -l 41720 | vtstat
foo $   vtcat @source | nc bar 41720
```

but requires one less process in the pipeline at each end.

When using the `++` syntax to make a persistent connection, the sending end will wait 5 seconds if the connection drops and then try to reconnect, retrying every 5 seconds. A receiving end will wait 1 second after loss of connection and go back to listening. Data will usually be lost during a network interruption and the receiving end will detect a timing break on the stream. When a connection is reestablished, the stream parameters (format,sample rate) must be the same as before, otherwise the receiving end will exit with an error.

With network connections set up using the `+` syntax, the sending end always makes the connection and the receiving end always listens. If you want the receiving end to make the connection, eg to contact and downlink data from a server, you must use `netcat`.

Package version

The installed version can be queried with

```
vtstat -V
```

Test scripts

The installation can be tested using the scripts in the `test` subdirectory. These exercise some of the functions of the software. The test scripts will create some temporary files `/tmp/vttest*`.

After doing a `make install`, run all the test scripts with

```
cd test
./runall
```

Time specifiers

Many of the programs accept a timestamp or timestamp range introduced by a `-T` option. Timestamps are given in ISO format,

```
YYYY-MM-DD_HH:MM:SS.ssss...
```

or as a real number of seconds since 1970-01-01 00:00:00 and are always in UT regardless of local time zone. The fractional part of the second has nanosecond resolution. Timestamps may be truncated, but at least YYYY-MM must be given. The special values 'today' and 'yesterday' can be used for the previous two midnights and the value 'now' is replaced by the current time when the command line is parsed. The following are valid timestamps,

```
2010-12-24_23:59:58.479312
2010-12-24_23:59:58
2010-12-24_23:59
2010-12-24_23
2010-12-24
1293235198.479312
```

A range of time is referred to in these notes as a 'timespec' and has the forms

```
start,end
start,
,end
```

where `start` and `end` are timestamps. If `start` is missing, the start timestamp defaults to the beginning of data. If `end` is missing, the default is the end of data or the current time.

`end` can be supplied as a real number beginning with a '+'. This indicates a number of seconds offset from the start time. For example the following two timespecs both select the same period,

```
2011-03-27_12:15,2011-03-27_13:15.25
2011-03-27_12:15,+3600.25
```

The absence of a timespec usually implies the selection of all available data.

Complex coefficients

Some of the programs accept complex numbers on their command line or configuration, such as the `-c` option of `vtmix` or the coefficients in the `eqmap` of `vtfilter`. This package uses a uniform syntax for these complex coefficients.

Coefficients may be given by their real and imaginary components, for example

```
1.5+0.2j       1.5-0.2j       -1.5+2e-1j
```

Note the 'j' follows the imaginary part. The coefficient can have no imaginary part, or no real part, for example

```
2.5            2.5j           -4.1e-2j    j      -j
```

Alternatively, coefficients may be given by their magnitude and phase lead in degrees, as follows

```
3.5/45        -1.5/350       4.5e-3/-10
```

In either form, coefficients must not have any embedded spaces.

Absolute phase

Several of the programs report either 'absolute phase' or complex amplitudes. For example `vtsid` records absolute phase, `vtwspec` and `vtnspec` report complex amplitudes. The absolute phase and the phase angles of the complex amplitudes are referenced to a cosine wave having a phase of zero at zero timestamp, ie at 1970-01-01_00:00:00.0

In other words, if `vtsid` reports that a carrier at frequency F has absolute phase P degrees, then that carrier can be reconstructed with a sinusoidal factor of

```
cos(2*pi*F*T + P*pi/180)
```

where T is a timestamp.

Periodic waveforms produced by the signal generator `vtgen` are also referenced to a phase of zero at zero timestamp.

A consequence is that all these programs can be stopped and restarted and they will continue to report or generate the same phase.

Signal Processing Modules

vtcat

Copy input stream to output stream.

```
vtcat [options] [input [output]]

options:
 -v          Increase verbosity
 -B          Run in background
 -L name     Specify logfile
 -T timespec Extract specified time range
 -S secs     Skip the first secs seconds of input
 -E secs     Transfer only specified number of seconds, then exit
 -p          Pad timing breaks with zeros
 -a secs     Adjust timestamp by adding an offset of secs seconds
```

examples:

```
   # Select only channels 1 and 3 from the input stream
   vtcat input:1,3 output

   # Swap left and right on a stereo stream
   vtcat input:2,1 output

   # Change a stream data format to 4 byte integers
   vtcat input output,i4

   # Get the stream from buffer @raw on host foo and load it into a buffer
   # of the same name on localhost
   ssh -nc blowfish foo vtcat @raw | vtcat - @raw

   # Send data from channel 2 of buffer @raw to port 9876 on host foo,
   # using 2 byte integer format, and maintain the connection
   vtcat @raw:2 ++foo,9876,i2

   # Listen for data on network port 9876 and transfer anything received
   # into a buffer @remote.  Continue listening if the connection drops.
   vtcat ++9876 @remote

   # Extract a particular time range of data from a stream
   vtcat -T 2010-08-01_14:11:40,2010-08-01_14:33:20 input output

   # Extract 30 seconds of data starting at the given time, and make
   # a histogram from it
   vtcat -T 2010-08-01_14:11:40, -E30 input | vtstat -h bins=100

   # Subtract 9uS from the timestamp of the stream
   vtcat -a -9e-6 input output
```

This program is used for changing the format of a stream, or selecting particular channels from a stream, or selectively transferring data between pipes, named fifos, and lock-free buffers.

If no output stream is specified, stdout is used. If no input or output stream is given, `vtcat` processes data between stdin and stdout.

The timestamp of the stream may be adjusted with a `-a` option. The argument specifies the number of seconds to add to the stream's timestamp.

One thing that `vtcat` doesn't do is concatenate streams. The standard Unix `cat` utility is adequate for that.

`vtcat` in conjuction with `xinetd` can be used to implement a simple stream server. For details see Stream server

vtmix

Additive mixer

```
   vtmix [options] -c matrow [-c matrow] ...  [input [output]]

options:
 -v          Increase verbosity
 -B          Run in background
 -L name     Specify logfile

 -c matrow   Define an output channel by its row of matrix coefficients
```

examples:

```
   # Turn single channel stream into 2 channel stream by duplication
   vtmix -c1 -c1 input output

   # Produce RH circular polarisation response from orthogonal loops
   # ch1 = E/W, ch2 = N/S, loop polarities in phase given north-east signal,
   # delay N/S by 90 deg and add to E/W
   vtmix -c1,-j input output

   # Or instead, advance E/W by 90 deg and add to N/S
   vtmix -cj,1 input output

   # Synthesise single loop oriented 040 deg from the above orthogonal loops
   # sin(40) = 0.643, cos(40) = 0.766
   vtmix -c0.643,0.766 input output

   # Produce RH and LH circular response as a two channel output stream
   vtmix -cj,1 -c1,j input output

   # As above, but include the original orthogonal channels to produce a
```

```
   # four channel output stream
   vtmix -cj,1 -c1,j -c1,0 -c0,1 input output
```

`vtmix` mixes the channels in the input stream according to the coefficients given by the `-c matrow` options. One output channel is produced for each `-c`. The `matrow` argument is a comma-separated list of real or complex coefficients and must have one coefficient (which may be zero) for each of the channels in the input stream. For example,

```
    vtmix -c1.2+0.5j,-0.4-0.1j ...
```

produces an output channel formed from

```
   ch1 * (1.2 + j0.5) + ch2 * (-0.4 - j0.1)
```

If any of the coefficients contain imaginary parts, the mixing is performed in the frequency domain by means of an overlap-add Fourier transform. If only real coefficients are given, the mixing is carried out in the time domain.

Coefficients may be given as complex numbers or by magnitude and phase. See Complex coefficients for the general syntax of the coefficients. Up to 32 input and output channels can be mixed.

vtmult

Multiplicative mixer

```
   vtmult -f freq [options] [input [output]]

options:
 -v          Increase verbosity
 -B          Run in background
 -L name     Specify logfile

 -f freq     Local oscillator frequency, Hz
```

examples:

```
   # Shift SAQ at 17.2kHz down to 400Hz
   vtmult -f16800 @source | vtmix -c1,-j - @dest
```

Each channel of the input is multiplied by a quadrature local oscillator to produce I and Q output signals. The oscillator phase is determined from the input stream's timestamp and is zero degrees at 1970-01-01 00:00:00. The mixer is balanced, so there is no output at the oscillator frequency unless there is a DC component in the input signal.

There are two output channels, I and Q consecutively, for each input channel.

Sum or difference frequencies can be selected by following `vtmult` with `vtfilter`. Upper sideband can be selected by following `vtmult` with `vtmix -c 1,-j` and lower sideband is obtained with `vtmix -c 1,j`.

vtjoin

Combine two or more streams, aligning by timestamp and correcting for sample rate differences.

```
   vtjoin [options] input1 [input2] ... output

options:
 -v          Increase verbosity
 -B          Run in background
 -L name     Specify logfile
```

examples:

```
   # Join together the stream coming through named pipe /tmp/str1
   # with the stream on stdin, selecting just the first two channels
   # from stdin.  Output to /tmp/joined
   vtjoin -- -:1,2 /tmp/str /tmp/joined

   # Turn a single channel stream into a 2 channel stream by duplication,
   # sending output to stdout
   vtjoin @test @test -
```

The input streams are merged together into a single output stream having a number of channels equal to the sum of the selected input channels. All the input streams must have the same nominal sample rate. `vtjoin` uses the timestamp information to align the inputs, and corrects for sample rate differences between the input streams by duplicating or discarded samples as necessary.

The output stream always has the nominal sample rate, ie has sample rate calibration factor of 1.0. `vtjoin` does not 'look' at the signals passing through it, and relies entirely on the incoming timestamps to do the alignment.

`vtjoin` must have an output stream specified, it does not default to stdout. Up to twenty input streams can be joined.

vtsid

Monitor for sudden ionospheric disturbances.

```
   vtsid [options] -c config_file input_stream

options:
 -v          Increase verbosity
 -B          Run in background
 -L name     Specify logfile
```

This program monitors VLF signals within the stream and logs various signal parameters that may be used to detect ionospheric disturbances. Either the whole band can be logged as an array of frequency bins, or specific channels can be logged, or both. Data is saved into a database maintained by `vtsid` and the program `vtsidex` is used to extract records.

Parameters which may be recorded against each bin or channel are:

- Signal amplitude or power;
- Absolute phase;
- Relative phase between channels;
- Bearing (goniometer, poynting vector, or NPE);
- Elevation;
- Polarisation and ellipticity;

If the input stream contains channels derived from orthogonal loops, the bearing (modulo 180 degrees) and relative phase angle between loops can be logged. If an E-field channel is also available, the bearing can be logged modulo 360 degrees by one of three methods. Elevation, polarisation and ellipticity can also be logged.

If the input stream is timestamped with sufficient accuracy, eg using `vttime` and GPS PPS, the absolute signal phase can be logged, mod 360 for CW signals, or mod 180 for MSK signals.

The program is controlled by a configuration file specified by the `-c` option. An example configuration file `vtsid.conf` is included in the package. Use this as a template, copy and edit to your requirements.

vtfilter

General purpose filter.

```
   vtfilter [options] [input [output]]

options:
 -v          Increase verbosity
 -B          Run in background
 -L name     Specify logfile
 -e eqmap    Load equalisation map
 -g gain     Gain factor  (default 1.0)
 -n bins     Number of frequency bins (default 16384)

 -a notchspec   Set automatic notch parameters
                notchspec consists of the following
                comma separated parameters:
                th=threshold,  (default 6.0)
                bw=width, (notch width, default 1 bin)

 -h filterspec  Apply a filter
 -h lp,f=corner,p=poles    Butterworth low pass
 -h hp,f=corner,p=poles    Butterworth high pass
 -h bp,f=center,w=width    Brick wall bandpass
```

```
 -h bs,f=center,w=width    Brick wall bandstop
```

examples:

```
   # Run an automatic notch filter combined with 5 pole high-pass
   # with 500Hz corner frequency, and 3 pole low-pass with 8kHz
   # corner, and an overall gain boost of factor 4.
   # Input from stdin and output to stdout.
   vtfilter -a th=7 -h hp,f=500,poles=5 -h lp,f=8000,poles=3 -g4
```

The input signal is transformed to the frequency domain and multiplied by an array of complex filter coefficients, and is then transformed back to the time domain using an overlap-add scheme. The filter coefficients are the product of all filter responses composed from the `-h filterspec` options.

An automatic notch filter, intended for removal of mains hum and harmonics, is activated with a `-a notchspec` option. This filter monitors the mean amplitude in each frequency bin and if a bin mean exceeds the average of its neighbours by a factor exceeding the threshold, the offending bin and some of its neighbours will be set to zero.

A `-n bins` option can be used to specify the number of frequency bins to use in the Fourier transform. The resulting resolution is given by `sample_rate/(2 * number_of_bins)`. For best results with hum removal using the automatic notch, the resolution should be set to around 0.5Hz or so. The number of bins can be any number but powers of two are the most efficient.

An equalisation map can be applied with `-e eqmap`, where `eqmap` is a text file containing a list of frequencies and complex coefficients. This mechanism is intended for use in equalising the amplitude and phase response of separate receivers so that their signals can be coherently combined. The map file consists of one row per frequency. Each row must start with the frequency and be followed by a list of complex coefficients, one for each of the channels in the input stream. For example, a two channel input stream would require records in the format:

```
   freq coeff_chan1 coeff_chan2
```

Frequencies are given in Hertz and the file can contain any number of entries, which must be in ascending order of frequency. `vtfilter` interpolates logarithmically between `eqmap` records to obtain complex coefficients for each frequency bin, such that the completed transfer function is represented on a Bode plot by straight lines joining the `eqmap` points.

The equalisation coefficients default to unity at DC and the Nyquist frequency, so it usually necessary to include at least these two points in the map file.

To inspect the computed filter coefficients, run `vtfilter` with the option `-d1`. The program will output a table of coefficients in `eqmap` format, one row for each frequency bin, and then exit. The coefficients are the product of all the filters and eqmap specified.

vtevent

Monitor for natural radio events - whistlers and risers.

```
   vtevent [options] -d outdir input_stream

options:
 -v            Increase verbosity
 -B            Run in background
 -L name       Specify logfile
 -p polarspec  Input channel assignments
 -X            Use X display
```

examples:

```
   # Read soundcard into buffer @eraw
   vtcard -Bvv -g1 -d hw:0,0 -r 48000 @eraw,20,i2

   # Filter and event detect
   vtfilter -a th=6.0 -h lp,f=10000,poles=2 -h hp,f=400,poles=2 @eraw |
     vtevent -v -L /tmp/vtevent.log -d /raw/events
```

This program uses a combination of principal component analysis and Hough transform to detect VLF signals having reasonably narrow bandwidth along with rising or falling frequency. Falling frequency signals are identified as whistlers if they conform to the expected shape of a whistler curve as defined by the low frequency Eckersley approximation. A dispersion range of 12.2 to 114 is examined by the Hough transform.

If a `-d outdir` option is given, the detected events are saved under `outdir`. For each event, a thumbnail spectrogram, raw data, and an information text file, are saved in files named by the event's timestamp.

Event detection thresholds are automatically set, relative to the background noise floor of the VLF input signal. The input signal should be pre-filtered to remove hum, as in the example above. If the sample rate is higher than 48k samples/sec, it is more efficient to resample to 48k/sec or lower before applying `vtevent`. A sample rate of 32k/sec gives the best results.

The `-X` produces a display of the analysis processes. This is intended only for testing and tuning, not for normal operation.

The option `-p polarspec` is required if the input consists of more than one channel. It indicates which channels are E-field and which are H-field and their orientations. It allows `vtevent` to estimate the apparent bearing, polarisation, and elevation of the whistler's incident signal. `polarspec` is a comma-separated list with one entry per input channel. Entries are numeric to indicate a loop orientation, or the letter 'E' to indicate vertical E-field. For example, with

```
   vtevent -v -d /datadir -p8,96,E
```

the program expects a 3-channel input with the first channel being a loop oriented on 8/188 degrees, the second channel a loop on 96/276 degrees, and the third channel is the vertical E-field.

vttoga

Measure TOGA and other parameters of sferics.

```
   vttoga [options] input_stream

options:
 -v            Increase verbosity
 -B            Run in background
 -L name       Specify logfile
 -p polarspec  Input channel assignments
 -N count      Capture this many pulses, then exit
 -E seconds    Operate on this many seconds of input, then exit

Frequency bands
  -F start,end  Frequency range (default 4000,17000)
  -M start,end  Mask frequency range (no default)

Trigger options
  -T timestamp  One-shot trigger time
  -c            Calibration mode
  -a thresh     Amplitude trigger threshold (no default)
  -i thresh     Impulse ratio trigger threshold (no default)
  -r rate       Auto threshold to this rate/second

Quality options (experimental)
  -f ascore=val Value 0 to 1, spectrum quality limit (default 0)
  -f pscore=val Value 0 to 1, phase slope quality limit (default 0)
  -f tscore=val Value 0 to 1, TOGA quality limit (default 0)

Output options
  -e opts       Extended output, opts:
                   B   Output multi-band measurements
                   S   Output spectrum
                   T   Output time domain
                   P   Output pulse measurements
                   Q   Output quality test results (experimental)
 -d outdir     Output directory (default stdout)
 -G seconds    Output file granularity (default 3600)
```

examples:

```
   # Set an initial threshold of 0.1 and then adjust it automatically to
   # average 10 triggers per second.   Send the output to files under
   # directory /data/togas and start a new file on 10 minute boundaries.
   vttoga -a0.1 -r10 -d /data/togas -G600 @source

   # Capture 100 calibration pulses from a GPS injected into the VLF
   # receiver input.  Record the spectrum and waveform of each.
   vttoga -a0.1 -eST c -N100 @source > datafile

   # Log the TOGAs and bearings from a 3-axis receiver. ch1 has azimuth
   # 102 deg and ch2 has azimuth 357 deg.  ch3 is an E-field signal
```

```
vttoga -a0.1 -r10 -p102,357,E @source > datafile
```

Sferics and other short pulses are analysed to measure their time of group arrival (TOGA) and some other parameters. The input signal is examined continuously in the time domain in a window of duration 2mS. When the program detects that the window contains an impulse exceeding a trigger threshold, the impulse is analysed. When triggered, the program analyses 1.2mS of signal beginning 0.2mS before the trigger point. If the analysis produces a result of sufficient quality, the measurements are output either to stdout or to a file.

Amplitude threshold triggering is enabled with a `-a` option. The program attempts to analyse all impulses with peak amplitude exceeding the `-a` threshold. A `-i` option enables triggering based on significance of the impulse in the time domain. The total energy in a 200uS window centered on the peak sample is divided by the total energy in a 200uS window preceeding the pulse, to calculate an *impulse ratio*. If the ratio exceeds the threshold set by a `-i` option, the pulse is analysed. If both `-a` and `-i` thresholds are given, then the impulse will be analysed only if both thresholds are exceeded.

By default the band 4-17kHz is used. The lower bound avoids the cut-off frequency of the dominant propagation mode. The upper bound is restricted to avoid strong MSK signals. One or more `-F` options can be used to override the default. The TOGA is measured independently in each band.

If `-r rate` is given, the program adjusts the thresholds to maintain the specified average rate of triggers per second. With multiple input channels, all channels are analysed given a trigger on any channel and the computed TOGA is the amplitude-weighted average of the TOGAs of all the channels.

`vttoga` operates continuously on the input stream unless `-T` is given, in which case a single sferic is analysed using the timestamp as a trigger time. The timestamp should point to a couple of hundred uS before the start of the pulse. `vttoga` requires triggering to occur before the TOGA, therefore if the timestamp is too late, no pulse will be analysed. If the timestamp is too early, accuracy will be reduced.

Using `-E` and/or `-N`, the program terminates after processing the given seconds of input stream or outputing the specified number of sferics.

If a `-p polarspec` is given, the bearing of the sferic will also be measured. `polarspec` indicates which channels are E-field and which are H-field and their orientations. `polarspec` is a comma-separated list with one entry per input channel. Entries are numeric to indicate a loop orientation, or the letter 'E' to indicate vertical E-field. The loops do not need to be orthogonal. The bearing is computed for each frequency bin and the output bearing is the average weighted by the bin amplitudes.

Mains hum and MSK signals and other strong continuous signals should be removed before the signal reaches `vttoga`, such that the sferics dominate the signal when examined with `vtscope`.

Output consists of 'H' records with the following format:

```
H 1540525949.249031 1.635e-01 2.34  240.4
        |             |       |      |
       TOGA           RMS    impulse bearing
```

The bearing field will only be present if a `-p polarspec` option has been given. The RMS field is the square root of the average signal power summed over all the input channels.

With one or more `-e` options, additional records are output following the 'H' record, as described below.

A `-eS` option generates 'S' records which describe the pulse spectrum using one record per frequency bin:

```
S 10105.263  0   3.993e-01 -46.464
     |       |     |          |
 frequency  mask amplitude  phase
```

The phase is the unwrapped phase relative to zero phase at the TOGA at the center frequency of the analysis band (`-F freqspec`) option. Mask is set to 1 if the bin has been masked from use in measurement by a `-M` option.

Option `-eT` produces 'T' records which describe the time domain waveform using one record per signal sample:

```
T 1.979167e-04 3.9439e-02 -1.7892e-02 ...  4.9356e-02  2.2174e+04     0.37
      |            |            |             |           |            |
   time offset  amplitude    amplitude ...  analytic   instantaneous  analytic
                  ch1          ch2          magnitude    frequency     phase (cycles)
```

The time offset is seconds relative to the start of the analysis window. The amplitudes are the input sample amplitudes, one for each channel. These are followed by the magnitude of the complex analytic signal, the instantaneous frequency and the phase in cycles.

A `-eB` option outputs measurements for each band specified by `-F` options.

