RemoteLab Toolbox User Manual

v1.0

A. Perinelli
L. Ricci

October 2020

Contents

1 Overview & License 2

2 Download & Setup 4

3 *Using the toolbox:* xoscilloscope 6

4 *Using the toolbox:* wavex 10

A Calibrating the sound card 12

B *Digital* and *Voltmeter modes*: analyzing DC signals 15

C Troubleshooting 17

D Technicalities 18

1. Overview & License

RemoteLab is an open-source Linux toolbox made of several intercommunicating programs that provide the following **two tools to carry out basic experiments in electronics.**

- An **oscilloscope** to acquire signals from the microphone input of a computer's sound card.
- A **waveform generator** to generate signals and deliver them to the headphones/speaker output of a computer's sound card.

The acquisition and generation of signal is bounded by the physical limits of the available sound card. Most notably, the sampling rate is usually 44.1 kHz, and the card's frequency band is approximately limited to the frequency range between 20 Hz and 20 kHz.

The package is developed for Linux. To access the sound card, **RemoteLab** relies on the **ALSA** Library API (Advanced Linux Sound Architecture) ↗ . ALSA is part of the Linux kernel and is therefore be available on any recent (> 2005) Linux system. The plotting engine used by the **RemoteLab** oscilloscope to display data is **gnuplot** ↗ , an open-source, command-line plotting program. The graphical user interface (GUI) to control the oscilloscope and the generator is built on the **wxWidgets** library ↗ , an open-source cross-platform GUI library[1].
Please refer to Part 2 ("Download & Setup") for detailed information on how to properly set up these software requirements.

Licensing

This toolbox, all the included source code, documentation and examples are released under the **GNU General Public License (GPL) v3**. A copy of the license is provided in the package root folder. In few words, the GNU GPL license implies that the software is free to use and to modify. Please notice that, as the license clearly states, the software comes with **NO WARRANTY. The entire risk as to the quality and performance of the program is with you (the user). Should the program prove defective, you (the user) assume the cost of all necessary servicing, repair or correction.** Moreover, in **no event** will any copyright holder, or any other party who modified and/or conveys the program, **be liable to you for damages**. To sum up, use this software **at your own risk!**
Regarding the auxiliary libraries and programs,

- **ALSA** is released under the GPL (GNU General Public license) and the LGPL (GNU Lesser General Public License) ↗ .
- **gnuplot** is released under the *gnuplot license* ↗ .
- **wxWidgets** is released under the *wxWindows Library Licence* ↗ .

[1] On Linux/Unix distributions, wxWidgets is usually built against GTK ↗ .

Package authors

Alessio Perinelli

Department of Physics, University of Trento, 38123 Trento, Italy

alessio.perinelli@unitn.it

Leonardo Ricci

Department of Physics, University of Trento, 38123 Trento, Italy

CIMeC, Center for Mind/Brain Sciences, University of Trento, 38068, Rovereto, Italy

leonardo.ricci@unitn.it

Nonlinear Systems & Electronics Lab website: nse.physics.unitn.it

2. Download & Setup

Requirements

RemoteLab is released for Linux systems. Because of the fairly general requirements, the toolbox should run on any reasonably recent Linux-kernel-based system that provides **gnuplot** as well as bindings to the **wxWidgets** library. The toolbox was successfully tested on Ubuntu versions 16.04, 18.04 and 20.04.

RemoteLab relies on the computer's sound card to acquire and generate signals. Any reasonably recent (> 2000s) desktop or laptop should provide such hardware. In order to work at its full potential, the package should be run by a computer having both a microphone (input) and a headphones (output) *stereo* jack socket (3.5mm ones). In most desktop motherboards, these jack sockets are identified with colors: the microphone (input) socket is pink, while the headphones (output) one is light green. Many recent laptops only have a single socket, acting both as input and as output (a "headset" socket). In this case, the two oscilloscope channels will be *identical*, thus effectively providing a single-channel oscilloscope.

Download

RemoteLab can be downloaded at https://github.com/LeonardoRicci/RemoteLab. The repository can be either manually downloaded or directly cloned within the current directory by means of the following command (`git` is required)

```
        git clone https://github.com/LeonardoRicci/RemoteLab
```

If the repository was manually downloaded, it is necessary to unzip it by typing

```
        unzip RemoteLab-master.zip
```

Configuring the system

A Linux system has to be configured before installing **RemoteLab**. In particular, libraries have to be installed. To do so, the following commands have to be entered within a terminal; root permissions are required to run these commands (hence the `sudo` keyword). Please note that these command are reported for Ubuntu/Debian distributions, which rely on the `apt` package manager, and they have to be adapted in the case of other distributions.