```
B 1573131278.748969 2.889e-03 4000 12000 0.001382 4.102e-02 24.6     12000 17000 ...
     |                 |        |    |      |        |       |
  Reference          Total     Start End    TOGA     Band    Phase     Start End  ...
  timestamp          energy    frequencies  offset   energy  residual  frequencies
```

The 'B' record timestamp is the timestamp of the first sample of the analysis window. Each band contributes to the output record start and end frequencies, a TOGA offset relative to the reference timestamp, a band energy and a phase residual in degrees.

With option `-eP` the program outputs 'P' records which describe the time domains peaks of the sferic waveform.

```
P 0.008 4.984211e-04 -1.861e-02 11455 5.420608e-04 ...
    |         |          |        |        |
   peaks    peak1       peak1   peak1     peak2
   ratio    offset    amplitude frequency offset
```

The offsets are seconds relative to the reference time given in the 'B' record. Up to six peaks are reported in time offset order. Amplitude is the peak amplitude in sound card units and the instantaneous frequency is that of the analytic signal at the peak. The peaks ratio is a quantity intended to distinguish short range sferics from long distance sferics. The oscillatory waveform of a distant sferic will have a peaks ratio close to zero. A nearby lightning stroke will produce a sferic with a distinct series of pulses with one polarity dominating and the peaks ratio will be closer to +1.0 or -1.0 according to the polarity.

Option `-eQ` generates 'Q' records which report the quality of the sferic analysis.

```
Q 0.84     0.95    0.78
   |        |       |
  ascore  pscore  tscore
```

The three quality scores have range 0.00 to 1.00. A perfect sferic is supposed to achieve a score of 1.00 in each parameter. Lower values indicate lower quality. `ascore` is a measure of the quality of the sferic's spectrum. A low value indicates the presence of multi-path or multi-mode interference, or a collision between two sferics. `pscore` indicates the quality of the phase slope measurement. When measuring multi-band TOGAs (more than one `-F` option), the TOGA of the lower frequency bands should be later than the TOGAs of higher frequency bands. The value of `tscore` reports the extent to which this is the case. The `tscore` compares TOGAs from multiple bands so if only one band is specified, the value will always be 1.00.

Following the extended output records is a single 'E' record:

```
E
```

This marks the end of the records for that sferic and closes off the data set started by the 'H' record. The output format is intended to be easy to parse using programs such as `awk` to separate and plot the data sets.

By default the stream of 'H' and optional extended records is sent to the standard output. If `-d outdir` is given, the output is sent to files created under `outdir`,

```
outdir/YYMMDD-HHMMSS
```

and a new file is started whenever the timestamp crosses a boundary set by the `-G` option.

Sferics successfully triggered and measured can be filtered based on their quality scores using the `-f` option. For example, `-f ascore=0.8` will discard sferics which have `ascore` less than 0.8. Multiple `-f` options can be applied, or combined comma-separated, for example,

```
-f ascore=0.8,pscore=0.7
```

Unless you particularly want to examine sferics above some specific strength, it is recommended to use the impulse ratio triggering with `-i` instead. To capture as many good quality sferics as possible, use `-i` with a low threshold such as 10, combined with filtering `-f` options to restrict the output to high ascore, pscore and tscore values.

As an aid to set up, a `-v` option will output every hundred seconds an 'RC' report to stderr. For example,

```
vttoga: RC raw 13565 ascore 9446 pscore 1202 tscore 18 out 2899
```

The `raw` count is the number of raw triggers accepted by the `-a` and `-i` trigger thresholds. `out` is the number of sferics reported to the output file, having passed any quality score thresholds if set (`-f` options). The other counts report the number of sferics dropped because they failed to meet the respective quality threshold. The counts are reset to zero after each RC report, so just divide the numbers by 100 to obtain average rates per second.

vtrsgram

Produce reassigned spectrogram.

```
   vtrsgram [options] [input]

options:
 -v            Increase verbosity
 -oa           ASCII data output
 -opng         png image output (default)
 -ox           X11 interactive display

 -b bins       Number of frequency bins
 -s stepping   Time axis stepping factor (default 2)

 -a factor     Prune low amplitude points
 -r factor     Prune long range points
 -p factor     Mixed partial derivative pruning

 -g gain       Image (brightness) factor (default 1.0)
 -m scale      Image magnification factor (default 2.0)

 -n            Produce ordinary non-reassigned spectrogram
 -h count      Use homomorphic vertical EQ

 -w window     Specify window function:
                  cosine (default), blackman, hamming,
                  nuttall, hann;
```

examples:

```
   # Render the whistler sample using 320 bins, advancing the FT frame by
   # half the FT width.  Render as a png file four times the size of the
   # STFT grid and discard the weakest 30% of points.
   vtrsgram -b320 -s2 -g4 -m4 -a0.3 /raw/whistlers/1321774508 > 1321774508.png

   # As above but ASCII output,  -g and -m don't apply.
   vtrsgram -b320 -s2 -a0.3 /raw/whistlers/1321774508 > 1321774508.dat
```

This program produces a short time Fourier transform of its input stream on a regular F,T grid of `bins` frequency values and time intervals of `2 * bins/(stepping * sample_rate)` seconds. The amplitudes of the grid cells are then reassigned to new non-grided F,T points according to the reassignment method of Auger and Flandrin. The T value is modified according to the local group delay of the STFT cell and the F value is reasigned to the instantaneous frequency of the cell.

If `-oa` is given, the reassigned points are output as 3-column ASCII numeric values: F, T and RMS amplitude.

Without `-oa`, the reassigned points are mapped to a new grid of size `scale` times the original STFT grid and the result is output as a png file. The `-g gain` option allows the brightness to be adjusted, with larger values of `gain` causing high amplitude points to saturate and low amplitude points to become more visible.

The `-s stepping` option indicates how far the Fourier transform window advances for each T step of the initial STFT spectrogram. With `stepping` of 1, the frame advances by the FT width and each input sample is used only once. A stepping of `N` advances the frame by `1/Nth` of an FT frame and each input sample is used `N` times.

Weak amplitude points can be pruned with `-a prune` where `prune` is a number between 0.0 (no pruning) and 1.0 (everything pruned). Reassigned points are ranked in amplitude order and the weakest `prune` fraction are discarded. A typical value for `prune` would be 0.5 which starts to have a significant effect on the image.

Points lacking local support can be pruned with `-p factor` which examines each point's mixed partial derivative of phase with respect to time and frequency, as per Nelson's method. Smaller values of `factor` will remove more points. `-p0.6` gives quite significant pruning, `-p2.0` gives slight pruning.

Range pruning is activated with `-r factor` and this removes points where the reassignment moves the point by more than `factor` times the STFT cell size. This exploits the redundancy present in the STFT and a point pruned by this option will usually be reinstated more accurately by reassignment of a closer STFT cell. A typical value for `factor` would be 2.0 and larger factors have less effect.

Intensity variation in the vertical direction can be levelled off using the homomorphic filtering option `-h count` where `count` is a small integer. The initial STFT is transformed to a 2D log spatial frequency domain and the lowest `count` number of vertical spatial frequencies are removed. The effect is to remove multiplicative gain variation with frequency. Typical values of `count` range from 1 to 20 and 10 is often sufficient.

vtmatch

Matched filtering and convolution

```
   vtmatch -t template [options] [input [output]]

options:
 -v           Increase verbosity
 -B           Run in background
 -L name      Specify logfile

 -F freqspec  Select frequency range (default all)

 -c           Convolve input stream with template
              (default is matched filter)
```

`vtmatch` implements a matched filter in which the input stream is continuously compared with a template waveform. The output consists of the cross-correlation of the input signal with the template.

The argument to `-t` specifies the name of an ASCII data file which supplies the waveform of the template. The file must contain a single column of numeric values in ASCII format representing time domain samples of the template at the same sample rate as the input stream.

If `-c` is given, the programs outputs the convolution of the input stream with the template. A `-F` option can be used to restrict the frequency range of the operation.

vtping

Meteor ping detection

```
   vtping [options] -d outdir input

options:
 -v          Increase verbosity
 -B          Run in background
 -L name     Specify logfile
 -X          Use X display

 -D thresh   Detection threshold (default 2.0)
 -R thresh   Rejection threshold (default 2.0)

 -F detect,low,high    Detection frequency range, Hz
 -F reject,low,high    Rejection frequency range, Hz

 -t secs     Trigger holdoff (default 0.4 seconds)

 -d outdir    Place event files under outdir
 -s low,high  Output spectrogram frequency range

 -ote         Output unix epoch timestamp
 -oti         Output ISO timestamps (default)
```

examples:

```
   # Monitor pings from two band 1 TV carriers at 55.25MHz
   vtrtlsdr -F 55.249e6 -r1000000 |
      vtresample -r8000 |
      vtmix -c1,-j |        # Select upper sideband
      vtping -F detect,340,400 -F detect 1600,1700 -F reject,500,1500 -d /data/pings
```

`vtping` monitors the audio stream from a VHF receiver looking for impulses that resemble meteor pings. Detected events are reported to standard output or, if a `-d outdir` option is given, each ping produces a set of three files in `outdir`.

A `-F detect` option specifies a frequency range to examine. Genuine meteor pings occupy a small band around the carrier frequency of the distant VHF transmitter. Interfering noise pulses have a much wider bandwidth and can be ignored by using `-F reject` options to cover a suitable region of spectrum outside the ping band. For example `-F detect,900,1100 -F reject,300,800 -F reject,1200,3000` will look for pings between 900Hz and 1100Hz and reject candidate pings if they spread into the surrounding reject regions. Multiple `detect` and `reject` bands can be specified.

Thresholds for detection and rejection are given by `-D` and `-R` options, respectively.

`vtping` automatically equalises the spectrum based on a moving average input noise floor. Some input is required to establish the noise floor and `vtping` will not detect events during the first 30 seconds of input, and will take a minute or two to settle the equalisation. It is best therefore to run it on a continuous stream or long duration batches.

When using `-d outdir`, each ping produces a thumbnail spectrogram, a short .vt file containing the audio data, and a text file which reports the timestamp and detection levels. Without `-d outdir`, events are reported to standard output, for example

```
PING 2014-02-12_15:22:43.268 5.571 1.309
```

The numbers following the timestamp are the detection and rejection levels respectively, which are comparible with the `-D` and `-R` threshold values. The timestamp may be switched between string and numeric formats using `-oti` and `-ote` options.

`vtping` is responsive to short soft underdense pings and head echos exhibiting significant Doppler shift. It will ignore long echos from overdense trails and it will ignore the distant carrier signal if present.

vtblank

Impulsive noise blanker

```
vtblank [options] input output
```

```
options:
 -v          Increase verbosity
 -B          Run in background
 -L name     Specify logfile
 -c          Clip impulses at the threshold
             (default is to blank the output)
 -d secs     Dwell time, seconds
             (default is 0.005 seconds)
 -h thresh   Threshold amplitude
 -a factor   Automatic threshold factor
 -t secs     Time constant for moving average
             (default 100 seconds)
 -e chanspec Examine only these channels
 -b chansepc Apply blanking only to these channels
```

examples:

```
# Apply a threshold of 12 times the mean, with zero dwell and 100 second
# time constant.   This is a reasonable nighttime setting when looking for
# narrow band signals underneath VLF background.
vtblank -a12 -d0 -t100
```

This program removes impulsive noise such as sferics and switching transients which exceed a given threshold, and is intended for use when looking for weak continuous signals.

Impulses may be blanked (output set to zero) or clipped (output limited at the threshold). When the input signal exceeds the threshold, the output is blanked or clipped for a retriggerable duration given by the `-d` dwell time option. Without `-d`, only those samples exceeding the threshold are modified.

A `-h` option specifies the threshold amplitude. Usually it is better to use the `-a` option which sets the threshold automatically to be the given factor higher than the exponential moving average noise floor of the signal. The time constant of this moving average can be set by a `-t` option.

By default, if the threshold is exceeded on any channel, all the channels are blank or clipped. This is usually desirable for further signal processing. A `-e` option can be used to restrict which channels are examined for triggering of the blanker, and a `-b` option will specify which channels the blanking or clipping is applied to. This can be useful to allow use of a wider bandwidth to detect the sferics while applying the blanking to a narrower band signal. Options `-e` and `-b` expect a comma-separated list of channel numbers.

Note that, from version 0.8 onwards, the automatic blanking threshold set by `-a` has a different range of thresholds.

Storage and Retrieval

vtwrite

Write stream to disk files

```
   vtwrite [options] input datadir

options:
 -v          Increase verbosity
 -B          Run in background
 -L name     Specify logfile
 -G seconds  Output file granularity (seconds)
             (default 86400 = 1 day)
```

examples:

The input stream is recorded into a set of disk files which are stored in the directory `datadir`. The format of the files is identical to that of the stream, which means the data files can later be used directly as input to VT modules.

Output files are named by their start time. For example, if `datadir` is `/raw`, an output file will have a name such as `/raw/110624-195425`.

A new output file is started whenever the stream timestamp crosses a boundary specified by the `-G` file granularity option. By default (86400 seconds), this starts a new file at midnight. A setting of `-G 3600` will start a new file on each hour.

A new file is also started if a timing break is detected on the input stream, or if the `vtwrite` program is stopped and restarted.

`vtwrite` stores data in the file in the same format as that of the input stream. Therefore, if you want to store samples in say `i2` format, you must provide an input stream in `i2` format, eg

```
   vtcat input -- -,i2 | vtwrite /raw/vt.
```

As mentioned above, the data files can be used directly as input to VT programs, or the `vtread` program can be used to extract data, which allows specific time ranges to be read, seamlessly across multiple files.

vtread

Read a stream from disk files created by `vtwrite`

```
   vtread [options] datadir

options:
 -v            Increase verbosity
 -B            Run in background
 -L name       Specify logfile
 -T timespec   Restrict records to a range of times.  Default is all records
```

`vtread` is used to selectively extract stream data from a set of files produced by `vtwrite`. `datadir` specifies the directory containing the file set. The program will scan the files in `datadir` to extract as much data as possible within the time range given by `-T timespec`.

See Time specifiers for the description of `timespec`.

vtgen

Signal generator.

```
   vtgen -r rate [options] output

options:
 -v            Increase verbosity
 -B            Run in background
 -L name       Specify logfile
 -r rate       Specify sample rate (no default)
 -c chans      Number of channels (default 1)
 -g gain       Output gain (default 1.0)
 -T start,end  Range of timestamps
```

```
 -t            Throttle output rate to approximately real time
 -s a=amplitude,f=freq,p=phase
               Generate a cosine wave, amplitude specifies peak
 -n a=amplitude
               Generate normally distributed white noise, amplitude
               specifies mean
 -u a=amplitude
               Generate uniformly distributed white noise, amplitude
               specifies the peak
 -p a=amplitude,f=freq,p=phase,d=duty
               Generate rectangular pulses, amplitude specifies level
 -m a=amplitude,f=freq,p=phase,b=bitrate
               Generate an MSK signal
 -d v=level    Generate DC level
 -a seed       Seed the random number generator
 -i infile     Specify input file for time source
```

examples:

```
   # Send an 18kHz sine wave out through the ALSA soundcard port hw:0,0 at
   # 48k samples/sec stereo
   vtgen -s a=0.5,f=18000 -r 48000 -c2 | vtraw -ow | aplay -

   # Simulate the MSK signal from NSY at 45.9kHz, using vtmix to produce
   # orthogonal loop signals with the MSK bearing 155 degrees.
   vtgen -r192000 -m a=0.1,f=45900,b=200,p=38.2 -t |
     vtmix -c -0.960 -c 0.423 | vtsid -c sid.conf
```

`vtgen` creates a stream containing a mix of signals, pulses, and noise. Multiple signal component options `-s,-n,-p` can be given and the signals are mixed together into the output stream. The timestamp of the stream is taken from the system clock unless a `-T` or `-i` option is given. Output is throttled to approximately real time rate if `-t` is given, otherwise data is generated as fast as possible.

Amplitude options should be given in the range 0 to 1, frequency is specified in Hertz, duty cycle must be in the range 0 to 1, phase is given in degrees.

The phase is an absolute phase taking account of the timestamp. Zero phase is at 1970-01-01 00:00:00.

MSK can be generated with a `-m` option, which is useful for testing `vtsid`. Set b= twice the value of the sid monitor's br= setting.

vtfm

Frequency modulator/demodulator.

```
   vtfm [options] input output

options:
 -v            Increase verbosity
 -B            Run in background
 -L name       Specify logfile

 -m            Modulator
 -d            Demodulator (default)

 -F hertz      Specify carrier frequency (modulator only)
 -k index      Modulation index, Hz per unit
```

examples:

```
   # Demodulate an FM sensor signal, carrier at 12kHz, 1kHz deviation produces
   # one unit output amplitude
   vtmult -f12000 @source |
      vtfilter -h lp,f=2000,poles=8 |
      vtfm -d -k1000 - @signal
```

This program extracts a signal which has been frequency modulated onto a carrier, for example with a voltage-to-frequency convertor. It is intended for capture of low frequency sensor signals.

The input stream must carry pairs of baseband I/Q signals, one pair per sensor channel, and the input must be band-limited to the width of the FM signal. This typically involves the use of `vtmult` followed by `vtfilter`.

The modulation index given by the `-k` option indicates the frequency deviation in Hertz required to produce an output amplitude of 1 unit.

With `-m` and `-F` options, the program functions as a modulator. This is only useful for generating test signals.

With suitable allocation of the soundcard spectrum, several low frequency sensors can be mixed together, each with its own carrier frequency, into a soundcard input channel.

`vtfm` can also be used on the output of `vtrtlsdr` to demodulate broadcast or communications FM signals.

vtam

Extract the magnitude of analytic signals.

```
usage:  vtam [options] input output

options:
  -v            Increase verbosity
  -B            Run in background
  -L name       Specify logfile

  -g gain       Gain (default 1.0)
```

examples:

```
   # Combine a signal with its Hilbert transform to make an analytic signal,
   # then use vtam to extract the envelope
   vtmix -c1 -c-j @vlf | vtam | vtjoin @vlf - - | vtscope
```

`vtam` expects one or more pairs of I/Q input channels and outputs the magnitude of each pair. With N I/Q signals there are 2N input channels and N output channels.

The program can be used to demodulate AM signals, eg from a RTL2832, after mixing down to baseband with vtmult. It is most useful to extract the instantaneous amplitude of an analytic signal. See also Analytic signals

vttime

Refine the timestamp of a stream.

```
   vttime [options] input output

options:
 -v          Increase verbosity
 -B          Run in background
 -L name     Specify logfile
 -c chan     Specify input channel containing timing signal
 -m method,params  Specify timing method and parameters
 -h seconds  Holdover limit (default none)
```

examples:

```
  # Re-time the signal in buffer @raw using the positive-going baseband PPS
  # signal on channel 2.  The centroid of the PPS pulse occurs 250uS later than
  # the UT second.
  vttime -c2 -m centroid+,c=250e-6 @raw @output

  # Use a narrow (~10uS) PPS for timing
  vttime -c2 -m pulse+ @raw @output

  # Timing using the leading edge of a long (> 0.1 second) duration PPS.
  vttime -c2 -m edge+ @raw @output

  # Re-time the stream using the incoming stream timestamps.
  vttime -m none @raw @output
```

A new timestamp is assigned to the stream, and the stream is resampled to the exact sample rate. Output samples are synchronous with UT, that is, there is a sample having a timestamp of the exact second and all the other samples are at intervals of 1/sample_rate.

A timing signal is presented to the program on the input channel given by the `-c` option. The `-m` option selects the timing method to use and indicates the expected form of the timing signal, as follows:

- **`-m centroid+,w=width,c=offset`**
  A baseband, positive-going, pulse per second, typically obtained from a GPS. For best results, the pulse rise and fall times should be slowed using suitable RC networks, such that the transitions take place over many (a few hundred) samples. This enables the pulse centroid to be located in time to a small fraction of a sample period. Parameter `c=offset` specifies the offset in seconds between the centroid of the pulse and

the second mark. The `width` parameter can be used to fix the width of the measurement buffer. It is usually best to leave this parameter out, in which case `vttime` will automatically choose a suitable width.

- **`-m centroid-,w=width,c=offset`**
  As above, but looks for a negative going pulse.
- **`-m pulse+,c=offset`**
  Measures the phase center of a short GPS pulse. The pulse duration should be between 0.5 and 2 sound card sample periods. A duration of 10uS is typical. The `offset` param adjusts the timing with respect to the phase center.
- **`-m pulse+,c=offset`**
  As above, but expects a negative going pulse.
- **`-m edge+,c=offset`**
  Measures the phase slope of the leading edge of a long duration GPS pulse. The pulse duration should be between 0.1 and 0.5 seconds. The `offset` param adjusts the timing with respect to the leading edge.
- **`-m none`**
  Does not measure a timing pulse. It just resamples the stream to an exact rate relying on the timestamps of the input stream.

With a good quality low noise PPS, the timing jitter should be below 50nS and it is often possible to achieve better than 30nS.

Limited use is made of the timestamp of the input stream. It is used only to get a rough estimate of when to expect the next timing pulse and should be accurate to better than 0.5 seconds otherwise the second will be ambiguous.

For advice on how to use `vttime`, see Timestamps and alignment

vtresample

Change the sample rate of a stream.

```
   vtresample -r rate [options] [input [output]]

options:
  -v          Increase verbosity
  -B          Run in background
  -L name     Specify logfile
  -r rate     Convert to this sample rate
  -q qual     Conversion quality:
               qual = 0  fastest (default)
                    = 1  medium
                    = 2  best
  -g gain     Gain factor (default 1.0)
```

examples:

The input stream is converted to the new sample rate specified by the mandatory `-r rate` option. `rate` must be an integer, but can have any ratio to the input sample rate.

The program will not run with a sample rate ratio more than about 250 in either direction and will terminate with 'src_process error'. In that situation, perform the conversion in two stages.

Input/Output Programs

vtcard

Initialise and read data from a soundcard.

```
   vtcard [options] [outstream]

options:
 -d device   Specify the OSS or ALSA device to use
             (default /dev/dsp or hw:0,0)
             Use -d- to read standard input
             Use -dq to list bus devices (ALSA only)
 -c chans    Number of channels to read (default 2)
 -b bits     Bits per sample (default 16)
```

```
 -r rate     Sample rate (no default, mandatory option)
 -g gain     Output gain (default 1.0)
 -u          Disable sample rate tracking
 -v          Increase verbosity
 -B          Detach from terminal and process group to become
             a daemon program
 -T stamp    Apply a dummy timestamp when reading stdin
 -A opts     Specify buffering options,
               -A b=buffer_size,p=period_size
             sizes in bytes, either or both b= or p= can be given
             If not specified, reasonable defaults are used
 -L name     Specify logfile
```

examples:

```
   # Simple test using the default soundcard in stereo
   vtcard -vv -r48000 @test
   vtstat @test

   # Read /dev/dsp2 at 192k/sec 4 byte samples, 2 channels, into
   # a lock-free buffer called dsp2raw.  The buffer is made 30 seconds
   # long and uses 4 byte integer values.
   vtcard -r 192000 -d /dev/dsp2 -b32 @dsp2raw,30,i4

   # Read ALSA soundcard, specifying a large buffer size suitable for 192k/s
   # Run in background, output to buffer called @hew
   vtcard -Bvv -A b=262144,p=2048 -d hw:1,0 -b32 -r 192000 @hew,20,i4

   # Use the USB device in port 3 of hub attached to port 5 of host
   # controller #2.  (ALSA only)
   vtcard -d usb:2-5.3 -c2 -r48000 @raw

   # Use the PCI device in slot 4 of PCI bus 05 (ALSA only)
   vtcard -d pci:0000:05:04 -c2 -r48000 @raw
```

This program initialises the soundcard for input and reads data into the specified output stream. It is strongly recommended to use a lock-free buffer as the output stream in order to prevent soundcard overruns.

If `vtcard` is run with superuser permissions, it will assign itself a real-time priority and will lock itself and its output buffer into memory. Under these conditions `vtcard` will only block while waiting for soundcard data and soundcard overruns are extremely unlikely.

It may take a minute or two before `vtcard` starts to write data to its output stream. During this time it makes an initial measurement of the soundcard's sample rate. After this, `vtcard` continues to monitor the sample rate (against the system clock) and includes this calibration information in its output stream.

The data is timestamped using the system clock, after correcting for the latency reported by the driver. If a soundcard overrun does occur, `vtcard` initiates a reset of the card and there will be a break in the output stream while the sample rate and timestamp are re-evaluated.

If the card cannot run at the requested sample rate, `vtcard` will use the closest available rate.

If the option `-d-` is given, `vtcard` will read raw PCM samples from its standard input. In this mode, an initial timestamp is taken from the system clock, and thereafter the sample rate is assumed to be exact. A `-T` option can be given to supply an alternative start time.

The program will choose a suitable period or fragment size and buffer length. These can be overriden with a `-A` option.

Option `-dq` will output a list of available sound devices along with their USB or PCI bus address. A bus address (for example usb:1-7.2 or pci:0000:00:05) can be given to the `-d` option to unambiguously specify which soundcard is required, based on the physical port that the device is plugged into. This avoids problems caused by ALSA's random assignment of card numbers. Currently this function does not work with OSS.

For further information, see Soundcards

vtvorbis

Encode to, or decode from, ogg/vorbis and multiplex the timestamp data as a second logical stream within the ogg container.

```
   vtvorbis [options] [-e|-d] name

options:
```

```
-v          Increase verbosity
-e          Encode
-d          Decode
-B          Run in background
-L name     Specify logfile
-p          Expect/generate only a pure ogg/vorbis stream
            (default is to multiplex with a timestamp stream)

encoding options:
-q factor   Specify VBR encoding with this quality factor
            factor range -0.1 to 1.0
-b kbps     Specify CBR encoding at this kbps per channel
            (default is CBR, 64kbps per channel)
-i          Independent encoding of each channel

decoding options:
-E secs     Decode only specified number of seconds, then exit.

network options:
-u method,server,port,mount,passwd
               Uplink to icecast server
-n server,port Send over network, without protocol, to port on server
-n port        Listen on port, without protocol
-k             Retry failed uplink connections
               (default is to exit if connection drops)
-t             Throttle uplink to sample rate
            (default is to encode and uplink as fast as possible)
```

examples:

```
# Send data from @test to a buffer with the same name on host xyz
vtvorbis -eq0.5 @test | ssh -c blowfish xyz vtvorbis -d @test

# Filter and stream channel 1 of the raw VLF signal in buffer @eraw,
# as pure ogg/vorbis to Icecast server
vtfilter -a th=7 -h hp,f=500,poles=2 -h lp,f=10000,poles=2 -g4 @eraw:1 |
vtresample -r 32000 |
vtvorbis -ep -q0.4 -ktu shout,46.4.26.83,80,/vlf1,xxxxx

# Downlink ogg/vorbis from a stream server.  Let wget make the http
# connection as it knows all about proxies, authentication, etc,
# and use aplay to listen to it.
wget -q -O- http://46.4.26.83/vlf1 2> /dev/null |
vtvorbis -dp | vtraw -ow | aplay -

# Send vorbis stream to server abelian.org port 1234
vtvorbis -ei -q0.4 -kn abelian.org,1234 @source

# Listen on port 1234 for incoming vorbis connection
vtvorbis -d -kn 1234 @destination
```

By default, `vtvorbis -e` multiplexes the timestamp and sample rate calibration as a second stream within the ogg container. This allows the receiving `vtvorbis -d` to reconstruct the VT stream, preserving timestamps across the link. Media players and other decoders are supposed to ignore any unrecognised streams in the container, but some don't and therefore when sending to a decoder other than `vtvorbis`, a `-p` option may be a necessary encoder option.

The `-E` option is useful in scripts which sample a stream and need to capture a specific length of data for analysis. The output will be approximately the requested number of seconds in length - output is stopped at the end of the vorbis page following the specified end point.

A `-i` option forces independent encoding of the channels. The default is to allow the vorbis encoding to exploit the redundancy between channels, which is usually desirable for efficient encoding of stereo but introduces spurious correlation of background noise and weak signals between channels. Therefore, when encoding independent streams, a `-i` option is necessary to avoid this problem.

`-n server,port` can be used with `-e` to send encoded data to a remote server. The server should use `-d -n port` to listen for the incoming connection. The connection has no connection or wrapping protocol and is equivalent to piping the vorbis stream through a `netcat` connection.

When uplinking over a WAN, a `-k` option is recommended. This will ensure that a dropped connection is retried until eventually reestablished. `vtvorbis -ek` will retry a failed connection every 20 seconds. While the link is down, encoded data will be discarded so that the upstream processing pipeline is not choked. Without `-k`, the encoder will exit if the uplink fails. `-k` can be used with both `-u` for protocol connections, and with `-n` for simple connections.

Recommended mode of operation is VBR with a quality factor of 0.3 or 0.4 for a single channel of 32k samples/sec.

LAN network connections can use the netcat (nc) program to good effect when the hosts involved are safely behind a firewall. For WAN connections, ssh is usually necessary for connections between `vtvorbis`. In this case, use blowfish encryption as this is the most efficient.

`vtvorbis` will do sample rates 1k/sec to 200k/sec when using variable bit rate encoding (`-q` option). With constant bit rate (`-b` option) the range is 15k/sec to 50k/sec.

vtflac

Lossless compression/decompression using FLAC, preserving timestamps.

```
   vtflac [-e|-d] [stream]

options:
 -v          Increase verbosity
 -e          Encode
 -d          Decode
 -B          Run in background
 -L name     Specify logfile
```

examples:

```
   # Archive an i2 format stream file
   vtflac -e /raw/111225-000000 > /arc/111225-000000.fx

   # Unarchive the same file
   vtflac -d /raw/111225-000000,i2 < /arc/111225-000000.fx
```

`vtflac -e` applies loss-less compression to the stream data using FLAC and multiplexes the compressed audio with the timestamp information. The result is a file which is readable only by `vtflac -d`.

Audio data is converted to 16 bit (i2) resolution by the encoder, and if the original file is i2 or lower resolution, the decoded samples are recovered exactly.

If no input or output buffer or filename is given, `stdin` and `stdout` are assumed and the program operates in a pipeline.

The files produced by `vtflac -e` will not be playable by a FLAC player because of the multiplexed timestamp data. This program is intended for archival of stream files but can also be used to transfer streams across a network connection if bandwidth is at a premium. The FLAC encoder is set to run at the maximum compression available and a file of VLF data is typically reduced to about 50% of the original (i2 format) file size.

vtain

Read A/D converters on Beaglebone

```
   vtain -r rate -c mask [options] [outstream]

options:
 -c mask     Bit mask 0..255 specifying which A/D channels to read
 -r rate     Sample rate (no default, mandatory option)
 -g gain     Output gain (default 1.0)
 -z offset   Input offset 0..4095 (default 0)

 -v          Increase verbosity.
 -B          Detach from terminal and process group to become
             a daemon program
 -L name     Specify logfile
```

examples:

```
   # Read AIN1 at 200 samples/second, scaling the A/D range of 0..4095
   # to output range -1 .. +1
   vtain -c1 -r200 -z2048 -g2 @output

   # Read all 8 A/D converters at 20 samples/second
   vtain -c255 -r20 @output
```

This program reads samples from the A/D convertor nodes in the directory /sys/devices/platform/omap/tsc. Up to 8 channels can be read, specified by the bit mask argument to the `-c` option.

By the default, the 0..4095 range of the A/D is mapped to 0..+1 in the channel data.

vtdata

Read data from ASCII source

```
   vtdata -r rate -c chans [options] [outstream]

options:
 -c chans    Number of channels in input file (no default)
 -r rate     Sample rate (no default, mandatory option)
 -g gain     Output gain (default 1.0)

 -v          Increase verbosity.
 -B          Detach from terminal and process group to become
             a daemon program
 -L name     Specify logfile
 -T timespec Start time to assign to data
 -i          Timestamp in input column 1
 -t          Throttle data flow to real time rate
```

examples:

```
   # Read two channels from a two-column text file
   vtdata -c2 -r32000 -g2 @output < source.txt
```

This program reads data from a flat ASCII text data file and converts it into a timestamped stream. Maximum input record width is 4096 characters. Input fields are white space separated and leading white space is ignored. A hash or semicolon in the input file is treated as a comment and causes the rest of the line to be ignored. Blank input lines are ignored.

The data is timestamped by the current time at the start of the program unless a start time is specified by a `-T` option. If `-i` is given, `vtdata` will expect a timestamp in column 1 of the input file and if no `-T` option is given, the column 1 timestamp will be used. If both `-i` and `-T` are given, the command line timestamp will be used and the input file timestamp will be ignored.

Program `vtraw -oa` can be used with `vtdata -i` to pipe stream data through some arbitrary signal processing script.

vtwavex

Extract data from WAV file

```
   vtwavex [options] [input [output]]

options
 -v           Increase verbosity.
 -B           Detach from terminal and process group to become
              a daemon program
 -L name      Specify logfile
 -T timespec  Start time to assign to data
 -r rate      Override sample rate in WAV file
```

examples:

```
  # Extract signal, defaulting to current start time
  vtwavex source.wav > source.vt

  # Extract signal assigning a specific start time
  vtwavex -T 2017-12-01_15:31 source.wav > source.vt

  # Extract a 2-channel I/Q signal for decoding by EbNaut
  vtwavex source.wav | vtraw -oa | ebnaut -d ...
```

Signal is extracted from the WAV file and output as a timestamped stream. If the WAV header contains an 'inf1' chunk supplied by Spectrum Lab, the start time of the stream will be set to the 'ut' field of the inf1 chunk. Otherwise the current RTC is used as the start time unless overridden with a `-T` option. The sample rate is taken by default from the WAV header. If 'inf1' is present and contains a sample rate, this is used instead. Both may be overridden with a `-r` option.

The WAV file may contain any number of channels. Only uncompressed formats are supported.

vtrtlsdr

Take data from RTL2832U dongle.

```
   vtrtlsdr [options] buffer_name

options:
  -v        Increase verbosity
  -B        Run in background
  -L name   Specify log file
  -d device Device number (default 0)
  -d ?      List available devices
  -r rate   Sample rate 1000000 to 3200000

  -F hertz  Tuner frequency, Hertz
  -g gain   Tuner gain (dB, default 0 = auto)
  -g ?      List available gain settings
  -q        Invert the Q signal

  -u        No sample rate tracking
  -T stamp  Start time when using -u
            (default is to use system clock)
```

examples:

```
   # Play FM broadcast station at 102.7Mhz
   vtrtlsdr -u -F 102.7e6 -r 1000000 |
      vtresample -r200000 |
      vtfm -k1e5 | vtresample -r48000 |
      vtraw -ow | aplay -

   # Record meteor pings in upper sideband at 55.25Mhz, into 1 hour files
   vtrtlsdr -F 55.249e6 -r1000000 |
      vtresample -r8000 | vtmix -c1,-j -- - -,i2 |
      vtwrite -G3600 /data/meteor
```

This program uses `librtlsdr` to retrieve raw I/Q data from the tuner dongle. `-F` specifies the center frequency in Hz and bandwidth is determined by the selected sample rate.

Output is a 2-channel stream with channel 1 carrying the I signal and channel 2 carrying Q.

`vtrtlsdr` may take a minute or so to measure the sample rate of the device before it starts to deliver data to the output stream. This function is turned off with a `-u` option.

Allowed sample rates are 1e6 to 3.2e6 samples/second.

To include `vtrtlsdr` you must configure with `--with-rtlsdr` and install some prerequisite libraries. For more info see RTL2832U

vtsdriq

Take data from SDR-IQ receiver.

```
   vtsdriq [options] buffer_name

options:
  -v        Increase verbosity
  -B        Run in background
  -L name   Specify log file
  -d device Device node (default /dev/ttyUSB0)
  -r rate   Sample rate (default 196078)
            Rate must be one of
               8138, 16276, 37793, 55556
               111111, 158730, 196078

  -F hertz  Frequency, Hertz

  -g gain   RF gain in dB (default 10)
            Valid range -8 to +34
  -a        Insert -10dB attenuator
  -i        IF gain in dB (default 0)
            Allowed values 0, 6, 12, 18, 24

  -q        Invert the Q signal
  -u        No sample rate tracking
  -T stamp  Start time when using -u
            (default is to use system clock)
```

examples:

```
   # Receive upper sideband 14.0 to 14.0277 Mhz without tracking the
   # sample rate and display the spectrum
   vtsdriq -F14e6 -r55556 -g10 -i0 -q -u |
      vtmix -c1,-j | vtspec

   # Receive a single channel streamm of baseband VLF with sample rate
   # tracking
   vtsdriq -F0 -r55556 -g10 -i0 @vlf,i2
```

This program initialises and reads data from an SDR-IQ receiver connected via USB. `-F` specifies the center frequency in Hz and bandwidth is determined by the selected sample rate.

`vtsdriq` may take a minute or so to measure the sample rate of the device before it starts to deliver data to the output stream. This function is turned off with a `-u` option.

If the frequency given by -F is non-zero, the output is a two channel I/Q stream with channel 1 carrying the I signal and channel 2 carrying the Q signal. If a frequency of zero Hertz is selected, the output is just a single channel of real data.

To include `vtsdriq` you must configure with `--with-sdriq`.

vtraw

Extract audio from a stream.

```
   vtraw [options] [stream]

options:
 -v          Increase verbosity
 -B          Run in background
 -L name     Specify logfile
 -g factor   Overall gain factor (default 1.0)

 -ob      16 bit raw interleaved PCM output (default)
 -oa      Produce ASCII output
 -ow      WAV output containing 16 bit PCM
```

examples:

```
   # Produce a WAV file from a whistler event file
   vtraw -ow 1325151287.vt > 1325151287.wav

   # Play a stream in real time
   vtraw -ow @signal | aplay -

   # Extract a short piece of signal for analysis
   vtread -T2010-09-15_12:31:02.47,+0.02 /raw | vtraw -oa > sferic.dat
```

`vtraw` outputs the audio content of its input stream, by default in raw 16 bit PCM with interleaved samples. With a `-oa` option, multi-column ASCII output is produced with the timestamp in column 1 and the audio channels in the remaining columns.

WAV output format is produced with the -ow option.

The signal level can be altered using a multiplicative gain factor via the `-g` option, but you'll need to avoid clipping when using the 16 bit binary and WAV outputs.

Display and Plotting Programs

Programs for displaying signals and producing graphs.

vtstat

Report statistics

```
   vtstat [options] input

options:
 -E secs     Examine only specified number of seconds, then exit.
 -h hspec    Produce a histogram. hspec has the form
```

```
              max=max,bin=bins
 -a aspec    Report amplitude statistics.  aspec has the form
               r=secs
             r= specifies output record interval in seconds
 -i          Report summary statistics
```

Without options, `vtstat` uses the terminal to display some stream statistics in real time.

A histogram is generated if `-h hspec` is given. `bins` specifies the number of amplitude bins, `max` gives the max sample amplitude.

The `-a` option extracts the first three moments of the rectified input signal, averaged over a period given by the `r=` parameter. If `r=0` is given, a single output record is produced for the entire input stream. The `-a` option allows amplitude probability distributions to be measured using standard metrics such as `Vd` or parameterisation by Gaussian, Rayleigh or Weibull distributions.

Option `-i` produces summary statistics to stdout.

vtscope

An oscilloscope to display a stream in the time domain.

```
   vtscope [options] input

options:
 -v          Increase verbosity
 -display display  Specify the X host and display number
                   (By default, the DISPLAY environment variable is used)
```

examples:

```
   # Display the signal passing through buffer @raw
   vtscope -v @raw

   # Merge the signals in buffers @raw1 and @raw2 and display them all
   vtjoin @raw1 @raw2 - | vtscope -v

   # Display just channels 1 and 3 in @raw
   vtscope -v @raw:1,3
```

This program uses the X display to make an oscilloscope and control panel.

Most of the controls should be obvious. Clicking the mouse on the display will report the timestamp and amplitude of the point clicked. Drag the display left or right to scroll through the capture buffer, or use the scroll bar. Button 'Arm' puts the scope into single shot mode, 'Run' sets continuous mode. If neither of these is pressed, the scope retains the captured buffer for analysis.

The display can be resized just by dragging a corner of the window.

One of the channels, or `UT`, can be selected as a trigger source. If `UT` is selected the scope triggers on the UT second mark - this is useful for examining timing pulses. A fixed pre-trigger period of 50mS is used, ie the trigger point will be at 50mS on the time axis.

The 'Plot' button puts the current waveforms into an interactive `gnuplot`. Press 'q' in the `gnuplot` window to dismiss. 'Save' stores the data into a file in ASCII format with space separated fields. The first column is the absolute timestamp, second column is the time offset into the capture buffer, and the remaining columns are the channel amplitudes.

vtspec

A spectrum analyser

```
   vtspec [options] [input]
```

examples:

This program uses the X display to make a spectrum analyser and control panel. If an input stream is not given, stdin is assumed.

The display can be scrolled and shifted by dragging with the mouse or by working the scroll bar and thumbwheels. Drag a corner to resize. A selection of window functions is available. Use 'Rect' for accurate peak level measurements but some of the others, eg 'Nuttall' or 'Blackman' have much lower scalloping effects but introduce amplitude errors that `vtscope` doesn't correct for.

The 'Plot' button puts the spectrum data into an interactive `gnuplot`. Press 'q' in the `gnuplot` window to dismiss. 'Save' stores the data to an ASCII text file with space separated fields. The first column is the frequency and the remaining columns are the bin amplitudes for each channel. Phase information is not stored.

vtcmp

Compare two channels in various ways.