First, add the universe repository and update the package list (just in case):

```
sudo add-apt-repository universe
sudo apt update
```

Install the GNU C compiler, the development version of the ALSA library and gnuplot, respectively:

```
sudo apt install build-essential
sudo apt install libasound2-dev
sudo apt install gnuplot
```

The development version of wxWidgets is required. First, a CodeLite public key is required:

```
sudo apt-key adv --fetch-keys http://repos.codelite.org/CodeLite.asc
```

The following code snippet reports commands for several examples of Ubuntu/Debian releases: check the wxWidgets website for other distributions, as well as for other wxWidgets versions.

```
sudo apt-add-repository ...
... 'deb http://repos.codelite.org/wx3.0.4/ubuntu/ xenial universe' # Ubuntu 16.04
... 'deb http://repos.codelite.org/wx3.0.5/ubuntu/ bionic universe' # Ubuntu 18.04
... 'deb http://repos.codelite.org/wx3.0.5/ubuntu/ focal universe' # Ubuntu 20.04
... 'deb http://repos.codelite.org/wx3.0.3/debian/ jessie libs' # Debian 8 (Jessie)
... 'deb http://repos.codelite.org/wx3.0.4/debian/ stretch libs' # Debian 9 (Stretch)
... 'deb http://repos.codelite.org/wx3.0.5/debian/ buster libs' # Debian 10 (Buster)
... # For other versions / distributions, check wxWidgets website
```

Finally, update the package list and install all the packages:

```
sudo apt update
sudo apt install libwxbase3.0-0-unofficial libwxbase3.0-dev
sudo apt install libwxgtk3.0-0-unofficial libwxgtk3.0-dev
sudo apt install wx3.0-headers wx-common
```

Compiling & installing the toolbox

If all required tools are properly installed, compiling the package only takes few commands to be run within the package root directory. First, reach the directory; then, check that all requirements are properly set up through the `check` script:

```
        ./check
```

The script prints some messages on the console to highlight which requirements are missing. If all requirements are met, you can compile the package:

```
        make
```

Compilation creates executables in the temporary `build/` directory. You can manually copy those executables within a directory of your choice, or you can automatically create an `installed/` directory within the package folder and automatically link executables in `~/bin/`, if available. To do so, type

```
        make install
```

Please note that the install procedure creates executables under `RemoteLab/installed/`, which are thereupon linked into `~/bin/`. Unless a different setup procedure is followed, deleting the `RemoteLab` directory will also remove the package from the system. Please also note that, if the `~/bin/` directory does not exist, the `make install` command will fail, and you will have to manually setup the executables to be accessible from anywhere within the system. In addition, you should check that the `~/bin/` directory is included into the path. This inclusion is granted by appending the following lines to `~/.profile`:

```
if [ -d "$HOME/bin" ]; then
        PATH="$HOME/bin:$PATH"
fi
```

3. *Using the toolbox:* xoscilloscope

The oscilloscope tool, **xoscilloscope**, collects data from the sound card ADC connected to the sound card microphone socket. Acquisition is initialized by setting the device at 44.1 kHz sampling rate, 16-bit stereo (two channels).

Launching the oscilloscope

The oscilloscope tool, **xoscilloscope**, is started and stopped by calling its command line launcher, `xoscilloscope-launcher`.

- To start the oscilloscope, open a terminal and type

  ```
  xoscilloscope-launcher start
  ```

  The programs making up the xoscilloscope tool print some diagnostic messages on the terminal from which the tool was launched. Nevertheless, this terminal can be closed after startup.

- To stop the oscilloscope type in a terminal **within the same directory where the oscilloscope was started**

  ```
  xoscilloscope-launcher stop
  ```

If everything is set up properly, you should be able to run the oscilloscope tool from anywhere within your system. If not, please check that executable files are properly installed (see Appendix C). Please note that you cannot stop `xoscilloscope` by calling the launcher from a directory that is not the one where `xoscilloscope` was started from. Please also note that "manually" stopping the programs (e.g. via `kill`) should be avoided, as some sub-processes might slip away and continue to run in the background.

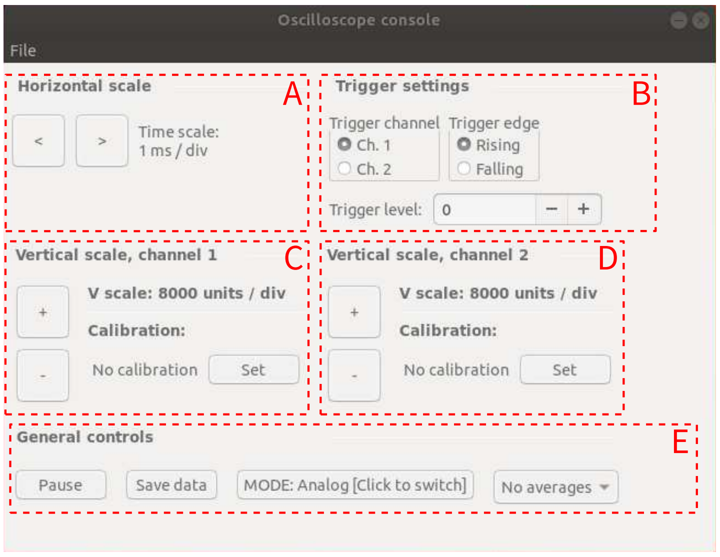
Upon startup, xoscilloscope generates two windows. The first one (referred to as *trace panel*) displays the oscilloscope traces and is created by gnuplot according to the `set terminal x11` command. The second one (referred to as *control panel*) provides oscilloscope controls.