```
   vtcmp -m mode [options] input

options:
 -S secs     Skip seconds of input before starting
 -E secs     End after accepting seconds of input

 -m type     Type of comparison.  type is one of
              cor     Cross correlation (default)
              pd180   Phase difference, mod 180
              pd360   Phase difference, mod 360

 -F freqspec
 -w seconds  Half-width over which comparison is made
 -N count    Average this number of input buffers
 -r          Envelope correlation
```

examples:

```
   # Outputs the correlation coefficient between channels 3 and 4
   # of the input stream, up to a range of +/- 10mS.
   vtcmp -m cor -w 0.01 source:3,4 > datafile

   # Produce an envelope correlation, smoothing the envelope by
   # ignoring components above 500Hz.
   vtcmp -m cor -w 0.01 -r -F,500 source:1,2
```

This program makes various types of comparison between two input channels, and writes the results to stdout. Output is multi-column plain ASCII. Exactly two input channels must be specified. `-S` and `-E` options can be used to limit the time range of the input signal to be analysed. If not given, the program continues to the end of input. The program averages the comparison in units of time given by the `-w` option.

With `-m cor` the cross correlation is averaged over the duration of the input signal. The range of time offsets to be reported is given by the `-w` argument.

Option `-F freqspec` can be used to restrict the frequency to be examined. `freqspec` takes the form `start,end` where `start` and `end` are frequencies in Hertz. Either can be omitted and `start` defaults to zero and `end` defaults to the Nyquist frequency.

A `-r` option causes both input channels to be rectified by squaring and the resulting 'power' signals are presented to the cross-correlation. A `-F` option should be used to low-pass filter the envelope as in the example above.

vtnspec

A narrow band spectrum analyser.

```
   vtnspec [options] input

options:
 -B              Run in background
 -L logfile      Specify logfile
 -v              Increase verbosity

 -N frames       Average up to this many transform frames, then exit
 -c              Coherent averaging of frames until end of input
 -a              Non-coherent averaging of frames until end of input

 -f hertz        Center frequency
 -w width        Frequency width in Hertz
```

```
 -r resolution   Frequency resolution - bin width in Hertz

 -p              Pad with zeros across timing breaks
 -W window       Specify window function:
                     rect (default), cosine, blackman, hamming,
                     nuttall, hann
```

examples:

```
   # Check sample rate accuracy against RSDN-20 on 11.9kHz using a 10uHz
   # bin size, using a sferic blanker to improve S/N ratio
   vtblank -a12 -d0 input |
     vtnspec -cv -f 11904.7619 -w 0.02 -r 0.00001 > data.txt
```

This program analyses its input signal in the frequency domain using an array of Goertzel filters. It is especially suited to examining a narrow piece of spectrum or to implement overlapping bins.

The frequency range specified by the `-f` and `-w` options is spanned by Goertzel filter bins separated by the resolution given with the `-r` argument. `-f` specifies the center frequency and the program always uses an odd number of bins so that there is always a bin positioned at the center frequency. If a width of zero is specified, a single bin will be transformed, positioned at the center frequency. A transform is completed after `1/resolution` seconds. By default, the program will issue an output and exit after one transform is completed.

If `-a` or `-c` is given, `vtnspec` will average successive transform outputs non-coherently or coherently, respectively, until end of input, or until the number of frames given by a `-N` option have been processed.

Output is multi-column space separated ASCII, one row for each bin, with the following format:

```
   ...
   1000.00000000 9.269461e-05 1.213293e-05 6.610408e-05 -1.023862e-05 ...
        |             |            |            |             |
   bin frequency   ch1_real    ch1_imag     ch1_rms      ch2_real     ...
```

If `-a` has been given to specify non-coherent averaging, the output contains just the RMS column for each channel:

```
   ...
   1000.00000000 2.330445e-05 1.539060e-05 ...
        |             |            |
   bin frequency   ch1_rms     ch2_rms     ...
```

`vtnspec` is typically used with up to a few hundred bins. If more than 1000 or so bins are used, the program will become very slow and it may be quicker to use `vtwspec`.

vtwspec

A wideband spectrum analyser.

```
   vtwspec [options] input

options:
 -B            Run in background
 -L name       Specify logfile
 -r hertz      Resolution (bin width)
 -a            Non-coherent averaging until end of input
 -N frames     Average up to this many transform frames, then exit

 -W window     Specify window function:
                   rect (default), cosine, blackman, hamming,
                   nuttall, hann
```

examples:

```
   # Average 60 seconds of source in 1Hz resolution
   vtcat -E60 source | vtwspec -a -r1 > datafile
```

All channels of the input stream are transformed to the frequency domain using a FFT and the resulting spectrum is sent to the stdout. Unless the `-a` option is given, the program does a single transform of length `sample_rate/resolution` and then exits. If `-a` is given, the program performs successive FFTs until the input stream ends, averaging the signal power in each bin. A `-N` option can be given to tell `vtwspec` to end the averaging and terminate after the given number of FFT frames.

The transform output is sent to stdout in ASCII format with one row per frequency bin. The first column is the bin frequency in Hz. Without `-a` there are three further columns for each channel: real amplitude, imaginary

amplitude, and the RMS magnitude. With `-a`, there is a single column per channel: the RMS magnitude.

vtpolar

Polar A/V display of orthogonal signals.

```
   vtpolar [options] [input [output]]

options:
 -v            Increase verbosity
 -B            Run in background
 -L name       Specify logfile
 -s size       Display diameter, pixels (default 224)
 -r rate       Frame rate (default 10)

 -m mode       Display mode (default tf)
                 -mf   spectrogram
                 -mt   time domain (vectorscope)
                 -mtf  both

 -am           Mono audio (default)
 -as           Stereo audio

 -b size       Buffer size before ffmpeg (default 10)
 -k from,to    Frequency range, Hz (default DC to Nyquist)
 -p polarspec  Specify alignment of input channels
 -c            Paint background

 -ga=gain      Audio gain (default a=1.0)
 -gv=gain      Video gain (default v=1.0)

 -f options    Options passed to ffmpeg
    options is a comma separated list of parameters:
     format=container_format, (default avi)
     vcodec=video_codec, (default mpeg4)
     acodec=audio_codec, (default libmp3lame)
     vbr=video_bitrate, (default 100)
     abr=audio_bitrate, (default 64)
```

examples:

```
   # Pipe mpeg A/V stream to mplayer
   vtpolar -f format=mpegts,vbr=1000,abr=128 -s400 -p96,6 @hloops | mplayer -cache 1024 -

   # Produce an avi file from 1 hour of signal extracted from saved data
   vtread -T2010-10-15_05:00,+3600 /data | vtpolar -s360 -r10 - file.avi

   # Send an A/V stream to ffserver.   The ffserver config specifies the
   # bitrates, container and codecs.  We specify the size and frame rate.
   vtpolar -s400 -r12 @hloops http://server/stream.ffm
```

This program takes a 2-channel stream from a pair of loops, or 3-channels consisting of 2 loops and an E-field signal, and produces a polar display of frequency and bearing and/or amplitude and bearing. Bearings are extracted mod 360 if an E-field channel is available, otherwise they are mod 180.

The display video is multiplexed with the audio stream and encoded using `ffmpeg`. `output` is a pipe, filename, or the uplink URL to an ffserver. Use `ffmpeg -formats` to get a list of available container formats and codecs. Some container formats, such as `swf` will only accept certain audio sample rates, in which case use `vtresample` before `vtpolar`.

The `-p polarspec` option indicates the assignment and alignment of each of the input channels. For example `-p 354,93,E` would indicate a three channel input with channel 1 from a loop oriented on 354 degrees, channel 2 oriented 93 degrees, and channel 3 is an E-field channel. For good results it is essential to equalise the phase response of the receivers involved. The loop signals do not have to be orthogonal.

A 10 second buffer is maintained between `vtpolar` and `ffmpeg`. The size may be changed with the `-b seconds` option but if the buffer is too small a deadlock will occur and `vtpolar` will go into a busy loop.

By default the accompanying audio is mono. If two inputs are provided, the mono audio is the sum of both channels. If there are three inputs the mono audio is taken from the E-field channel. Option `-as` switches to stereo audio derived from the two H-field channels.

At present, the http uplink option seems to be broken due to a problem with `ffmpeg` or `ffserver`.

vtplot

Time domain plotting

```
   vtplot [options] [input] > imagefile

options:
 -s x,y      Plot size in pixels, default 640,480
 -t title    Title for the plot
 -o format   Specify output image format
```

examples:

```
   # Show available output image file formats
   vtplot -o?

   # Plot 15mS of data extracted from a signal database
   vtread -T2011-04-12_03:27.7246,+15e-3 /datadir |
     vtplot -s 1200,600 -t "Nice multimode tweek" -o x11

   # Plot 30mS from the real-time data in @source
   vtcat -E0.03 @source | vtplot -t "Live snapshot" -o 'png small' > live.png
```

`vtplot` is a shell script which uses `vtraw -oa` to convert the input stream to ASCII data and then invokes `gnuplot` to do the plotting.

If no `-o` option is given, `vtplot` will display the time domain waveform on the X display. Enter 'q' into the persistent display window to remove it.

vtsidex

Extract records from the data files produced by `vtsid`.

```
   vtsidex [options] datadir

options:
 -v            Increase verbosity
 -d datadir    Specify data file root directory, no default
 -m monitor    Specify a monitor.  Defaults to spectrum data.
 -o format     Specify output format options, see below
 -T timespec   Restrict records to a range of times.  Default is all records
 -F freqspec   Restrict spectrum columns to a range of frequencies. Default is
               all frequencies.
 -f filename   Specify a particular file name
 -s            Report status
 -a n          Average over n raw records
 -h degrees    Apply hysteresis to angle values
 -t            Tail the data
```

Selected records are extracted from the `vtsid` data files under the directory given by the `datadir`. If a `-m` option is given, the monitor's records are extracted, otherwise spectrum data is extracted.

Option `-T timespec` delimits the time range to be extracted. See Time specifiers for a description of `timespec`.

When extracting spectrum data, option `-F freqspec` can be used to restrict the frequency range extracted. `freqspec` takes the form `start,end` where `start` and `end` are frequencies in Hertz. Either can be omitted and `start` defaults to zero and `end` defaults to the Nyquist frequency.

Output options are introduced by `-o` flags. These are:

```
 -ote   Timestamps in unix epoch, seconds since 1970-01-01 00:00:00
 -oti   Timestamps in format YYYY-MM-DD_HH:MM:SS.SSSS  (default)
 -otr   Timestamps in seconds offset from the start time
 -oh    Output a heading record at the start of data
 -ohn   Issue a heading record every 'n' data records
 -otb   Output a blank line when there is a timing gap in the data
```

The `-otb` option is useful when piping the data into gnuplot, causing the timing break to be represented by a gap in the chart.

A summary of a data file is produced by the `-s` option. A filename may be given, in the form YYMMDD-HHMMSS, otherwise the current data file is reported. The `-m` option specifies which monitor is referred to.

Records are averaged if `-a n` is given. An output record is generated after every `n` raw records. The timestamp assigned is the timestamp of the first raw record of the average. If a timing break occurs in the input stream, a new average is started. Bearings and phase angles are averaged correctly, modulo 180 or 360 according to the type of raw data field.

To avoid too much wrapping of phase and bearing measurements, a hysteresis can be applied with `-h degrees`. This is useful for plotting noisy angles that slowly move through the wrapping angle. For example `-h20` will allow a 0-360 degree bearing to range from -20 to 380 degrees, and a -180 to +180 phase angle will use the range -200 to +200. This option can also be used with a large hysteresis, eg `-h5000` to completely unwrap long phase changes, such as occur during the day/night transitions during which the signal phase can change by several complete cycles.

vtsidplot

Plot data from `vtsid` monitor data

```
   vtsidplot -m monitor -T timespec [options] datadir [> imagefile]

options:
 -m monitor     Specify a monitor
 -T timespec    Time range
 -s x,y         Plot size in pixels, default 640,480
 -t title       Title for the spectrogram
 -a count       Average over count raw records
 -h degrees     Apply degrees of hysteresis to phases and bearing

 -f fields      Specify fields to be plotted (default all)
                Use -f? to show available fields

 -o format   Specify output image format (default x11)
             vtsidplot -o? for available formats
```

examples:

```
   # List the available fields for monitor NAA
   vtsidplot -m NAA -T2011-03-18 -f? /raw/sid

   # Plot carrier phase and bearing for a whole day, into a png file
   vtsidplot -m NAA -T2011-03-18,+86400 -f 'cp1 b360' -o 'png small' /raw/sid > out.png
```

This is a script which invokes `vtsidex` to extract the requested data and runs `gnuplot` to do the plotting. The options `-m, -T, -a, -h` and the database directory are passed down to `vtsidex`. See the description of `vtsidex` for their usage.

If `-f fields` is given then only the requested fields are plotted. The `fields` argument is a comma separated, or a quoted space separated, list of fields. A `-f?` option will report the available fields and a `-T` option must be given with this because the available fields may depend on the configuration of `vtsid` at that time.

Due to a limitation of `gnuplot`, this utility is only useful when the monitor logging interval is greater than one second. If not, then `vtsidplot` may still be used if a `-a count` option is given such that the interval between averaged records is greater than one second.

vtsidgram

Plot spectrograms from `vtsid` spectrum data

```
   vtsidgram [options] datadir [> imagefile]

options:
 -s x,y      Plot size in pixels, default 640,480
 -t title    Title for the plot
 -o format   Specify output image format
 -T timespec Specify time range
 -F freqspec Specify frequency range
```

examples:

```
   # Produce output in png format and display with 'xv'
   vtsidgram -T2011-10-17_10:30,+3600 -F12500,14500 -o 'png small' -s320,480 /raw/sid | xv -
```

This is a script which runs `vtsidex` to extract spectrum data and passes it to `gnuplot` to render in `pm3d` mode. The `-T timespec` and `-F freqspec` options are passed on to `vtsidex` and are described above. With no `-o` option, the x11 display is used, otherwise an image file is sent to stdout.

`vtsidgram` selects the amplitude fields from the output of `vtsidex` and if there is more than one amplitude field (multiple channels), the RMS sum of the amplitudes is used.

vtsgram

Plot spectrogram

```
   vtsgram [options] [input] > imagefile

options:
 -t title   Title for the plot
 -p pps     Horizonal pixels per second (default 100)
 -s 'opts'  Extra options passed to Sox
 -b bins    Number of frequency bins (variable default)
```

examples:

```
   # Plot 3 seconds of the source, at 200 pixels per second
   vtcat -E3 @source | vtsgram -p 200 > file.png

   # Plot the signal file with intensity range -20 dB to -70 dB
   vtsgram -s '-Z-20 -z50' signal.vt | display
```

`vtsgram` is a shell script which invokes the Sox utility to produce a spectrogram in png format. Sox will choose the number of frequency bins to use unless you supply a `-b bins` option. By default, the title text of the plot will be the timestamp of the start of the plot. Alternatively the title can be specified with a `-t` option.

A `-s` option passes its argument to Sox which enables detailed control over the plot. The Sox options should be quoted to contain them in a single argument to `-s`. Some of the Sox options are listed below:

```
   -Z dB    Set the signal amplitude corresponding to full intensity
   -z dB    Set the dynamic range of the plot
   -m       Monochrome palette
   -l       Use colours more suitable for printing
   -r       Turn of axes and legends
```

See the spectrogram section of the Sox man page for more details and options.

For more information about `vtsgram`, see Spectrogram plots

Utility Programs

A selection of programs useful for monitoring the signal processing modules.

vtps

Display all VT processes.

```
   vtps [options] [modes]

options:
 -f          Full format
 -F          Extra full format
 -l          Long format
 -o format   Custom format options

modes:
   u       Show memory and cpu used
   v       Show virtual memory
   w       Wide listing
```

This program is just a front-end for the `ps` utility, which is run for every VT process on the host. The options and mode letters on the `vtps` command line are just passed on to `ps`. See the `ps` man page for more options.

If no options are given, `ps` is run with the options `-o pid,rtprio:2=RT,pcpu,pmem,command`.

vttop

Active display of VT processes.

```
   vttop [options]

options:
 -b          Run in batch mode (useful for scripting)
 -d ss.tt    Delay between updates, seconds.tenths
```

This program is a front-end for the `top` utility, running `top` for all VT processes on the host. Options are just passed straight on to `top`. See the `top` man page for more options and interactive commands.

vtwait

Wait for data on a stream

```
   vtwait [options] input

options:
 -v          Increase verbosity
 -t          Wait for data on the stream
             (Default is to wait for buffer creation
```

examples:

```
   # Run vtcard to read the soundcard into buffer @raw
   vtcard -B -d hw:0,0 -r 48000 @raw,20,i2

   # Wait for @raw to be ready
   echo "waiting for @raw..."
   vtwait -t @raw || {
      echo "buffer @raw invalid" >&2
      exit 1
   }

   # Start programs which use @raw for input
   vtfilter ... @raw -
   vtwrite ... @raw
```

This program is intended for use in start-up scripts in which `vtcard` or `vtvorbis` is used in background to run data into a lock-free buffer. The program waits for the buffer to be created, and if `-t` is given, it continues to wait until data is available.

Without this program, the start-up script would proceed to launch subsequent processing modules which may fail immediately because their input buffer does not yet exist.

vtcardplot

Plot soundcard performance

```
   vtcardplot [options] vtcard_logfile [> imagefile]

options:
 -d date     Plot only the specified day, yyyy-mm-dd (default whole logfile)
 -s x,y      Image size (default 640,480)
 -o format   Output format (default x11)
```

examples:

```
   # Plot the performance of the vtcard and display on x11
   vtcardplot -d 2011-11-18 /tmp/vtcard.loop_rx.log

   # Show available output formats
   vtcardplot -o?
```

This is a script which uses `gnuplot` to graph the soundcard conversion rate and timestamp error logged by a `vtcard` program. The `vtcard` logfile must be given and if no `-d date` option is supplied, the entire log file is plotted.

The offset graph shows the timing offset of the output stream relative to the system clock. A positive offset indicates that the stream timestamp is late (has a higher timestamp value).

vtdate

Conversion and operations on timestamps

```
   vtdate [options] [timestamp|timespec]

options:
 -v       Increase verbosity
 -i       Integer output
 -n       Output numeric timestamps
 -s       Output ISO format timestamps

 -t secs  Truncate to a multiple of secs seconds
 -a secs  Add secs seconds to the timestamps
```

examples:

```
   # Get the current time as an ISO string
   vtdate -s now

   # Split a timespec into numeric start and end fields
   set -- `vtdate -n 2011-10-09_08:32:17.49,+2400`

   # Subtract an hour from a timestamp
   vtdate -a -3600 2011-10-09_08:32:17.49

   # Convert a numeric timestamp to ISO form
   vtdate -s 1318151537.49
```

This utility takes a timestamp or a timespec and prints their values to stdout in accordance with the `-n` and `-s` options. Outputs have a fixed length canonical form, eg

```
   1318145537.490000
   2011-10-09_07:32:17.490000
```

If neither `-n` nor `-s` are given, both forms are output. Output fields are space separated and if a timespec is given, the start and end times are output in separate fields. A `-i` option will discard the decimal point and following digits.

This utility is intended for use in scripts that need to manipulate timestamps.

vtspot

Geographic calculations, trilateration and triangulation.