Control panel

An instance of the control panel [1] is shown in Fig. 3.1. The panel is made of five sectors, labeled by A, B, . . . , E in the figure.

Horizontal (time) scale controls (A). The two buttons allow to change the horizontal time scale used to acquire and display traces. There are 16 available settings (in seconds per division), from 50 μs/div (screen covers 0.7 ms) up to 5 s/div (screen covers 70 s).

[1] The graphical aspect of this window might vary according to your system version, your GTK version, etc.



Time scale: 1 ms / div

Trigger channel: Ch. 1 [✓] / Ch. 2; Trigger edge: Rising [✓] / Falling

Trigger level: 0

V scale: 8000 units / div

Calibration: No calibration

V scale: 8000 units / div

Calibration: No calibration

MODE: Analog [Click to switch]

No averages

Figure 3.1: Control panel of **xoscilloscope**.

Trigger controls (B). These controls allow to change the channel, the edge direction and the level used by the triggering system. The oscilloscope *trigger* mimics the one that of any oscilloscope altogether, albeit at a software level.

Vertical (voltage) scale controls, channel 1 (C). The two buttons allow to change the vertical scale used to display the trace of channel 1. There are 12 available settings (in sample units per division), from 4 units/div up to 20000 units/div. If no samples → Volts calibration is available ("No calibration"), vertical axis displays numbers as acquired by the card ADC, i.e. 16-bit signed integer numbers. The "Set" button allows to insert a calibration for the channel, namely an integer number corresponding to 1 Volt. If such calibration is available, the oscilloscope shows data in Volts and the vertical scale settings are updated accordingly: the 12 available settings (in Volts per division) go from 1 mV/div up to 5 V/div. Calibration is not shared between channels.

Vertical (voltage) scale controls, channel 2 (D). The two buttons allow to change the vertical scale used to display the trace of channel 2. The available settings are the same as those for channel 1. Calibration is not shared between channels.

General controls (E). These controls allow to: (1) pause/run the oscilloscope; (2) save data to file; (3) switch the oscilloscope operating mode; (4) set the number of averages, in Analog mode only.

When the "Save data" button is clicked, the oscilloscope is paused: the data acquired *just after* the click are saved. The output file format consists in three columns (time in seconds, channel 1 voltage, channel 2 voltage) separated by a tabulation character. After the file is written, the oscilloscope resumes normal operation.

The available modes are cycles through by means of the dedicated button.

- *Analog mode*: the basic behavior you would expect from any oscilloscope. Averages can be used.
- *X-Y mode*: displays channel 1 on the x-axis and channel 2 on the y-axis. In this mode, horizontal time scale controls remain active, and they set the *acquisition time* used to record traces before displaying them (this time is equal to $14\times$ time per division).

- *Digital mode*: allows to visualize digital signals modulated by a chopper circuit (see Sec. B) as logic levels (0, 1).
- *Voltmeter mode*: displays, for each channel, the DC voltage as reconstructed after modulation by a chopper circuit (see Sec. B). In this mode, horizontal time scale controls remain active, and they set the *integration time* used to measure the DC voltage (this time is equal to $14\times$ time per division).

Trace panel

An istance of the trace panel [2] is shown in Fig. 3.2. The panel mimics a standard oscilloscope