```
   vtspot [options] meas1 [meas2 ...]
   vtspot -b standpoint forepoint
   vtspot -d standpoint bearing range
   vtspot -s location timestamp
   vtspot -m site=file [...]

options:
 -v           Increase verbosity
 -L           Specify logfile
 -c factor    Specify a velocity factor (default 0.9922)
 -n minsites  Minimum number of sites to attempt a solution (default is all sites)
 -r residual  Maximum residual to accept stroke solution, in units of measurement
              standard deviation (default 1.0)
```

examples:

```
   # Bearing and range of 44.6N,67.3W as seen from 53.7N,2.1W
   vtspot -b 53.7N,2.1W 44.6N,67.3W

   # Determine source from two arrival times and a bearing
   vtspot T/39.9N,74.9W/1334382837.867327 T/53.7N,2.1W/1334382837.884484 B/53.7N,2.1W/298

   # Calculate azimuth and elevation of the Sun
   vtspot -s 53.703N,2.072W 2017-12-21_12:06:30

   # Output great circle points in 2 degree steps
   vtspot -g 52.146N,8.458E 53.703N,2.072W 2
```

This program takes an arbitrary mix of arrival time and bearing measurements and calculates the most likely source locations and times for the signal. All the calculations are performed using n-vectors to avoid

singularities at the poles.

Measurements may be given on the command line or in the standard input stream. Each measurement must conform to one of the following formats:

```
T/location/time[/std_dev]
B/location/bearing[/std_dev]
A/location1/location2/atd[/std_dev]
I/ident
```

'T' measurements specify an arrival time, 'B' measurements specify a bearing, and 'A' records specify an arrival time difference (for example from a cross-correlation). Each measurement can specify a standard deviation. Arrival times are given as a `timespec`, see Time specifiers for the description of `timespec`. Bearings are given in degrees mod 360. Standard deviations are specified in seconds for timestamps and ATDs, and degrees for bearings. These default to 20e-6 seconds and 5 degrees. Locations are specified as `latitude,longitude` and the following examples illustrate the formats recognised:

```
53.703N,2.072W
53.703,-2.072
53:42:10.8N,2:04:19.2W
```

You can also give locations using a symbolic name listed in the `spots` file, see below.

The collection of measurements is referred to as a 'measurement set'. A measurement set must contain at least two bearings, or three arrival times, or two arrival time differences. Combinations are allowed, such as two arrival times and one bearing, or one ATD and a bearing. If the measurement set is given on the command line, `vtspot` processes the measurement set and then exits. Multiple measurement sets can be presented to `vtspot` on the standard input stream with one measurement set per line and each will be processed independently.

An 'I' field in the measurement set tags the set with an identifying string token which appears in the output records. This is useful when reading from standard input in order to associate each result with its measurement set.

If a solution cannot be found for a measurement set because the measurements do not intersect anywhere (for example one or more of the timestamps or bearings are not consistent with the others) then no output will be made for that measurement set.

Some measurement sets will have two solutions, for example a pair of bearings or three arrival times will have two exact solutions. Two solutions can also occur with some measurement sets where the receivers are not well located with respect to the source.

Results go to standard output with the following format:

```
ident index latitude,longitude timestamp residual km ns
```

The `ident` field will be missing if the measurement set did not supply a `I/ident` field. Index is 0 or 1 to distinguish the two solutions possible with some measurement sets. The residual is the worst error of the solution tested against each measurement of the set and is in units of standard deviations. `km` is an estimate of the location accuracy in km. `ns` is the number of receiver sites contributing to the measurement set. An example output looks like

```
test1 0 44.594,8.948 1522493298.065420 0.42 2.04 3
test1 1 39.928,8.475 1522493298.063946 0.01 0.02 3
```

This is a simulated lightning stroke received at three sites - thus two equally likely solutions.

Locations can be given symbolically using site names defined in a file named `spots`. An example `spots` file is included in the package. This should be placed in the working directory or in the user's home directory. The symbolic names can be used wherever the program expects a latitude,longitude.

A `-b` option computes the bearing of `forepoint` as seen from `standpoint`. A `-d` option computes the forepoint given a standpoint, bearing and range in km.

`vtspot` can be used to match and solve batches of sferic TOGAs produced by `vttoga -d`. The source file for each site is given with `-m site=filename`, where `site` is the latitude,longitude or symbolic name of the site, and `filename` is the path to the output file of `vttoga`. At least six sites must be used and the sites should surround the lightning source. Ideally twelve or more sites should be used. Up to 200 sites can be specified for matching. Sferics in the source files are matched into lightning strokes and solved for the stroke location and time.

In matching mode, `vtspot` will output one record for each successfully resolved lightning stroke, for example

```
370 -0.899,30.277 2018-04-18_00:01:12.422935 0.90 11.30 6
```

The fields are as follows:

- 1: An incrementing serial number to identify the solution, zero for the first solution in the batch;
- 2: The estimated location latitude,longitude;
- 3: The timestamp of the lightning stroke;
- 4: The residual value of the downhill simplex, in units of measurement standard deviations;
- 5: An estimate of the sferic timestamp error in uS;
- 6: The number of sferics (sites) used for this solution;

Timestamps are numeric by default. Option `-o iso` will produce ISO timestamp strings instead. An extended matching output format can be requested with option `-o ext`. With the extended output format, the above matching result will look like:

```
A 370 -0.899,30.277 2018-04-18_00:01:12.422935 0.90 11.30 6
R site-15  5575.7  -21.0 -13.2 1.827e-04 0.00
R site-38  5963.5   -7.0 -10.1 1.705e-04 0.00
R site-6   6241.9  -15.9   9.4 1.628e-04 0.00
R site-35 11778.9  -51.2 -10.1 8.554e-05 0.00
R site-19 12194.0  -61.0   9.4 8.254e-05 0.00
R site-37 12405.3  -45.3  14.5 8.122e-05 0.10
E
```

The normal output record now has an 'A' prefix. The following 'R' records list all the receivers and sferic measurements which contributed to the solution. The 'R' records have fields:

- 1:The symbolic name of the site if given with the -m option, otherwise the latitude,longitude;
- 2:Distance from lightning stroke to receiver, in km;
- 3:Azimuth of the receiver from the standpoint of the stroke, in degrees;
- 4:Residual timing error, uS;
- 5:RMS amplitude of the sferic as reported by the site in its vttoga output file;
- 6:Path illumination factor, indicates the fraction 0.0 to 1.0 of the path which is in daylight;

The illumination factor in field 6 will be 0.0 for a path in darkness, 1.0 for entirely in daylight. Intermediate values imply the sferic has crossed a terminator.

An 'E' record marks the end of data for that lightning stroke. On completion of the matching batch, a 'G' record is output for each receiver site, for example:

```
G site-37    1233    632   12.0
```

The 'G' record has the following fields:

- 1: The symbolic site name or latitude,longitude;
- 2: The number of sferics supplied by that site;
- 3: The number of sferics successfully matched into stroke solutions;
- 4: The sum of all the timing offsets, in units of uS;

Following the 'G' records, a final 'Q' records is output. This marks the end of output for the batch and supplies some summary information. For example:

```
Q 37800 20086 3034 1200
```

Fields are:

- 1: Total number of sferics processed;
- 2: Total number of sferics successfully matched;
- 3: Total number of stroke solutions;
- 4: The overall duration in seconds of the sferic data;

In this example, 20086 sferics are matched from 37800 supplied, a usage of 53.1%. On average each stroke solution is composed of 20086/3034 = 6.62 sferics;

In matching mode, a `-n sites` specifies the minimum number of sites required for a solution. The value should be 5 or more. A `-r` option specifies the maximum residual to accept a solution, in units of standard deviations. The default value is 1.0.

The current version of `vtspot` uses a spherical Earth model. It also uses a fixed value of group velocity for all paths. For these reasons the output locations can be quite inaccurate, especially at long range.

See Lightning location for advice on the use of `vtspot` in matching mode.

vtubx

U-blox GPS configuration

```
   vtubx [options] device

options:
 -v             Increase verbosity

 -F [tp,]Hz     Set frequency output, Hertz
 -P [tp,]secs   Set pulse-per-second output, width in seconds
 -S             Save configuration to GPS non-volatile memory

 -b?            Query SBAS
 -b+            Enable SBAS
 -b-            Disable SBAS

 -rh            Report hardware status
 -ra            Report antenna settings
 -rv            Report GPS software and hardware versions

 -f             Output timestamp and position fix records
 -n             Output NMEA sentences
 -l             List satellites

 -N?            Query NMEA protocol version
 -N version     Set NMEA protocol version

 -g?            Query GNSS settings
 -g options     Setup GNSS

 -T?            Query timing mode
 -T options     Set timing mode
```

examples:

```
   # Set up a series 8 u-blox to use GPS, Galileo and GLONASS, and save settings
   vtubx -g +GPS=16,-SBAS,+Galileo=8,-BeiDou,-IMES,+QZSS=1,+GLONASS=14 -S /dev/ttyACM0

   # Set up a series 7 u-blox to use GPS and save settings
   vtubx -g +GPS=21,+QZSS=1,-SBAS,-GLONASS -S /dev/ttyACM0

   # Set up a series 7 u-blox to use GLONASS and save settings
   vtubx -g -GPS,-QZSS,-SBAS,+GLONASS=22 -S /dev/ttyACM0

   # Program timepulse 1 for 2mS pulse width and save settings
   vtubx -P 2e-3 -S /dev/ttyACM0

   # As above, but explicitly timepulse 1
   vtubx -P 1,2e-3 -S /dev/ttyACM0

   # Set timepulse 2 to 100 kHz frequency
   vtubx -F 2,100e3 /dev/ttyACM0

   # Disable SBAS on series 6
   vtubx -b- /dev/ttyACM0

   # Output fix data, one record per second
   vtubx -f /dev/ttyACM0

   # Initiate survey-in following by timing mode on a series 8 device
   # Survey-in for at least 8 hours and continue until all position
   # axes are accurate to within 1.5 metres.
   vtubx -T 8,1.5 -S /dev/ttyACM0
```

`vtubx` configures or queries the mode of operation of u-blox series 6, 7, and 8 GPS receivers. Command line arguments are applied from left to right. Not all models of u-blox GPS will accept all commands. See U-blox GPS for more information.

The `-g` option configures which GNSS networks the device will use. Use `-g?` to query the device to see which GNSS can be used. The options string is a comma-separated list of GNSS. A leading minus disables use of that network. A leading plus enables that network and the maximum number of receiver channels to use can be specified. For example `-g +GLONASS=14` enables the GLONASS network and allocates up to 14 receiver channels.

Options `-P` and `-F` configure a timepulse output to the specified frequency (Hz) or pulse width (seconds). Some devices have two timepulse outputs and the frequency or width can be prefixed with the timepulse number. For example, `-P 2,0.1` sets timepulse output 2 to 100mS width. The rising edge of the pulse indicates the timing mark. The timepulse output can be inverted by giving a negative frequency or width.

When used as a timing reference, SBAS should be turned off with `-b-` or equivalently, `-g -SBAS`.

Devices such as the M8T can be operated in timing mode, in which the position of the GPS is fixed and all of the satellite measurements are devoted to timing accuracy. Timing mode is initiated by requesting a 'survey in' using a `-T hours,metres` option. This puts the GPS into survey-in mode in which it averages its position for at least the specified number of hours, until the position is settled within the specified number of metres on all three axes. When survey-in is completed the device automatically enters timing mode. `-S` should be used to store the surveyed position and timing mode setting.

Soundcards

This package relies on ALSA or OSS drivers to deal with the soundcard. The program `vtcard` is used to read data from the soundcard. For each soundcard, a `vtcard` process must be started to transfer data from the card to a lock-free buffer.

- Preferably, run the soundcard at its maximum sample rate;
- A lock-free buffer should always be used for the output stream of `vtcard`;
- Ensure the PC running `vtcard` is running `ntpd`.
- If possible use the soundcard's line input rather than the mic input, because this will have a flatter response and lower distortion.
- Run `vtcard` with superuser privilege and it will use real-time scheduling for maximum reliability.

Begin by running `vtcard` with `-vv` which will report some information about the soundcard. For example,

```
vtcard -vv -r96000 -d hw:0,0 @test
```

You may get an error such as `"cannot set channel count"` if the card does not have two channels, in which case use option `-c1`. The output will look something like

```
vtcard: buffer name: [@test]
vtcard: using ALSA device hw:0,0
vtcard: rate min 44100 max 96000
vtcard: sample rate set: 96000
vtcard: period size: 256 frames, 2666 uS
vtcard: buffer size: 8192 frames, 85333 uS
vtcard: periods per buffer: 32
vtcard: nread: 256
vtcard: avail min: 256
vtcard: channels: 2  bytes: 2
vtcard: using SCHED_FIFO priority 99
vtcard: pre-run complete 96011.35
vtcard: setup -2.612e-04 sr 96007.59 rr 95986.3 n=0 r=0.86
vtcard: setup +3.878e-05 sr 96008.15 rr 96011.3 n=1 r=0.13
vtcard: setup +3.024e-04 sr 96012.50 rr 96037.2 n=3 r=0.99
vtcard: setup -2.767e-04 sr 96008.52 rr 95985.9 n=4 r=0.91
vtcard: setup +2.870e-05 sr 96008.93 rr 96011.3 n=5 r=0.09
vtcard: setup +2.933e-04 sr 96013.15 rr 96037.1 n=8 r=0.96
vtcard: setup -2.850e-04 sr 96009.05 rr 95985.8 n=9 r=0.93
vtcard: run +2.715e-05 sr 96009.180 tadj +2.320e-06 rr 96011.7 offs +0.271mS r=0.09
vtcard: run +2.122e-05 sr 96009.282 tadj +1.814e-06 rr 96011.2 offs +0.212mS r=0.07
vtcard: run +2.246e-05 sr 96009.390 tadj +1.920e-06 rr 96011.4 offs +0.225mS r=0.07
vtcard: run +2.070e-05 sr 96009.489 tadj +1.769e-06 rr 96011.4 offs +0.207mS r=0.07
  ...
```

The program runs in 'setup' state until it is happy with the smooth flow of data and then switches to 'run' state. During 'setup' it looks for `r=` values less than one. If the `r=` value does not settle down the card will not go into 'run' state. In this event, note the default value of period size and use a `-Ap=` option to try different sizes. Stick to powers of two and try values 2 or 4 times the default, or 1/2 or 1/4 of the default.

The buffer size is not important because `vtcard` should be scheduled promptly to read from the soundcard driver whenever a single period is available.

Once the program enters 'run' state, it will deliver data into the output stream, in this case buffer `@test`. It will take a minute or two to do the setup. Monitor the signal level in the buffer with

```
vtstat @test
```

which will report the peak value. Adjust mixer controls so that the VLF signal is peaking around 0.2 to 0.5 which leaves a small amount of headroom.

If at full mixer gain you are unable to reach a reasonable peak signal, that indicates you need more analogue gain before the line input. If this is not available then you can use a `-g` option to apply a software gain to the soundcard samples - at the expense of dynamic range. For example, if `vtstat` is reporting a peak of 0.1 at best, then you might restart `vtcard` with a `-g4` option.

If the card can do 24 bit conversion, you will probably need to use option `-b32`. The card returns 24 bit samples in 32 bit words with the lowest significant byte set to zero.

If the soundcard conversion rate changes too fast for `vtcard` to deal with, or if the system clock makes a step change or slews too fast, `vtcard` will drop back into 'setup' state. Output will stop until the 'run' state is regained and processes taking data from the output buffer will detect a timing break when the stream restarts. To check for this you can leave `vtstat @test` running and it counts up the number of breaks detected.

When satisfactory operation is obtained, restart `vtcard` with a `-B` option to background it, and a `-L` option to create a log file.

When `vtcard` has been running for a few minutes, the `offs` value will be giving small random positive and negative values, for example

```
2011/11/18 16:43:57 run +2.345e-06 sr 96011.471 tadj +2.004e-07 rr 96011.7 offs +0.023mS r=0.01
2011/11/18 16:44:07 run +4.275e-07 sr 96011.473 tadj +3.655e-08 rr 96011.5 offs +0.004mS r=0.00
2011/11/18 16:44:17 run -1.159e-06 sr 96011.467 tadj -9.938e-08 rr 96011.4 offs -0.012mS r=0.00
2011/11/18 16:44:27 run -1.901e-06 sr 96011.458 tadj -1.625e-07 rr 96011.3 offs -0.019mS r=0.01
2011/11/18 16:44:37 run -1.906e-06 sr 96011.449 tadj -1.629e-07 rr 96011.3 offs -0.019mS r=0.01
2011/11/18 16:44:47 run +5.188e-06 sr 96011.474 tadj +4.435e-07 rr 96011.9 offs +0.052mS r=0.02
2011/11/18 16:44:57 run -1.051e-07 sr 96011.473 tadj -1.278e-08 rr 96011.5 offs -0.001mS r=0.00
2011/11/18 16:45:07 run -1.430e-05 sr 96011.405 tadj -1.222e-06 rr 96010.1 offs -0.143mS r=0.05
2011/11/18 16:45:17 run +1.212e-05 sr 96011.463 tadj +1.035e-06 rr 96012.6 offs +0.121mS r=0.04
2011/11/18 16:45:27 run +1.944e-06 sr 96011.472 tadj +1.662e-07 rr 96011.6 offs +0.019mS r=0.01
2011/11/18 16:45:37 run -4.188e-06 sr 96011.452 tadj -3.580e-07 rr 96011.1 offs -0.042mS r=0.01
2011/11/18 16:45:47 run +9.563e-07 sr 96011.457 tadj +8.483e-08 rr 96011.5 offs +0.010mS r=0.00
2011/11/18 16:45:57 run -1.591e-06 sr 96011.449 tadj -1.360e-07 rr 96011.3 offs -0.016mS r=0.01
2011/11/18 16:46:07 run +2.988e-06 sr 96011.463 tadj +2.554e-07 rr 96011.7 offs +0.030mS r=0.01
2011/11/18 16:46:17 run -1.796e-06 sr 96011.455 tadj -1.535e-07 rr 96011.3 offs -0.018mS r=0.01
```

`offs` is the residual error of the timestamped stream with respect to the system clock. `sr` is the smoothed sample rate, `rr` is the raw sample rate taken oven the previous 10 seconds.

You can use the script `vtcardplot` to graph the performance of the soundcard and timestamping. The script scans the logfile of `vtcard` to extract the sample rate and timing offset columns. You must specify the full pathname to the logfile, eg

```
vtcardplot -s800,400 /tmp/vtcard.test.log
```

produces a plot size 800 by 400 on the x11 display. A date can be given with a `-d` option and `vtcardplot` will report just that date. Otherwise it reports the whole logfile. Alternate output formats can be request with `-o` option.

```
vtcardplot -d 2011-11-18 -o 'png small' /tmp/vtcard.test.log > out.png
```

The graphs reveal the variation of souncard conversion rate. `vtcard` tracks these variations and the current sample rate is reported in the output stream in the sample rate calibration factor. If the conversion rate changes too quickly, `vtcard` will reset to `setup` state to reestablish timing. This should be avoided if possible. In one case, resets occured on windy days and the problem was traced to a draught blowing in through the air vents on an outdoor equipment cabinet and being drawn into the PC air intake. The cure in that case was to glue some foam over the soundcard's conversion rate crystal to increase the thermal time constant.

In running state, `vtcard` applies two servo loops, one to slew the output timestamp error towards zero, and the other to track the soundcard conversion rate. In `setup` state only the sample rate servo is active and this must settle first before the program will enter `run` state and enable the timestamp correction loop.

Connections

Most sound cards and audio cables use the convention:

```
Channel 1: Left, White or Black, Tip;
Channel 2: Right, Red, Ring;
```

where the colour refers to the phono (RCA) plugs often used in audio break-out cables. Don't be too surprised if you come across a cable or sound card which uses the opposite, with the channels swapped in either the analog electronics or the digital sample frame.

Higher quality sound cards often have balanced inputs. These take the form of a pair of sockets, one for left, the other for right, with each socket being a TRS jack or an XLR. Balanced inputs are much preferred for their relative immunity to ground loop interference. You may find that using a balanced input saves having to have an isolating transformer before the PC input (but you still need a transformer at the VLF receiver end of the cable).

Bus devices

ALSA assignes device numbers to USB sound cards in an unpredictable order. This means that if you have more than one USB sound device, they may not always appear with same device number when you reboot or when you remove and reconnect the device or its hub. USB devices and hubs sometimes reset themselves in response to a noise glitch and again the USB sound devices can reconnect with a new device number.

The same problem also occurs with PCI cards - a particular card will not necessarily be given the same ALSA device number after a reboot.

These problems are especially acute when you have two or more identical devices as few sound interfaces supply a serial number to the host.

To avoid these problems you can specify the physical port of the device via the `-d` option instead of using the ALSA device name. Connect your sound devices and run the command

```
vtcard -dq
```

which will produce an output something like

```
card0 pci:0000:00:05
card1 usb:1-7.6
card2 usb:1-7.2
card3 usb:1-7.4
card4 usb:2-8
```

This PC has five sound devices: one internal PCI card and four USB devices. card4 is connected via port 8 of host controller number 2. Host controller number 1 has an external hub attached to its port 7 and this hub has three sound dongles connected to its ports 2, 4, and 6. After a reboot or some re-plugging, ALSA is likely to assign the card numbers in a different order. To guarantee that a `vtcard` is using, say, the device attached to port 4 of the external hub, use the option `-d usb:1-7.4`.

If `vtcard` is running and you unplug the USB device, `vtcard` will wait for a device to be reconnected to that same port.

Timestamps and alignment

The program `vtcard` assigns timestamps to stream packets by reference to the system clock, which should be disciplined by `ntpd`. The resulting timestamps have an accuracy of about +/- 50mS. This is adequate for some applications, for example SID monitoring and casual whistler detection. Other applications may require more accuracy, for example:

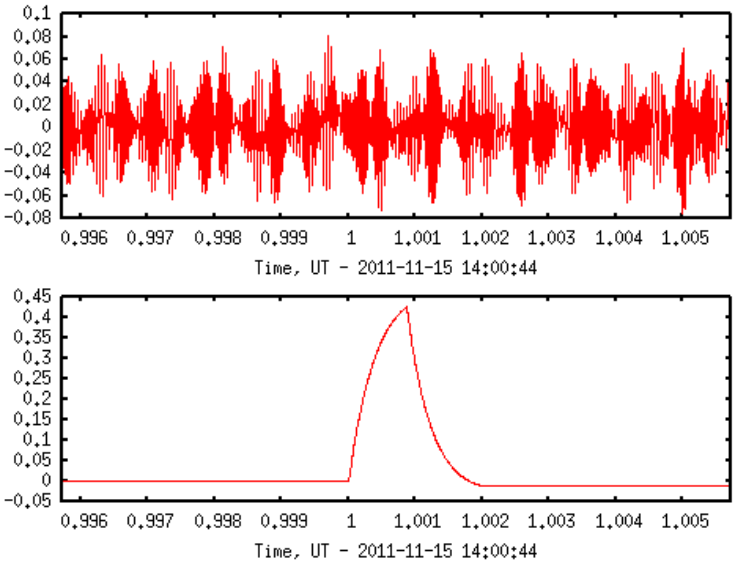
- **Stereo listening**: When signals from two soundcards are combined (`vtjoin`) into a single stream, each stream needs to be timed to better than about 5mS in order to prevent audible artifacts;
- **Arrival-time triangulation**: Estimating the point of origin of sferics, whistlers, and artificial signals, based on difference of arrival time, requires accuracy of 0.1mS or better;
- **Coherent reception**: Achieving phase coherence between two soundcards requires timestamp accuracy much better than one sample period.
- **Precision frequency and absolute phase measurements**: using `vtnspec` and `vtsid`.

For these applications, the program `vttime` will refine the timestamp by reference to timing signals within the streams. A timestamp accuracy of better than 50nS can be achieved when `vttime` is given a good quality GPS PPS reference.

Using a pulse-per-second

A channel of the soundcard must be dedicated to receiving a PPS signal. Typically a single channel of VLF will occupy channel 1 of the soundcard and the PPS will be on channel 2. You can use one of three timing pulse methods: centroid, pulse, or edge. Whichever method you use,

the PPS channel must be treated as a high quality low noise analog input to the soundcard, not a digital signal. The pulse amplitude should use most of the dynamic range of the soundcard.



Example PPS on the 2nd channel with VLF on channel 1.
The pulse has been shaped by an RC network for use with the centroid method.

Centroid `vttime` measures the centroid of the pulse, which must be suitably shaped, typically with a simple low-pass RC circuit, so that the pulse rises and falls smoothly with a well defined peak with no flat (clipped) top. The ideal GPS pulse width is around 2mS but 1mS to 10mS will work. The centroid timing method can be used with any sound card or A/D converter.

Pulse `vttime` measures the phase center of a short rectangular GPS pulse. The pulse should have a duration between 0.5 and 2 sound card sample periods, for example 10uS at 192k/sec, 40uS at 48k/sec. This method can be used only if the sound card uses sigma-delta sampling, which is the case for most if not all sound cards. This is your only option if the GPS provides a fixed short timing pulse (eg many Trimble devices).

Edge `vttime` measures the phase slope of the leading edge of a long GPS pulse. The pulse width should be about 0.1 seconds or longer. This method also relies on using a sigma-delta sound card. It is your only option if the GPS provides only a fixed duration long timing pulse (eg many Garmin devices).

With all three methods, best results are obtained if the GPS is isolated using an opto coupler such as the TLP2955. The `pulse` method is the easiest one to use and is the recommended mode if your GPS supports a short PPS width. The timing accuracy is limited by the signal/noise ratio of the sound card input but a raw timing jitter well below 100nS is normally achieved.

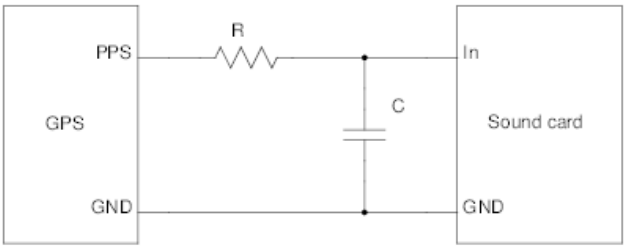
Pulse shaping for centroid method

The simplest arrangement of RC filter is shown on the right. The values of R and C can be calculated with

```
Rin = input resistance of the sound card, ohms;
Vf = full scale input amplitude of the soundcard, volts;
Vp = pulse amplitude out of the GPS, volts;
Pw = pulse width, seconds
```

then

```
R = Rin * (1.25 * Vp/Vf - 1)
C = Pw/(pi * Rin * (1 - 0.8 * Vf/Vp))
```



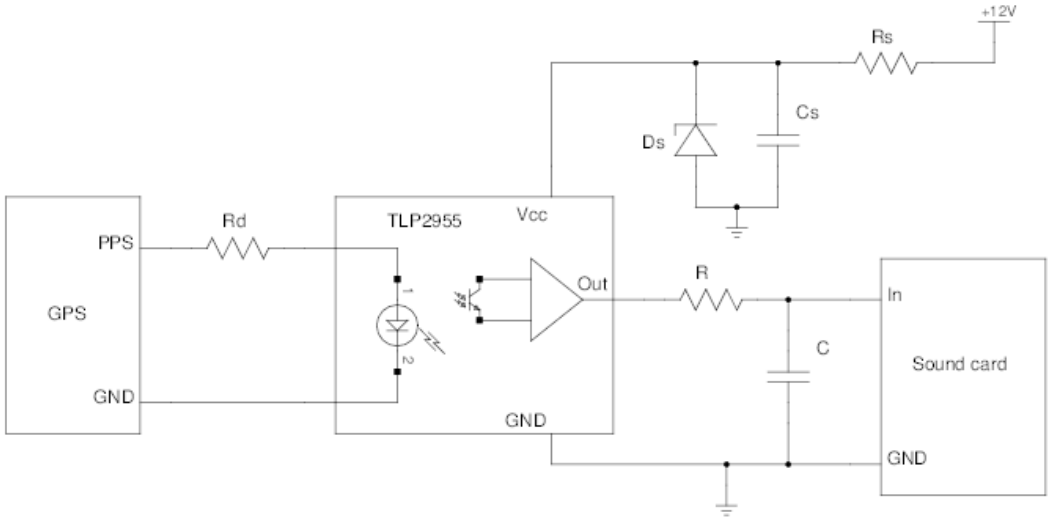
A simple RC low pass filter to shape the PPS.

The formulas aim to give a shaped pulse which peaks at about 0.8 of the sound card full scale. For maximum noise immunity, run the PPS channel of the sound card at its lowest gain if possible, and Vf should be measured at that setting.

It is desirable for the time constant of the RC shaping network to have low temperature dependence or be kept at a fairly constant temperature, as this affects the delay between leading edge and centroid and therefore directly affects the timing calibration.

This arrangement is very simple and will get you up and running quickly but may not perform very well. The problem is that when the GPS drives the PPS high, the PPS signal is connected to the supply rail of the GPS which will carry a lot of noise and ripple. This noise is then imposed on the shaped PPS which leads to an irregular centroid measurement.

An improved circuit is shown below.



An opto-isolated PPS signal.

An opto isolator with a logic level ouput is used to decouple the GPS supply from the PPS signal. Its output logic driver can then be supplied from a clean voltage rail. It is also possible to keep the GPS ground separate from the sound card ground. The R and C values are calculated as above.

The input stream to `vttime` must be reasonably well timestamped by `vtcard` which is ensured by running ntpd on its host. `vttime` uses the incoming timestamp to get a rough idea of when to look for the pulse. It is convenient to use `gpsd` as a timing source for the `ntpd`.

Timing system setup

Assume that the raw soundcard signal is available in a buffer `@raw`, with VLF on channel 1 and a PPS on channel 2. Then test `vttime` by running

```
vttime -vv -m centroid+,c=0 -c2 @raw @timed
```

The output will look something like

```
vttime: selected channel: 1 = @raw:1
vttime: selected channel: 2 = @raw:2
vttime: channels: 2, input rate: 192000
vttime: calibration offset: 0.000e+00
vttime: buffer size: 703 blocks, 29.995 seconds
vttime: break detected on ++9802: 79.869
vttime: wild PPS pulse interval 19829.855520 - skipped
vttime: st0 PPSpmr 164.4 PPSmad 0.000uS in 22.358mS out   0.000uS rate_err  3.38 inrate 192001.6895 int 192003.3791
vttime: inrate 192002.6570 int 192003.3424
vttime: st0 PPSpmr 164.5 PPSmad 0.041uS in 22.259mS out   6.639uS rate_err  0.71 inrate 192003.0122 int 192003.3674
vttime: st0 PPSpmr 164.6 PPSmad 0.055uS in 22.227mS out   3.549uS rate_err  0.40 inrate 192003.2118 int 192003.4114
vttime: st0 PPSpmr 164.3 PPSmad 0.055uS in 22.194mS out   1.537uS rate_err  0.15 inrate 192003.2889 int 192003.3659
vttime: st2 PPSpmr 164.0 PPSmad 0.055uS in 22.162mS out   0.778uS rate_err  0.08 inrate 192003.3020 int 192003.3678
vttime: st2 PPSpmr 164.1 PPSmad 0.054uS in 22.130mS out   0.949uS rate_err  0.07 inrate 192003.3138 int 192003.3729
vttime: st2 PPSpmr 164.3 PPSmad 0.068uS in 22.097mS out   0.859uS rate_err  0.02 inrate 192003.3180 int 192003.3387
vttime: st2 PPSpmr 164.1 PPSmad 0.076uS in 22.066mS out   1.173uS rate_err  0.09 inrate 192003.3333 int 192003.4100
vttime: st2 PPSpmr 163.9 PPSmad 0.074uS in 22.034mS out   1.159uS rate_err  0.05 inrate 192003.3417 int 192003.3834
vttime: st2 PPSpmr 164.0 PPSmad 0.074uS in 22.002mS out   1.186uS rate_err  0.05 inrate 192003.3501 int 192003.3924
```

The program spends a little time in state `st0` which is a setup state, then switches to `st2` which is the normal running state. The output reports a value `PPSmad` which is the mean absolute deviation of the interval between consecutive centroids. This should be quite a bit less than a uS if everything is working. `PPSpmr` is the peak/mean ratio of the PPS channel and should be 50 or 100 or more.

Allow `vttime` to run a while in foreground to see that it is stable, then restart in background, giving it a `-B` option and specify a `-L logfile`.

The output channel `@timed` contains the VLF signal on channel 1 and the timing pulse on channel 2.

For accurate frequency and phase measurement, the above is sufficient. If timestamp accuracy is required, then the centroid offset, which was set to zero above, needs to be set correctly. The setting can be improved by disconnecting the VLF and feeding the raw rectangular PPS pulse into channel 1 of the soundcard via a small

capacitor so that a short sharp pulse, just 1 or 2 samples wide, is captured. There will be two such pulses, one for the leading edge and another for the trailing edge of the PPS. Use `vtscope` or `vtplot`, or examine the ASCII data produced by `vtraw -oa` to measure the timestamp at which the the leading edge pulse on channel 1 begins to rise. This will probably be slightly offset from the stream timestamp's second mark. Adjust the `c=` value to correct for this. If the leading edge pulse starts early with respect to the stream timestamp, this means the stream timestamp is late with respect to UT and the centroid offset needs to be increased by that amount.

With the method described above, the centroid offset can be calibrated to about the nearest sample. Further improvement requires some ingenuity.

RTL2832U

Installation

On Ubuntu, you can install the librtlsdr library with

```
apt-get install librtlsdr-dev
```

Otherwise, download from https://codeload.github.com/steve-m/librtlsdr/tar.gz/v0.5.2 and build with:

```
tar xzf librtlsdr-0.5.2.tar.gz
cd librtlsdr-0.5.2
mkdir build
cd build
cmake ../
make install
ldconfig
```

You may also need to install libusb development files.

To include `vtrtlsdr`, you must configure with the option `--with-rtlsdr`.

If you find that

```
vtrtlsdr ... | vtmix -c1,-j
```

gives you lower sideband (inverted spectrum) instead of upper sideband as you would expect, then use `vtrtlsdr -q` to invert the Q signal to put things right.

Annoyingly, librtlsdr throws lots of gratuitous messages to stderr during setup. You could redirect stderr but then you lose your own log messages unless you're using `-L`. Determined users will hack into the librtlsdr source to clean it up.

Hints and Tips

Playing a stream

Use `vtraw -ow` to convert the stream to WAV format and pass this onto `aplay` or whatever raw PCM player you use. You may want to resample the stream first, filter it, or change the number of channels. For example, to play the 2-channel stream `@source` at a 44100 sample rate, you might do

```
vtresample -r44100 -q1 @source | vtraw -ow | aplay -
```

You can produce mp3 using lame. For a mono stream at 64kbps,

```
vtresample -r44100 -q1 @source | vtraw -ow | lame -b64 -mm - - > output.mp3
```

To produce a vorbis file or stream, you don't need to convert to raw first, just pipe the data through `vtvorbis -ep`,

```
vtresample -r44100 -q1 @source | vtvorbis -ep
```

You can configure `xinetd` to run any of these pipelines in response to a network connection, making an easy way to distribute a playable stream on your LAN.

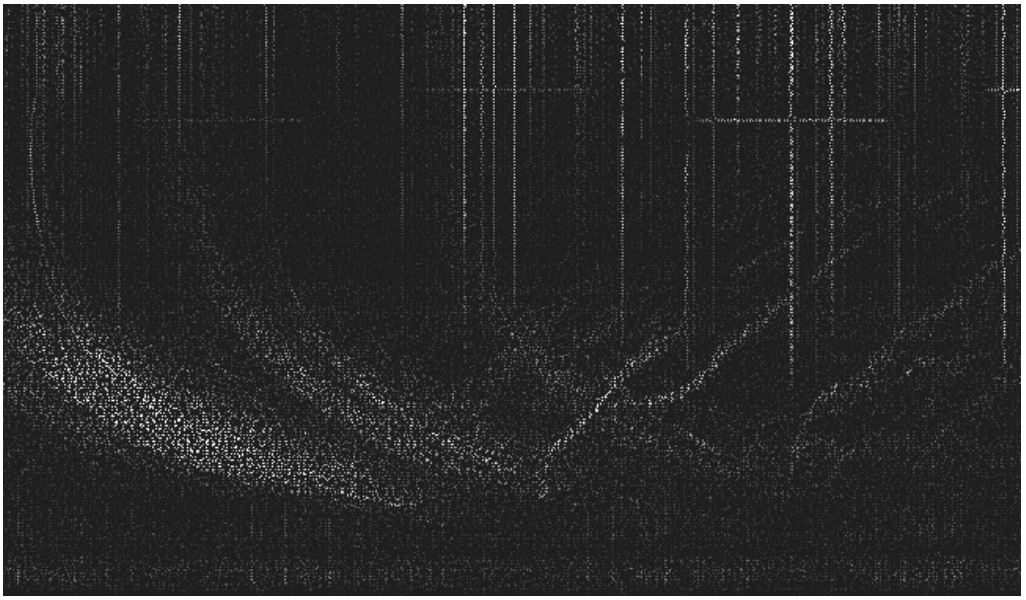
Whistler detection

This is very straightforward because `vtevent` requires little or no configuration and automatically adjusts itself to the characteristics of the input stream. It is desirable to reduce the sample rate to 32k/sec or 48k/sec and filter the hum first before presenting the signal to `vtevent`. For example, with a stream `@raw` containing a single E-field or loop signal, the following might be used,

```
vtresample -r32000 @raw | vtfilter -ath=6 | vtevent -v -d outdir
```

If orthogonal loops are available, whistler bearing and polarisation can be extracted if the program is given a `-p` option. For this to work properly, the two or three inputs must be calibrated and corrected for phase response using `vtfilter -e eqmap`. In 3-axis mode of operation, the whistler data shown here has been compiled.

Detected whistlers are captured and saved under the directory `outdir`. For each event, the program records a thumbnail spectrogram, a 6 second stream file, and a text file containing information about the event. Examples are found in the event pages under here.



A reassigned spectrogram produced by `vtevent` brings out the tendrils of triggered emission caused by this whistler.

The event detector applies principal component analysis to the short-term Fourier representation of the signal to identify regions of the F,T space that contain curvilinear features. A Hough transform then recognises when several regions combine to conform to a whistler curve. Risers are detected by the presence of sufficient regions containing rising features. The detector is not very sensitive to diffuse activity such as hiss bands or diffuse chorus.

The program normalises the input signal by looking for the VLF noise floor in between the sferics and then the signal is equalised and the dynamic range compressed with respect to the this.

Weak signal detection

When the received signal is accurately timestamped by GPS using `vttime`, it is suitable for extraction of weak narrow-band signals such as used by radio amateurs. During reception the signal should be passed through `vttime` and into raw storage via `vtwrite`. Signal detection is then carried out during post-processing using `vtread` to extract the raw data.

The tool to use for narrow band detection is `vtnspec`. It is essential to blank the sferics from the signal before passing it to `vtnspec` and in turn, to operate properly, the blanker `vtblank` must have its input stream band-limited to a width of 2 or 3kHz. Prior to band-pass filtering, any beam synthesis and steering should be performed by `vtmix` on the raw signal extracted by `vtread`.

For example, to look for a signal at 8970.00245Hz which was transmitted for one hour starting 2011-08-12 06:15 UT, you might use something like the following pipeline,

```
vtread -T2011-08-12_06:15,+3600 /rawdir |
  vtmix -c 0.643,-0.766 |
    vtfilter -h bp,f=8970,w=3000 |
      vtblank -a12 -d0 -t1 |
        vtnspec -f 8970.00245 -w0.040 -r278e-6 > spectrum.dat
```

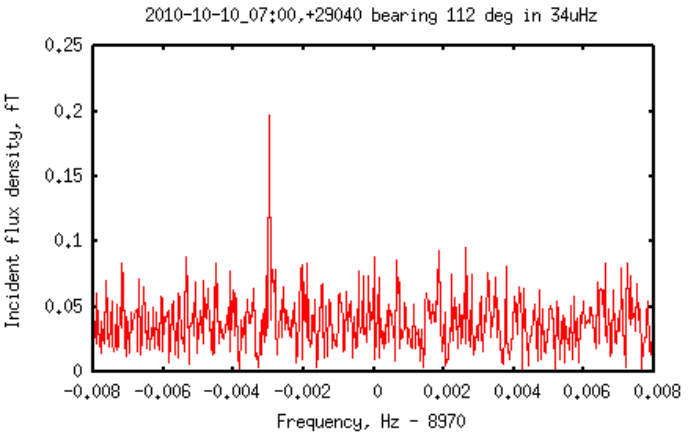
`vtread` extracts 3600 seconds of signal from orthogonal loops stored in `/rawdir` and the `vtmix` produces a single output channel synthesising a loop oriented on the desired bearing. `vtfilter` limits the bandwith to +/-1.5kHz of the operating frequency and the options to `vtblank` are reasonably optimum for normal VLF.

`vtnspec` analyses the spectrum over the range 8970.00245Hz +/-20mHz using frequency bins of width 278uHz, which is 1/3600 seconds. The output file is ASCII with frequency in column 1 and RMS amplitude in column 4.

A spectrum plot can be scripted with something like the following

```
echo "set terminal png small
      set style data lines
      set xlabel 'Frequency - 8970Hz'
      set ylabel 'Relative amplitude'
      unset title
      plot 'spectrum.dat' using (\$1 - 8970):4
  " | gnuplot > spectrum.png
```

or you can just use your favourite plotting tool interactively. The example spectrum on the right is produced by commands very similar to above. This is an eight hour integration of a 5uW transmission from DF6NM at a range of 1030km. Column 4 has been turned into fT by multiplying by the system's calibration factor.



A 5uW ERP rubidium controlled carrier from DF6NM received at 1030km range.

Storing the raw signal and then post-processing is strongly recommended, as it allows different antenna orientations, polarisations, and bandwidths to be tried.

If the signal is strong enough, columns 2 and 3 of the output of `vtnspec` can be used to report the absolute phase of the received signal. To do this, it is first necessary to establish the average frequency of the signal, then re-run the analysis using a `vtnspec` set for a single bin centered on the average frequency. Running a sequence of short, overlapping single bin 'spectra' allows the signal phase to be reported against time. This reveals the variations in path length and will identify glitches in the transmitter phase and gives some idea of the maximum symbol duration usable by differential phase shift keying.

Stream server

A simple stream server can be set up using `vtcat` and `xinetd`. Assume you have a buffer called `@source` on the server which you want to make available to other hosts on your LAN, on demand. Define a service with `xinetd` by creating a file `/etc/xinetd.d/vt-source`, containing the following

```
service vt-source
{
         type = UNLISTED
           id = vt-source
         port = 4420
  socket_type = stream
     protocol = tcp
         user = root
         wait = no
      disable = no
    only_from = 192.168.2.0
       server = /usr/local/bin/vtcat
  server_args = @source
}
```

You might want to pick your own spare port and suitable user ID. Reconfigure `xinetd` with the command

```
killall -HUP xinetd
```

Now, when anyone connects to port 4420 on the server, they will receive the output of the command

```
/usr/local/bin/vtcat @source
```

Clients can then connect to and receive the stream using `netcat`, for example,

```
nc server 4420 | vtstat
```

and so on. Of course, you can be more creative with the server command and arguments. For example you might prefer to stream vorbis encoded, in which case you might have

```
      server = /usr/local/bin/vtvorbis
 server_args = -e -q0.8 @source
```

in the xinetd service definition file.

Setting up a stream server doesn't get much simpler than this.

Storage and retrieval

It is strongly recommended to set up a signal database to store raw or raw and timed, VLF signal. Disk space is cheap these days and there is no difficulty keeping 10 days or so of raw signal. With this facility, you have several days in which to hear about interesting events such as whistlers, aurora, flares, GRBs, TLEs, meteor fireballs, earthquakes, satellite re-entries, amateur radio transmissions, and so on.

Create a directory or perhaps mount a separate filesystem, to hold the data and use `vtwrite` to build the data files. Store the data as raw as possible (unfiltered and unequalised) although if you are using a GPS timing system, it is better to pass the raw data through `vttime` first before storage. You may also want to apply `vtresample` if you decide not to store at the full sample rate of the soundcard.

Assuming you have one or more channels of data direct from the soundcard in a buffer called `@raw`, and a raw data filesystem mounted on /rawdata, then create daily data files with