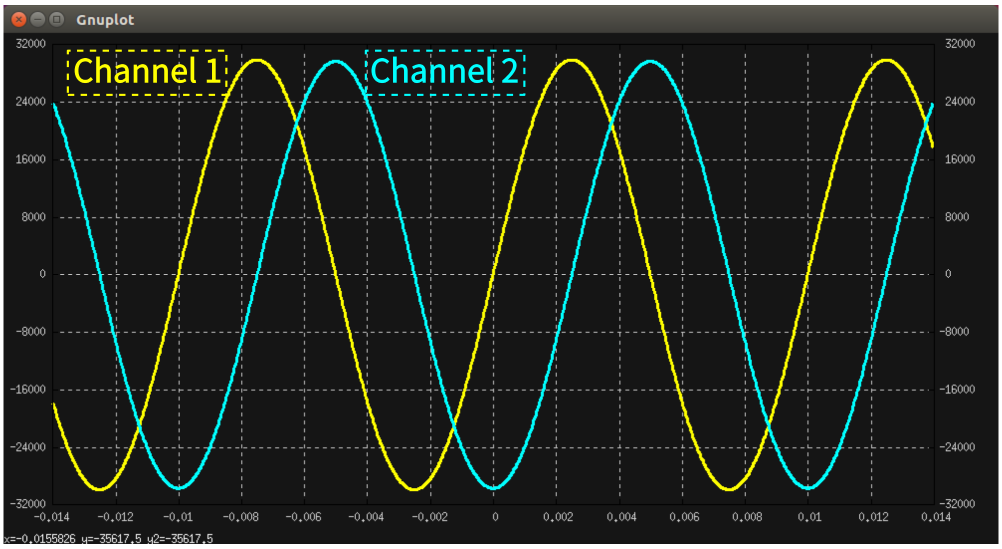


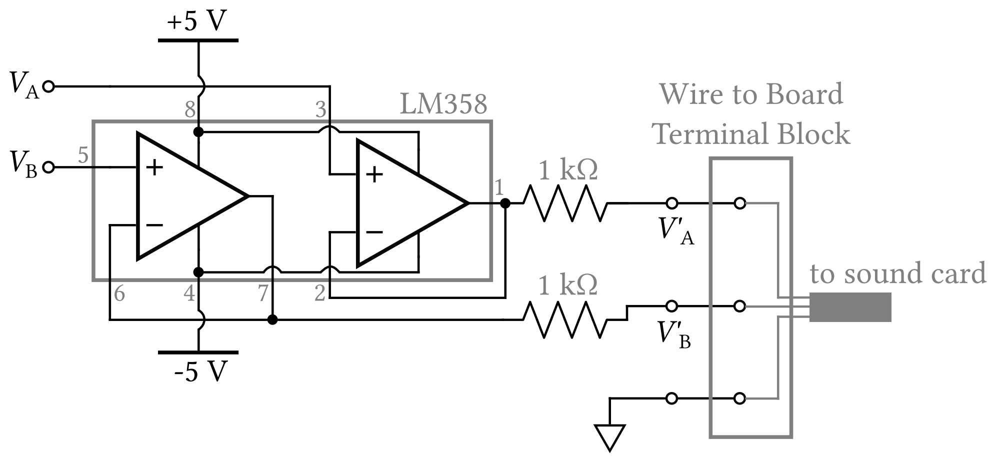
Figure 3.2: Plot panel of **xoscilloscope**.

screen, with 14 horizontal divisions and 8 vertical divisions. On the horizontal scale, time is displayed in seconds. On the vertical scale, voltage is displayed in Volts. The leftmost vertical scale refers to channel 1, while the rightmost one refers to channel 2. The trace corresponding to channel 1 is displayed in yellow, while the trace corresponding to channel 2 is displayed in cyan.

[2]The graphical aspect of this window might vary according to your system version, your gnuplot version, etc.

Buffer/protection circuit

The following diagram shows a **suggested** setup to be installed on a breadboard: the circuit provides impedance matching and protects sound cards from excessive currents. The suggested LM358 chip conveniently provides two op-amps in an 8-pin package.



4. *Using the toolbox:* wavex

The waveform generator tool, **wavex**, delivers data to the sound card DAC connected to the sound card headphones socket. Playback is initialized by setting the device at 44.1 kHz sample rate, 16-bit stereo (two channels).

Launching the generator

The waveform generator tool, **wavex**, is started by calling its standalone executable, `wavex-generator`:

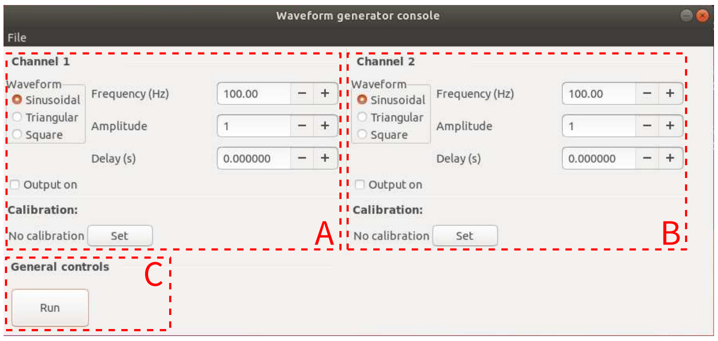
```
wavex-generator
```

To close the waveform generator just close the corresponding window. Alternatively, you can send a SIGINT signal from the parent terminal (Ctrl+C). If everything is set up properly, you should be able to run the waveform generator from anywhere within your system. If not, please check that executable files are properly installed (see Appendix C).

Upon startup, wavex generates a window (referred to as *control panel*) that provides the generator controls.

Control panel

An instance of the control panel [1] is shown in Fig. 4.1. The panel is made of three sectors, labelled by A, B, C in the figure.


Waveform generator console

File

Channel 1
Waveform: Sinusoidal [✓], Triangular, Square
Frequency (Hz) 100.00
Amplitude 1
Delay (s) 0.000000
Output on
Calibration: No calibration

Channel 2
Waveform: Sinusoidal [✓], Triangular, Square
Frequency (Hz) 100.00
Amplitude 1
Delay (s) 0.000000
Output on
Calibration: No calibration

General controls

Figure 4.1: Control panel of **wavex**.

Waveform parameters (channel 1) (A). Controls to set the waveform parameters (shape, frequency, amplitude and delay) of channel 1. If “Output on” is not set, a stream of zeroes is delivered to the sound card. If no calibration is provided, waveform amplitude is expressed as a positive integer number within the range of a 16-bit signed integer. The “Set” button allows to insert a calibration for the channel, namely an integer number corresponding to 1 Volt. If such calibration is available, waveform amplitude is expressed in Volts.

[1] The graphical aspect of this window might vary according to your system version, your GTK version, etc.

Waveform parameters (channel 2) (B). Controls to set the waveform parameters (shape, frequency, amplitude and delay) of channel 2. Controls are identical to those of channel 1.

General controls (C). This control allows to start/stop the generator. When it is stopped, the generator does not deliver *any* data to the sound card.

A. Calibrating the sound card

The sound card analog-to-digital and digital-to-analog converters connected to the microphone and headphones sockets do not provide an absolute reference in Volts, neither in input nor in output. Indeed, the conversion factor (Volts/bit) depends not only on the input (or output) **volume levels**—which are set through the operating system—but also on the specific sound card model. Consequently, the sound card has to be calibrated both in input and in output in order to provide meaningful voltage values. Please note that, despite the efforts to precisely calibrate the card, measurements will suffer from unavoidable sources of errors, most notably thermal drifts of the card components.

Any calibration will eventually rely on a **reliable voltage reference** of some kind. In the following, it is assumed that a +2.5 V reference is available and fed to a chopper circuit (see Appendix B), which is thereupon connected to the sound card input. It is crucial that the "chain" connecting the chopper circuit to the sound card is the same as the one that is used for actual measurements, including possible buffers, protection resistors, and so on.

Finally, please note that the following procedure is not mandatory to use the package.

Identifying source and sink numbers

On Linux machines, audio is managed by **PulseAudio**. Audio inputs (microphones) are referred to as *sources*, while audio outputs (headphones and speakers) are referred to as *sinks*. PulseAudio identifies and labels all installed/connected devices with an integer number starting from 0. Before calibrating, it is necessary to identify the *source number* that corresponds to the connected input audio jack and the *sink number* that corresponds to the connected output audio jack.

Concerning the *source number*, run the following command:

```
pacmd list-sources
```

The output is quite long, structured like this:

```
2 source(s) available.
        index: 0
                ...
        index: 1
                ...
```

The *source number* is one among the `index` numbers listed by this command. Look for a property called `ports`: with all probability, only one of the source(s) has this property, which looks like this:

```
                ports:
                        analog-input-internal-mic: Internal Microphone ...
                                properties:
                                        ...
                        analog-input-headphone-mic: Microphone ...
                                properties:
                                        ...
                active port: <analog-input-headphone-mic>
```

If you have a jack connected to the microphone socket, the `active port` should be the one corresponding to the socket itself. The source that has this active port should be the one corresponding to the jack input. If the output list of `pacmd list-sources` does not contain any microphone port, either a driver is missing or the port does not physically exist.

Once the *source number* is established, a very similar procedure is followed to find out the *sink number*. The command is, as one might expect,

```
pacmd list-sinks
```

The output is again quite long. As before, each sink is `index`ed by an integer number, and the `ports` property is the relevant one. For example:

```
                ports:
                        analog-output-speaker: Speakers ...
                                properties:
                                        ...
                        analog-output-headphones: Headphones ...
                                properties:
                                        ...
                active port: <analog-output-headphones>
```

If you have a jack connected to the headphones socket, the `active port` should be the one corresponding to the socket itself. The source that has this active port should be the one corresponding to the jack input. If the output list of `pacmd list-sources` does not contain any headphones port, either a driver is missing or the port does not physically exist.

At this point, *source* and *sink numbers* should be identified.

Calibrating the acquisition system (xoscilloscope)

Upon connecting the voltage reference to both sound card channels, start up **xoscilloscope** and switch to *Voltmeter mode*. The two channels display an integer number, corresponding to the averaged values of the 16-bit signed integer numbers provided by the sound card analog-to-digital converter. Signed 16-bit integers are limited to the range ±32767. Moreover, the sound card input socket can accept a maximum voltage which swings in the range ± 2 V ÷ ± 3 V before the signal is clipped or heavily distorted (this limit depends entirely on the specific card model). The 2.5 V voltage reference, once processed by the chopper circuit and fed to the sound card, which is coupled in AC, swings in the range ± 1.25 V, and should therefore be within the acceptable range of any sound card.