```
vtwrite -B -G86400 @raw /rawdata
```

The stored format will be the same (f8,i2,i4, etc) as that of the buffer `@raw`. Maybe your raw buffer is 32 bits and you prefer to store in 16 bits, which should be satisfactory providing the VLF signal levels are using most of the 16 bit dynamic range. Then you must pass the signal through `vtcat` to change the sample format, eg with

```
vtcat -B @raw -- -,i2 | vtwrite -B -G86400 /rawdata
```

We are using `-B` options to everything here so that the programs run in background. `vtwrite` starts a new file every 86400 seconds, ie one day, beginning at midnight. It will also start a new file whenever it sees a timing break on the input stream.

It is most desirable to have the raw data storage on a different host to that running the soundcard(s). This prevents `vtcard` operation being compromised by heavy use of the retrieval program `vtread`. In this case, use a network connection to transfer the data. For example, on the machine called 'bar' hosting the soundcard and raw buffer, run

```
vtcat -B @raw ++foo,9876,i2
```

and on the machine called 'foo' which is to run the data storage, use

```
vtwrite -B -G86400 ++9876 /rawdata
```

The use of '++' here allows either machine to start up first after a reboot, and causes the connection to be re-established after a network interruption.

If you're running multiple receiver channels, it is desirable to join them into a single multi-channel stream before storage.

The data files created in `/rawdata` can be used directly as input into any of the VT programs, but in most cases portions of data will be selected by using `vtread` to extract a particular time range, which is much faster and more efficient. For example, suppose you hear that there was a well documented TLE event occurring at say 2011-08-14, 16:37:25.256, and you want to see if you have the associated sferic, then you might run

```
vtread -T2011-08-14_16:37:25.240,+50e-3 /rawdata | vtraw -oa > data.txt
```

which extracts 50mS of raw signal into `data.txt` which can then be plotted using `gnuplot`. A more refined command would run the data through `vtfilter` to remove the hum first, and this requires the data extraction to begin 10 or 20 seconds earlier to allow the automatic notch filter time to dispose of the hum. A suitable pipeline would be like

```
vtread -T2011-08-14_16:37,+30 /rawdata |
  vtfilter -ath=6 |
    vtcat -T2011-08-14_16:37:25.240,+50e-3 |
      vtraw -oa > data.txt
```

Here, `vtread` starts early and extracts 30 seconds of data which is hum filtered, and `vtcat` is used to select the exact time range to be plotted. Of course, if you're doing a lot of this, you will have the above pipeline and the plotting ready in a script. See Time domain plots for help with the plotting of short signals.

SID monitoring

The toolkit contains an advanced SID monitor `vtsid` which is capable of monitoring multiple signals and records amplitude, bearing, and absolute phase. It can also capture the full spectrum of the VLF band. Data compiled by `vtsid` is stored in a database and the data is queried using `vtsidex`.

The procedure is to combine all the received channels into a single stream, pass this through `vtfilter` to apply your phase equalisation map, and send it all into `vtsid`. For example

```
vtfilter -e eqmap @source | vtsid -vc config
```

You will have assigned a directory for storage of monitor and spectrum data, and prepared a configuration file `config` to suit your site and available stations. The buffer `@source` is expected to contain your combined signals, with timestamps polished by `vttime` if phase information is to be extracted.

A template configuration file is included vtsid.conf, - copy and edit this. A complete and up-to-date list of VLF stations is maintained by Lionel Loudet, available here.

Convenient monitoring intervals are 5 seconds for monitored channels and 120 seconds for spectrum data. If you are looking for high speed events such as LEPs, it is advisable to run a separate copy of `vtsid` configured to monitor just the relevant frequencies at a higher rate, eg 20 samples per second. In this case, distinct data directories must be used for each `vtsid`.

When `vtsid` is configured with a `monitor_interval` of zero, it will output records as often as possible and the rate is determined by the `resolution` or `bins` configuration. To obtain rapid sampling you must sacrifice frequency resolution. For example, configuring the resolution to 50Hz with

```
monitor_interval 0
resolution 50
```

produces an output record every 20mS. You will need to set options `mod=90,fast` against any MSK monitors when sampling at high rates.

The options `-m ident -t` can be used to look at the current output for the specified monitor, for example,

```
vtsidex -mNAA -t -oh10 /siddata
```

produces an output like

```
ts a1 a2 cp1 b180
2011-11-14_09:16:22.3492 1.31e-03 2.31e-04 156.6 104.6
2011-11-14_09:16:27.3839 1.31e-03 2.34e-04 156.0 104.7
2011-11-14_09:16:32.4186 1.29e-03 2.27e-04 165.1 104.5
2011-11-14_09:16:37.4533 1.32e-03 2.30e-04 152.9 104.6
2011-11-14_09:16:42.4879 1.31e-03 2.29e-04 158.7 104.6
2011-11-14_09:16:47.5226 1.32e-03 2.32e-04 154.6 104.6
...
```

which continues until you terminate the program. The `-oh10` option produces a heading every 10 records. `a1,a2` are the RMS channel amplitudes averaged over the 5 second monitor inverval. `cp1` is the MSK carrier phase (mod 180) and `b180` is the apparent bearing of the incident signal. The columns reported will depend on the configuration of `vtsid`. In this example, `vtsid` is being fed with signals from a pair of loop antennas so there are two amplitude columns and a bearing column, and the 'phase' keyword is included in the config file so the program issues a phase column. If an E-field signal is also supplied to `vtsid`, it will report three amplitude columns and a 360 degree bearing column `b360`.

A similar `vtsidex` command is used to extract historical data, with a `-T` option specifying the time range,

```
vtsidex -mNAA -T2011-10-06_23:30,+3600 -ote /siddata > data.txt
```

Here, a `-ote` option causes unix epoch timestamps to be produced which is convenient with the '%s' time format specifier in `gnuplot`. To plot the fluctuating phase and apparent bearing of NAA, you might then use `gnuplot` commands like

```
set xdata time
set time format '%s'
set format x '%H:%M'
plot 'data.txt' using 1:4 title 'phase', '' using 1:5 title 'bearing'
```

The `-t` and `-T` options can be combined, for example

```
vtsidex -mNAA -Ttoday, -t -oh20 /raw/sid
```

reports everything from the previous midnight and then continues to follow the incoming records. Note the comma after 'today' which indicates there is no closing time.

The package includes a shell script `vtsidplot` which runs `vtsidex` and plots its output data using `gnuplot`. See vtsidplot for details. You can copy this script to a new name and customise to your local requirements. For example you can modify the scripts to factor in your receiver calibrations to plot vertical axes directly in pT or uV.

If you have several monitors and you want to plot them all together, you can make a simple script to run `vtsidplot -o png` on each monitor, all with the same `-T timespec` and then use the `montage` program to produce a composite plot.

Note that `gnuplot` cannot handle fractional timestamps. Therefore when there is more than one sample per second, you must make your own arrangements for plotting. `gnuplot` can still be used but the timestamp from `vtsidex -ote` must be treated as a numeric field. One solution is to subtract the previous midnight from the timestamp and plot the remainder as a 'Seconds since midnight' x-axis. Alternatively, subtract the first timestamp from all the timestamps so that the x-axis becomes an offset in seconds. The option `-otr` tells `vtsidex` to do this.

If `vtsid` is configured to log spectrum data, you can produce spectrograms from the stored data. See vtsidgram and Spectrogram plots for details.

Time domain plots

The toolkit contains a script `vtplot` to do simple time domain plots of a signal stream. Simply run a short length of stream into `vtplot` to get a display on your X terminal,

```
vtcat -E0.01 @source | vtplot
```

Make sure you have something in the pipeline which restricts the length of the stream to a short sample. You can add a label to the X window by using

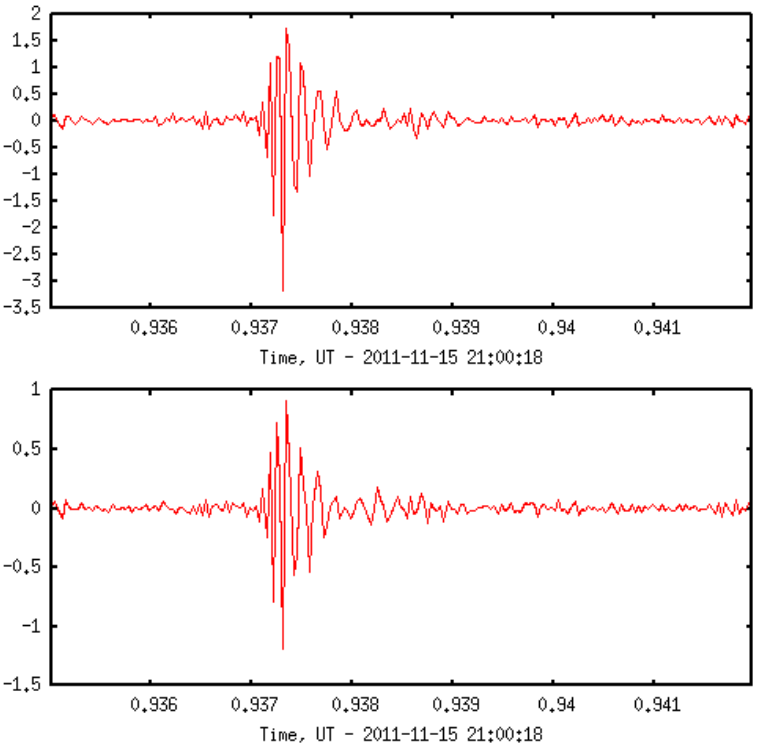
```
vtplot -o "x11 title 'My signal'"
```

The mouse acts as a cursor in the persistent display window. Type a 'q' into the window to remove it. Send the output to an image file with

```
vtplot -o "png small" > image.png
```

To customise plotting for your own requirements, for example you might want to do something different with the time axis or layout, copy `vtplot` to a new name and edit as required.

If you prefer another plotting program, or want to use `gnuplot` interactively, just use `vtraw -oa` to run the data into an ASCII file and apply your favourite tools to that.



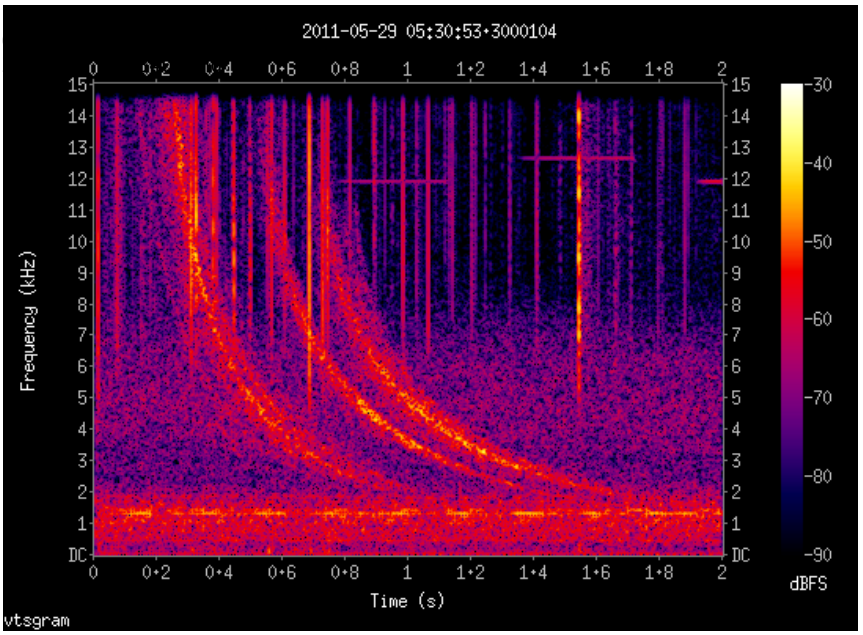
A sferic received by a pair of loop antennas, rendered by `vtplot`.

Spectrogram plots

A simple script `vtsgram` is included which produces spectrograms of short (up to a few minutes) time periods. It is a front-end for the `Sox` utility and assumes the installed version of `Sox` supports the spectrogram option. The example on the right was produced by the pipeline

```
vtread -T2011-05-29_05:30:40,+60 /r3/raw |
   vtfilter -ath=6 -g4 -- -:1 |
```

```
vtresample -q2 -r30000 |
   vtcat -T2011-05-29_05:30:53.3,+2.
      vtsgram -p200 -b300 -s '-z60 -
```



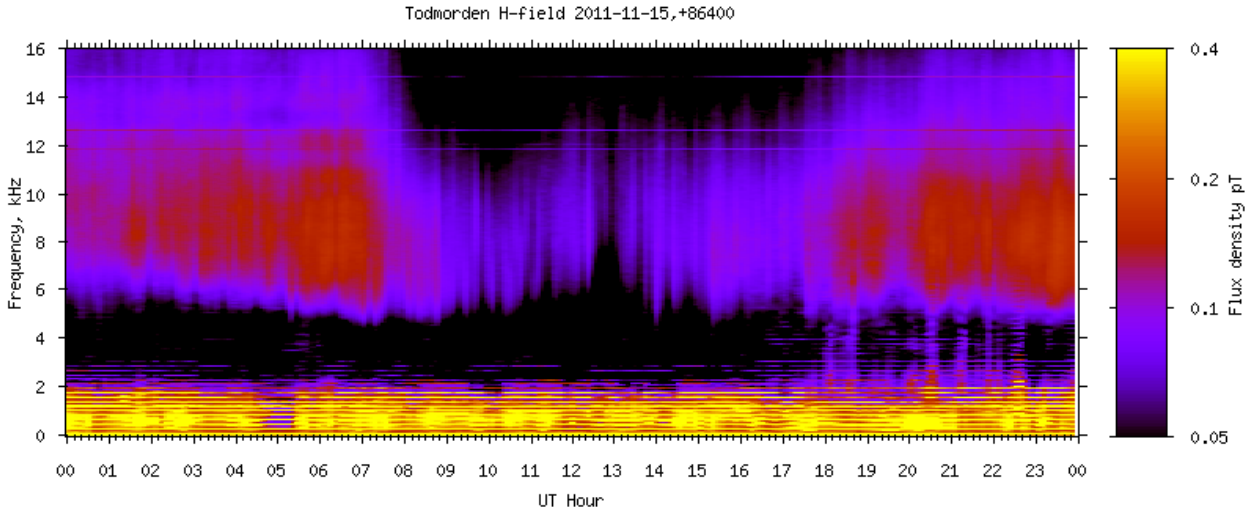
A strong whistler set against a background of natural VLF hiss.

A slightly longer than necessary section of raw data is extracted by `vtread` so that the filter can have 10 or so seconds of preamble in order to settle its automatic hum notching before the whistler comes along. The stream is resampled to 32k which sets the vertical scale of the spectrogram, and the `-T` option to `vtcat` is the one that selects the period of signal to plot.

The `-s` option allows extra options recognised by `Sox` to be passed along, in this case to set the z-axis range. These `Sox` options must be quoted into a single argument to `vtsgram`.

A second way to produce spectrograms is to use `vtwspec` or `vtnspec` to produce spectrum data and use the `pm3d` mode of `gnuplot` to do the plotting. This requires some scripting to run the spectrum analyser repeatedly, sliding an overlapping transform window along the required time axis. The resulting collection of spectrum data must be fed through awk to rearrange it for `pm3d` plotting mode. An example of this in operation is the real time spectrogram here.

A third method makes use of the spectrum data logged by `vtsid`. This is extracted by `vtsidex` for the required time range, rearranged by awk and plotted as above with `gnuplot`. This is well suited for quickly plotting long spectrograms spanning a day or several days. An example is shown below and the package contains a script `vtsidgram` for running these plots. Even longer time periods can be plotted this way by using synoptic sampling of the spectrum.



A 24 hour spectrogram, produced by `vtsidex` into `gnuplot pm3d`, shows a typical diurnal variation of sferic activity.

Multi-channel reception

The program `vtjoin` brings together two or more streams having the same nominal sample rate and assembles them into a single stream, using the timestamp and sample rate calibration information in the streams to align the inputs. Normally, each of the input streams will have had its timestamp polished by `vttime`.

Using `vtjoin`, multiple soundcards from different hosts can be combined for antenna synthesis or polarisation and bearing measurement, signals received on different sites can be merged for coherent reception or arrival time difference measurement, and streams from different signal databases can be brought together for analysis.

The streams are brought together onto one host and applied to `vtjoin`. For example, the following three background commands collect three streams, two are on the LAN and the third comes in from a remote site via

vtvorbis

```
vtcat -B ++6001 @vlf1
vtcat -B ++6002 @vlf2
nohup nc -ldk 6003 2>/dev/null | vtvorbis -B -d @vlf3 &
```

The contents of the three buffers are then combined,

```
vtjoin @vlf1 @vlf2 @vlf3 - | vtfilter -e eqmap - @output
```

and it is convenient at this point to use `vtfilter` to apply an equalisation map to deal with any phase and amplitude calibration adjustments if required. Note that an output stream must be explicitly given to `vtjoin` otherwise it would try to use buffer `@vlf3` for output.

The output buffer provides the input stream then for antenna synthesis using `vtmix`, arrival time difference measurement using the cross-correlation function of `vtcmp`, and so on.

Filtering

The general purpose filter program `vtfilter` is a frequency domain filter. The input signal is transformed to the frequency domain, multiplied by an array of arbitrary complex filter transform coefficients, then converted back to the time domain. Some simple filter transforms can be specified by `-h` options on the command line. More complex transforms can be constructed in a text file which is given to `vtfilter` with a `-e eqmap` argument. The filter transform used is the product formed from all the given `-h` options and the `eqmap`.

An automatic notch filter is built-in to `vtfilter` and activated with the `-a` option which requires a threshold argument. This is intended primarily for hum removal. An option `-a th=8` should be sufficient for light hum and `-a th=4` for more severe cases. Use the largest value that you can. Lower values will remove more hum harmonics but will increase the amount of reverberation. Values of `-a th=3` and below will sound quite unnatural. When first started, the automatic notch will take several seconds to locate the hum harmonics and then the filter will activate. An example of the hum filter being activated can be heard here. The raw hum is present for a few seconds until `vtfilter` applies automatic notches. The threshold in this example is `-a th=7` as the hum is fairly light and free of modulation sidebands.

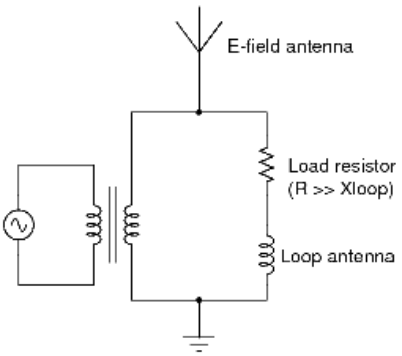
You may wish to discard the initial few seconds of output when running the automatic notch to remove hum. Just follow `vtfilter` with a `vtcat -S10` to discard the first 10 seconds of output.

Phase equalisation

To achieve antenna synthesis and coherent reception involving multiple receivers, it is necessary to compensate for differences in phase response of the receiving systems involved. The compensation is easily applied using the `-e eqmap` function of `vtfilter`. The difficult part is establishing a sufficiently good equalisation map. One useful tool for this is `vtcmp` with its `-m pd180` and `-m pd360` modes of operation. These modes compare the phase and amplitude of a two channel stream in the frequency domain and produce an ASCII output data file which reports the difference for each frequency bin.

A relatively straightforward case of phase equalisation is that of a pair of orthogonal loops. Presumably the two loops and amplifiers are identically built and so will only have small differences in their phase and amplitude responses. An end-to-end phase comparison can be performed by using a test loop placed near the antenna. The test loop carries a small signal current sufficient to produce a clear signal in the receiver outputs. Both loops should respond to this near-field signal with identical or inverted phase, depending on the location and orientation of the test loop. In this case `-m pd180` will report the residual phase difference, modulo 180 degrees.

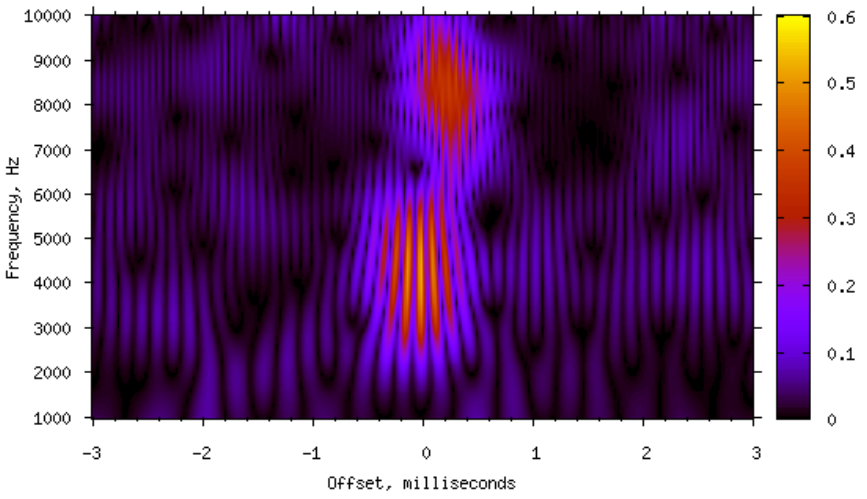
Off-air signals can also be used for this. Only distant sources should be used, as anything originating within 2000km or so will show considerable ellipticity of polarisation which produces large phase differences between the orthogonal loop signals. One option here may be to place the two loops in the same orientation while calibrating. Distant MSK signals for spot frequencies, and distant sferics for broad band calibration can be used. A suitable time must be chosen when there are no nearby (<2000km) sources of sferics. When using sferics for phase calibration, `vtcmp` must be set to do a long averaging and used in `-m pd180` mode so that it doesn't matter which quadrant the sferics are arriving from.



Correcting the phase response between co-located E-field probes and loop antennas is more difficult. A rough calibration can be performed by using distant sferics and MSK stations as described above. For more precise calibration, a test source can be arranged to produce both electric and magnetic near fields in phase with one another, as illustrated on the right. The load resistor must be rated to handle all the power from the signal source and its resistance must be much higher than the reactance of the test loop so that the loop current is in phase with the E-field antenna voltage. The test source is placed near enough to the receiving antennas to obtain a good signal in both fields. `vtcmp -m pd360` or `vtnspec` can then be used to report the received phase difference.

General scheme for a calibrator to align E-field and H-field phase response. The AC source is stepped up to a high voltage via the transformer, which produces an electric field. Current through the load resistor produces a magnetic field in the loop which is almost in phase with the electric field.

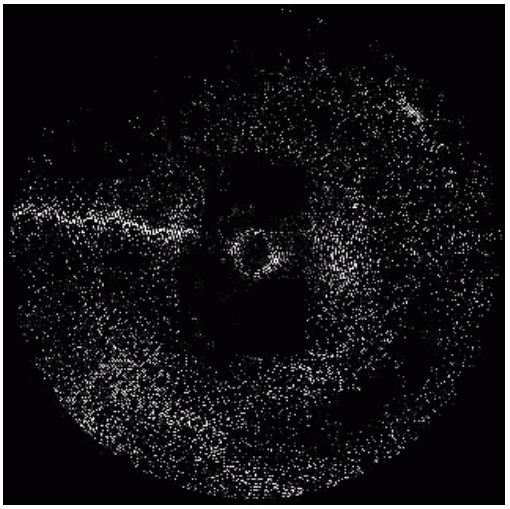
Equalising the phase of signals brought together from remote receivers is more difficult still. It requires a test source in which the test signal is synchronised to a time standard such as a PPS from a GPS. First the timestamping of the two sites must be set as carefully as possible via the `c=` parameter of `vttime` at both sites. There may be a residual systematic time difference which should be identified and corrected by a `vtcat -a` applied to one of the sources. The `vtcmp` program can again be used for these timestamp calibrations, in its `-m cor` mode of operation. This reports the cross-correlation between two channels. To confirm time alignment to within a fraction of a sample period it is necessary to upsample (`vtresample`) to a much higher sample rate before using `vtcmp -m cor` and fairly long averages should be taken. Once the time alignment of the two sites is reasonable, a phase comparison and correction can be made. It is probably necessary to repeat the time delay and phase correction steps iteratively, perhaps by first establishing a good time and phase alignment at a low test frequency and then working upwards through the range of interest to equalise the phase. A sanity check on the alignment between remote sites can be made by measuring arrival time differences of suitably placed distant MSK stations. `vtcmp -m cor` is a very good tool for ATD measurements. The image on the right shows arrival time differences of sferics and a whistler received at two sites.



Arrival time differences of whistler and sferics between two sites 30km apart, produced by `vtcmp -m cor` and rendered with `gnuplot pm3d`

Polar displays

If crossed loop receivers are available, or crossed loops and an E-field receiver, then `vtpolar` can be put to good use. The phase response of the receiving systems must be matched over the frequency range to be displayed and this can be achieved with the `vtfilter -e eqmap` function.



An example is available at this link which displays about 35 minutes of activity captured with 3-axis reception on 23rd Jan 2018. North is at the top with the center DC and the perimeter at 24kHz. Circularly polarised whistlers are arriving almost directly from above. The flashes in the north-east are beeps of the Alpha navigation system and there is sferic activity to the west and south west.

A script to retrieve raw signal data, filter and eq, and generate this AVI goes something like