The calibration task consists in first regulating the sound card microphone volume to a level that gives, for the 2.5 V reference, an integer number which is high enough to provide a good resolution throughout the whole range, but not too large, to avoid clipping. A reasonable choice is that of aiming, while the reference is connected and **xoscilloscope** is in *Voltmeter mode*, to a recorded value somewhere around 20000 (60% of the aforementioned limit, to accomodate swings up to about ±4 V). The microphone volume level is adjusted from a terminal command line as follows:

```
        pacmd set-source-volume <n> <vol>
```

Here `<n>` is the *source number* (see above), and `<vol>` is the volume expressed as an integer number between `0` (mute) and `65535` (maximum volume). As a rule of thumb, the volume should be set to a “small” number, e.g. 5000 (~7%) so that the input signal is not amplified too much and even “large” signals swinging up to, say, ± 3 V, are still sampled by the sound card analog-to-digital converter without clipping. On a test laptop, for example, the microphone volume level was set by means of

```
        pacmd set-source-volume 1 5000
```

Needless to say, during any measurement the card should be set to the same `<vol>` as the one used for calibration.

Once the source volume is tuned so that the oscilloscope in *Voltmeter mode* yields a number around 20000, the integration time—which is controlled by means of the “Horizontal scale” controls—should be increased up to a couple of seconds, for example by setting the *Time scale* to $0.2\,\mathrm{s/div}$, corresponding to 2.8 s of integration time (see Chapter 3). When the integer numbers reported by the oscilloscope are reasonably stable, calibration factors (CF) for the two channels are computed as

$$\begin{aligned} \text{CF}_1[\text{V}^{-1}] &= \frac{\text{integer of ch.1}}{2.5\,\text{V}}, \\ \text{CF}_2[\text{V}^{-1}] &= \frac{\text{integer of ch.2}}{2.5\,\text{V}}. \end{aligned}$$

In general, the two numbers are slightly different. For example, on a test laptop the values turned out to be CF$_1 = 8364\,\text{V}^{-1}$, CF$_2 = 8383\,\text{V}^{-1}$.

Calibration factors should be inserted into the **xoscilloscope** console by means of dedicated "Set" buttons. Please note that the program only accepts calibration factors as integer numbers, yielding a 1-bit calibration error due to quantization alone. For the calibration factors given above, a 1-bit error is reflected in a voltage error of order 0.1 mV.

Calibrating the generation system (wavex)

The calibration procedure for **wavex** relies on the previously-calibrated **xoscilloscope**. The sound card output is generated by providing 16-bit signed integer numbers to the card digital-to-analog converter, again limited to the range ±32767. The correspondence between these integer numbers and the output signal in Volts depends on the specific sound card and on the volume level.

To calibrate the sound card output, first connect the output port to the sound card input port. It is crucial that the "chain" connecting the output socket to the input one is the same as the one that is used for actual measurements, including possible buffers, protection resistors, and so on: the condition used when calibrating the input port should be replicated.

Launch both **wavex** and **xoscilloscope** (in *Analog mode*). Set both output channels of **wavex** to a sinusoidal wave having frequency 440 Hz and amplitude 32000. This amplitude roughly corresponds to the maximum amplitude that can be fed to the sound card. The calibration task consists in regulating the sound card headphones volume to a level that yields, for this wave swinging between ±32000, an effective swing of about ±2.5 V. The headphones volume level is adjusted from a terminal command line as follows:

```
        pacmd set-sink-volume <n> <vol>
```

Here `<n>` is the *sink number* (see above), and `<vol>` is the volume expressed as an integer number between `0` (mute) and `65535` (maximum volume). As a rule of thumb, the volume should be set to a "large" number, e.g. 65000 (~99%) so that the output signal is fully amplified and can swing up to the card full range. On a test laptop, for example, the headphones volume level was set by means of

```
        pacmd set-sink-volume 0 64500
```

Needless to say, during any measurement the card should be set to the same `<vol>` as that used for calibration.

Once the sink volume is tuned so that the oscilloscope in *Voltmeter mode* shows two sinusoidal waves (one per channel) oscillating between ≈ -2.5 V and $\approx +2.5$ V, data should be saved from **xoscilloscope** to disk. The amplitudes A_1, A_2 of the stored waveforms should then be estimated, for example, by means of a fit. The calibration factors (CF) for the two channels are computed as

$$\begin{aligned} \text{CF}_1[\text{V}^{-1}] &= \frac{32000}{A_1\,[\text{V}]}, \\ \text{CF}_2[\text{V}^{-1}] &= \frac{32000}{A_2\,[\text{V}]}. \end{aligned}$$

In general, the two numbers are slightly different. For example, on a test laptop the values turned out to be CF$_1 = 12987\,\text{V}^{-1}$, CF$_2 = 13093\,\text{V}^{-1}$.

Calibration factors should be inserted into the **wavex** consoles by means of dedicated "Set" buttons. The same considerations as for the **xoscilloscope** program apply.

B. *Digital* and *Voltmeter modes*: analyzing DC signals

This Appendix describes how to adapt DC and digital signals in order to read them with the **xoscilloscope** tool.

Motivation

Correctly measuring digital signals by means of a sound card is tricky. Sound card inputs are AC-coupled, while digital signals are not zero-mean (for example, TTL logic levels are represented by 0 V – $+5$ V). Moreover, due to their higher harmonics falling above the bandwidth limit of the sound card ADC, square waves get heavily distorted when sampled by a sound card. For these reasons, it is recommended to electronically condition digital signals *before* reaching the sound card, and to post-process the acquired signal in order to detect the two logic levels.

A similar issue concerns the measurement of DC signals. In this case, the AC coupling of sound card inputs completely prevents this possibility. Again, a solution consists in electronically conditioning the DC signal by increasing its frequency *before* reaching the sound card, and then post-processing the acquired signal to compute the original DC voltage.

In both cases, the approach relies on a *chopper* circuit through which a high frequency signal (e.g. several kHz) modulates the low frequency/DC signal to be acquired.

Chopper circuit

A possible implementation of a *chopper* is shown in Fig. B.1. The RC555 circuit provides a square wave having frequency ~ 1.5 kHz between 0 and $+5$ V. The outputs V_A', V_B' of the circuit are square waves having the same frequency and whose maximum level V_{max} is given by the input signal magnitude. Note that this approach **does not allow to measure negative voltages**. Because of its AC coupling, the sound card "sees" the signal as a square wave oscillating between $\pm V_{\text{max}}/2$. The **xoscilloscope** software finally reconstructs the original signal, either as a DC voltage (in *Voltmeter mode*) or as logical levels (in *Digital mode*).

A summary of the approach is shown in Fig. B.2.

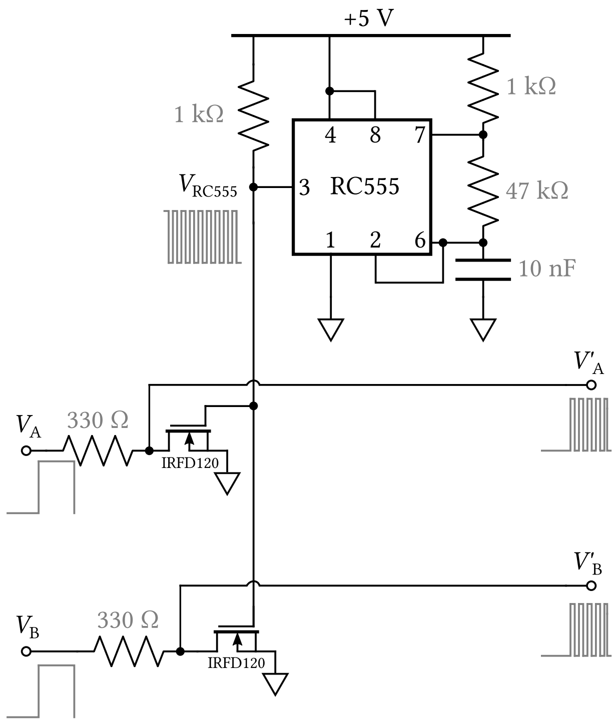


Figure B.1: *Chopper* circuit to adapt DC or digital signals, V_A and V_B, to be collected by a sound card.

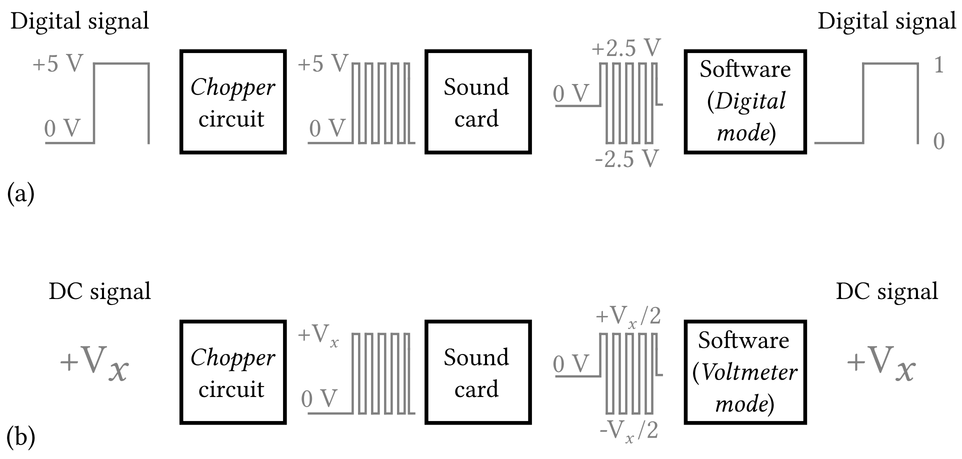


Figure B.2: Summary of the approach to acquire digital (a) and DC (b) signals.