```
ts=start_timestamp    # Begin at this timestamp, integer seconds
secs=length           # Length in seconds
```

```
vtread -T$((ts-30)), /raw |
  vtfilter -a th=5 -h hp,f=400,poles=2 -e eqmap -g8 |
    vtcat -T$ts,$((ts+secs)) |
      vtpolar -f vbr=500,abr=128,vcodec=h254 -s500 \
              -k0,r24000 -mf -am -gv=10 -ga=10 -p96,6,E  > movie.avi
```

A frame from the avi output of `vtpolar -mf` with 3-axis VLF reception. A sferic is arriving from the west and the zig-zag line reveals the 'polarisation error' of the bearing as a function of frequency. The central ring is some mains hum and the blob to the north-east is a beep from the Alpha navigation system.

Raw data extraction is started 30 seconds early to give a bit of preamble for the hum filtering autonotch and then a `vtcat` crops the exact time period to display. The `-p96,6,E` indicates that the first channel is a loop oriented on 96/276 deg, the second channel is oriented 6/186 degrees, and the third channel is the vertical electric field.

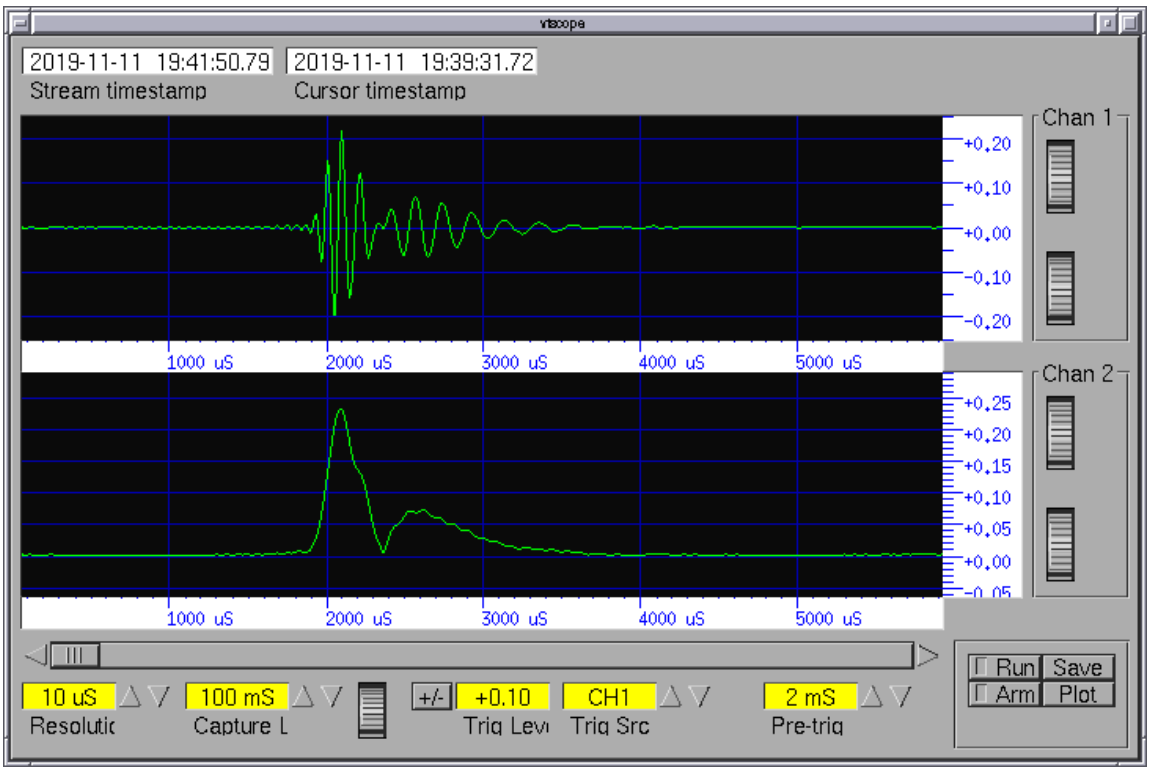
The `ffserver` uplink option is currently broken.

Analytic signals

It is easy to generate an analytic signal using `vtmix`, for example

```
vtmix -c1 -c-j
```

turns a single channel stream into a two-channel stream in which both channels are a copy of the input signal, channel 1 is an exact copy and channel 2 has all frequency components delayed by 90 degrees.



`vtscope` display with the original signal on channel 1 and the magnitude of the analytic signal on channel 2.

A most useful application for the analytic signal is to extract the envelope of a non-stationary signal. An example is the sferic waveform shown in the `vtscope` image on the right. Channel 1 is the original waveform and channel 2 is the magnitude of the analytic signal, obtained using `vtam`. The following command was used,

```
vtmix -c1 -c-j @vlf |
   vtam |
   vtjoin @vlf - - |
   vtscope
```

Extracting the envelope in this way allows the intensity and timing of an oscillatory signal to be examined without worrying about its phase. Useful for timing sferics, whistlers, and the edges of timing signals such as MSF and DCF.

Antenna synthesis

Signals from several receivers can be combined in order to synthesise a desired response pattern. The program `vtmix` performs the required mixing but first the signals must be brought together into a single stream and have their phase and amplitude responses equalised. See Multi-channel reception and Phase equalisation for details. In the following, assume this has been done and the combined signals are available in a buffer called `@source`. This stream might contain a pair of orthogonal loop signals and one or more E-field signals.

In these notes, the convention is used that in right-hand circular polarisation, the field vectors are rotating clockwise as seen from the source looking towards the receiver.

As a simple example, suppose `@source` contains two channels, one from each of two E-field receivers. Adding their signals will double the signal amplitude but only increase the system noise by sqrt(2). The command

```
vtmix -c0.5,0.5 @source @output
```

produces the summed output with the coefficients of 0.5 restoring the original signal amplitude.

A synthesised loop signal is derived from a pair of orthogonal loop signals with a command such as

```
vtmix -c0.866,-0.5 @source @output
```

If ch1 is an E/W loop and ch2 is a N/S loop, and the loop polarities are such that a signal from the north east is in phase in both loops then this will synthesise a loop oriented along 120/300 degrees. In general the coefficient of the E/W loop should be sin(bearing) and that of the N/S loop, cos(bearing).

The synthesised loop has a dipole response and but this can be made unidirectional if `@source` contains an E-field channel with its amplitude and phase correctly equalised. If loop polarities are as above with the additional requirement of being in phase with the E-field given a signal incident from the north-east, the command

```
vtmix -c0.866,-0.5,1 @source @output
```

will produce a uni-directional response along the bearing 120 degrees. Loop signals from the back of the beam will be equal in amplitude and opposite in phase to the E-field signal and be cancelled.

Circular polarisations can be synthesised from the orthogonal loop signals by giving imaginary coefficients to `vtmix`. A coefficient of 'j' advances a signal by 90 degrees and '-j' retards a signal. Adding a delayed N/S signal to the undelayed E/W signal produces a RH circular response,

```
vtmix -c1,-j @source @output
```

or in terms of magnitude and phase coefficients,

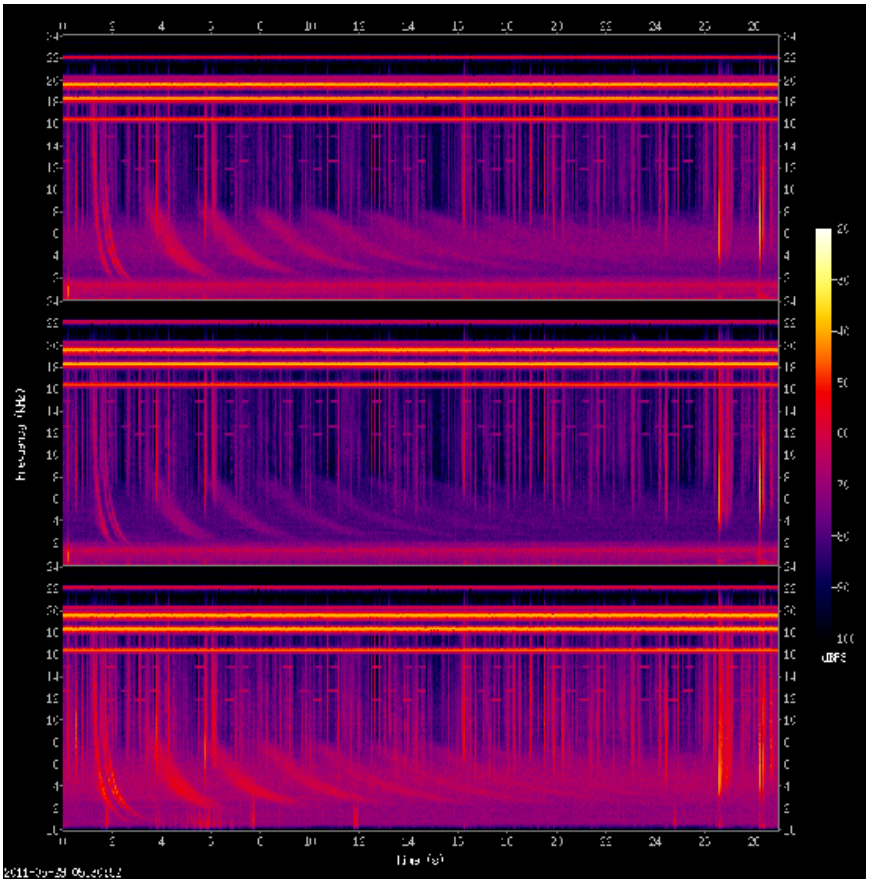
```
vtmix -c1,1/-90 @source @output
```

A useful mix for looking at whistlers is to synthesise both polarisations to produce a 2 channel output, with

```
vtmix -cj,1 -c1,j @source @output
```

Whistlers are normally either strongly RH or strongly LH polarised and therefore when the above dual circular stream is plotted on a spectrogram (`vtsgram`) one or other of the channels will be significantly stronger and may also have a better signal/noise ratio than the original linear polarised loop signals.

The spectrogram on the right was produced using `vtmix` to generate the three signals and was rendered by `vtsgram`. This whistler and its echo train was strongest in RH polarisation. The hiss is also RH polarised.



A whistler train and hiss band, seen in three different polarisations. From top to bottom: RH circular, LH circular, vertical E-field.

Startup scripts

Hosts which are running `vtcard` will have to run `amixer` or `ossmix` to select and un-mute the line input and set the PCM input gains. They must also ensure that the system clock is set and ntpd is running.

After `vtcard` is started it may take up to a minute or two before its output lock-free buffer is created. Before launching programs which use the raw buffer, startup scripts should run `vtwait` on the buffer in order to wait for

the buffer to appear, otherwise programs attempting to use the buffer will fail immediately. For example,

```
vtcard -Bvv -d hw:0,0 -b32 -r 192000 @raw,20,i4

echo "waiting for @raw..."

vtwait -t @raw || {
   echo "problem with buffer @raw" >&2
   exit 1
}

echo "@raw running"
vtcat -B @raw ++somewhere,someport,i4
```

The script blocks on the `vtwait` until `@raw` is created and data appears.

It is likely that signal processing will be distributed across more than one host. Typically one or more hosts will run `vtcard` and possibly `vttime`. Another will assemble and store data with `vtjoin` and `vtwrite` and will probably host various other programs which operate on the live stream and process the output of `vtread`.

It is desirable that the routine processing pipelines will not collapse if one or more of the hosts involved is rebooted, and if they are all starting up, the pipelines should assemble and begin work regardless of the order in which the hosts start up. With this in mind, LAN stream connections between hosts should use the ++ syntax to specify the network connections. WAN connections would need to have loops around ssh commands in order to reestablish contact. A `vtvorbis -e` uplink to Icecast servers should use the `-k` option and `vtvorbis -d` programs need to be put within a retry loop.

U-blox GPS

U-blox GPS modules are ideal for timing VLF signals, are readily available at low cost, reliable and well documented, and are probably the most popular for this application. The program `vtubx` is included as a convenient means to set up the GPS to the required mode of operation. `vtubx` is scriptable and avoids having to find a Windows PC to run the u-center program.

The program supports the following u-blox series:

Series	Supported GNSS
6	GPS
7	GPS or GLONASS
8	GPS, GLONASS, Galileo, QZSS, BeiDou

The 'T' versions, such as the M8T, support a fixed-position time mode which offers the best performance and is ideal for VLF timing. The non-T versions have more than adequate performance.

Modules are available with NEO or LEO prefix:

NEO	Built-in LNA, can be used with active or passive antennas
LEO	No built-in LNA, must be used with active antennas

If possible, choose the NEO range.

To use `vtubx` you must first disable `gpsd` if it is running, to release the serial/USB port.

To activate time mode on modules that support it, initiate a survey-in with a command such as:

```
vtubx -T 24.0,5.0 -S /dev/ttyACM0
```

This will make the survey-in average the position for 24 hours (to examine the full diurnal) and it will continue to survey-in until the position is known to within 5 metres, representing a timing accuracy of about 17nS.

When the GPS decides survey-in is completed, it will then switch to time mode. `vtubx -T?` will then report time mode active.

Lightning location

Locating the source lightning of sferics involves two programs: `vttoga` to measure the arrival time (and possibly the bearing) of the sferics at at least three sites, followed by `vtspot` to perform the trilateration.

`vtspot` is run on a central computer which receives either baseband VLF signal, or the output from `vttoga` via some network connection. The `vttoga` may be run on the receiver sites themselves, or if the sites send the baseband signal back, it can run on the central computer. The choice depends on the available network bandwidth. If possible it is better to retrieve the VLF signal from the receiver site - it can be used for other purposes and can be monitored for quality.

The process only requires the 'H' records from `vttoga`, but the 'S' and 'T' records can also be returned if required by the application.

It is convenient to process TOGAs in batches of a few minutes duration, using the `-G` option of `vttoga`. For example

```
vttoga -G 300 -i 6 -F 6000,16000 -d /togas @filtered
```

will create a new output file in the /togas directory at every 5 minute UTC boundary, ie HH:00, HH:05, and so on. The output files will have name format /togas/YYMMDD-HHMMSS.

If the `vttoga` are running on receiver sites, the output files will need to be transferred over a network back to a central computer. The best command for this purpose is `rsync`, which uses ssh for transport. The receiver computers should have ssh keys set up so that they will accept login from the central computer without password.

TOGA files can then be transferred periodically, with a command on the central computer such as

```
rsync -a user@site1:/togas/ /togas/site1
rsync -a user@site2:/togas/ /togas/site2
...
```

where /togas/site1 is a site-specific destination directory. The `-a` option is important to preserve the timestamp of the TOGA files. Note that the source directory is terminated with / but the destination is not. The `rsync` command will transfer both completed files and partially completed files.

The retrieved TOGA files must then be presented to `vtspot` running in its 'matching' mode, using `-m` options to specify the TOGA batch file for each site. A typical `vtspot` launch script might look like

```
batch=180501-120500
OPTS=""
for site in $(ls /togas)
do
   file=/togas/$site/$batch
   [ -s $file ] && OPTS="$OPTS -m $site=$file"
done

vtspot -c0.9872 -v -o iso -o ext -f envelop=180 -f nearmax=12e3 -n6 -r1.7 $OPTS > output-$batch
```

The velocity factor of 0.9872 seems to work best for global lightning location.

The TOGA detection threshold at the receiver sites should be set so that the total rate of sferics from all the sites does not exceed about 170 per second. If the aggregate sferic rate is much higher than this, `vtspot` will not be able to reliably determine which sferics belong to which lightning stroke (too many overlaps) and the output will begin to include isolated false solutions randomly distributed over the globe. The `vttoga -r` option can be used to set the average TOGA measurement rate per site.

Successfully located lightning strokes can be listed from the output file by grepping the 'A' records:

```
grep ^A < output-180501-120500
```

and the 'G' records provide a summary of the performance of each site:

```
grep ^G < output-180501-120500
```

The output format of `vtspot` is designed to make it easy to work on the results using simple scripts.

You need a lot of receivers for lighting location to work well. At the very least, vtspot needs TOGAs from six receivers and these should surround your region of interest. Indeed, `vspot` when running in matching mode, will only output a stroke solution if it lies within a polygon formed by at least six receivers.

It often happens that you want to locate the source of a particular sferic. It is usually possible to identify the sferic manually in the VLF recording, especially if it is a prominent sferic associated with something like a TLE. In this case you can often estimate the location using only three TOGAs, which you can determine by running short samples containing the sferic through `vttoga`. The measurement set for `vtspot` would then resemble

```
vtspot T/site1/1522493298.069741 T/site2/1522493298.065848 T/site3/1522493298.069540
```

`vtspot` will always produce two solutions for a case like this because the ATD hyperbolas for the two independent baselines will intersect in exactly two places. In most cases it is clear which is the correct solution.

If you find that `vtspot` produces no output for your measurement set, then it is likely due to one of the arrival time differences exceeding the light travel time of a receiver pair baseline. Re-run with a `-v` option to see which is the problematic baseline. You may find that you have manually chosen a wrong sferic, or a TOGA measurement is poor because the sferic waveform is distorted for some reason, eg receiver overload or a multi-path effect. This often happens with a powerful stroke at short range. Your best option then is to try to measure (from a time domain plot) the earliest arrival time of each sferic, and use those instead of the TOGAs.

Paul Nicholson vt12@abelian.org