C. Troubleshooting

- **When I type `xoscilloscope-launcher start` in a terminal, I get a “command not found” error message.**
  Check that the package is compiled, i.e. the package folder contains an `installed/` directory containing some executables. If this is not the case, you should compile and install the package from scratch, as described in Part 2, Section **Compiling & installing the toolbox**.
  If executables are present, check whether a `~/bin/` directory exists (type `ls ~/bin/`) and whether the package binaries are correctly linked into it (they should appear in the output of the `ls ~/bin/` command). If this is not the case, create the folder and launch a `make install` command within the package’s folder.
  Finally, check that the `ls ~/bin/` directory is within the shell path: type `cat ~/.profile` and check that the following lines are present:

  ```
  if [ -d "$HOME/bin" ]; then
          PATH="$HOME/bin:$PATH"
  fi
  ```

  If this is not the case, append these lines to the `~/.profile` file.

D. Technicalities

This Appendix is for those who are interested in what goes on *under the hood* of the software tools. In addition, some information is provided about the audio sample formats.

General remarks

Hard-wired constants such as sample rate, buffer sizes, channel number, and so on, are `#define`'d in C++ headers (`*.h`, `*.hpp`). If some other configuration is required by your hardware, you should modify these definitions and recompile. Similarly, the device sample size is set during initialization by calling

```
snd_pcm_hw_params_set_format(..., SND_PCM_FORMAT_S16_LE)
```

where `SND_PCM_FORMAT_S16_LE` defines 16-bit stereo (`S16`), little-endian (`LE`). Please refer to the ALSA C library reference webpage for other available hardware formats. Read/write buffers are defined within the C++ source code as

```
int16_t* buf = (int16_t*) malloc(sizeof(int16_t) * BUF_SIZE * CHN_SIZE);
```

relying on the fixed width integer type `int16_t` (16 bits). If your device has a different sample size, this declaration should be changed as well.

Structure of the *xoscilloscope* tool

Underlying the oscilloscope tool, **xoscilloscope**, there are two executable files, one concerning data acquisition and display (`xoscilloscope-engine`) and the second one providing the control GUI (`xoscilloscope-console`).

The `xoscilloscope-engine` executable is in charge of initializing the sound card as well as the interface to **gnuplot**. This interface consists of a standard pipe – created in the current directory – through which the executable (unidirectionally) delivers commands to a child `gnuplot` process. Data are passed to gnuplot by means of a dedicated fifo, which is created by `xoscilloscope-engine` as well. During normal operation, the first executable indefinitely cycles through three stages:

1. *data acquisition*, by relying on ALSA API calls, until "sufficient" data are stored;
2. *data plotting*, by passing commands to the gnuplot interface and piping the stored data;
3. *console polling*, i.e. a request to `xoscilloscope-console` for queued user commands or changes (if any) in the settings.

The `xoscilloscope-console` executable is in charge of creating the GUI containing oscilloscope controls, as well as to deliver control commands to `xoscilloscope-engine` when necessary. This executable runs two threads: the main one manages the GUI and all related events (user clicking buttons, etc.), while a child thread waits the requests from `xoscilloscope-engine`.

The two executables communicate via a socket created in the current directory by means of the `socket()` Linux system call. Communication essentially consists in `xoscilloscope-engine` periodically polling the socket; on the other hand, `xoscilloscope-console` (more precisely, its child thread) listens to the socket and answers to each poll by communicating whether commands or changes in the settings are present.

Structure of the *wavex* tool

The waveform generator tool, **wavex**, consists of a single executable file, `wavex-generator`.

The `wavex-generator` executable runs two threads. The main one is in charge of creating the GUI containing the generator controls, as well as to create a child thread that delivers data to the sound card. During normal operation, the child thread indefinitely cycles through three stages:

1. *data generation*, according to the current parameter settings;
2. *data write on device*, by relying on ALSA API calls;
3. *check parameter changes*: if any parameter has changed, the child thread exits and the main thread re-initializes it[1].

The start/stop button on the GUI window allow to manually trigger creation and deletion of the child thread. On the contrary, disabling one of the channel's output does not prevent the child thread from running: a disabled channel will simply be filled with zeroes instead of actual data.

[1]The purpose of this approach is to flush hardware immediately after user action. Because buffers on sound cards can be as large as several megabytes, if wave parameters are changed without resetting hardware the actual update of the output signal would occur several tens of seconds after user input (a 4 Mb buffer at 44.1 kHz, 16-bit stereo implies ~ 24 seconds of latency when full).