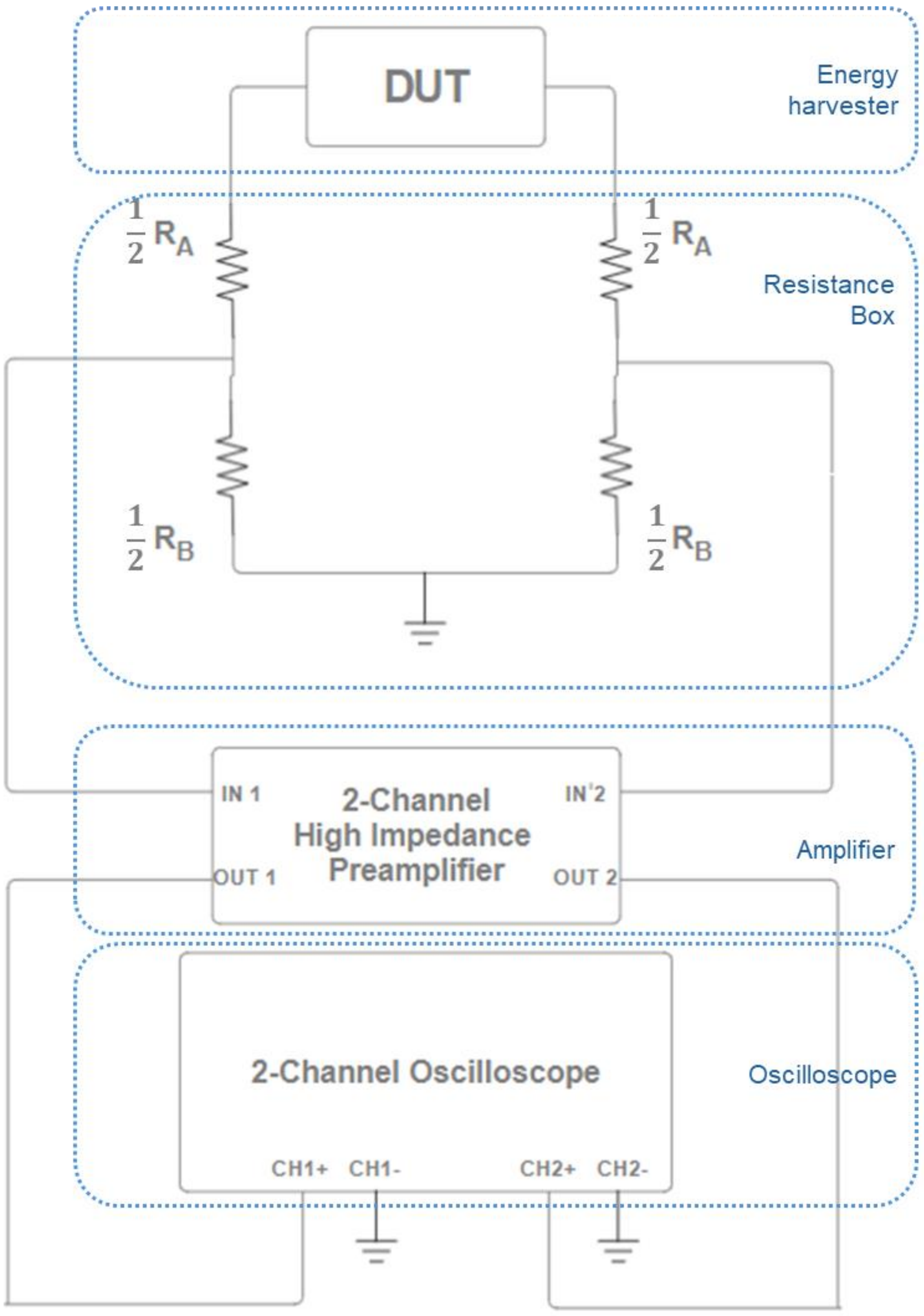
Data acquisition example: 2-channel preamplifier and oscilloscope

This data acquisition example has been tested using an oscilloscope Tektronix TBS 2000B Series, but should be applicable for other oscilloscopes based on SCPI communication.

Setup circuit schematic



The preamplifier used in this example was a High Impedance Preamplifier SR551 from Stanford Research Systems. The Device Under Test was a Triboelectric nanogenerator in which movement stimuli were applied thanks to a linear motor. Due to the preamplifier's maximum input voltages, a resistance box in voltage divider configuration was manufactured. Values of R_A and R_B could be changed using a set of different fixed resistances through rotary switches. Therefore, each channel of the oscilloscope will measure the voltage between each resistor $R_A/2$ and $R_B/2$ in the schematic. The mathematical subtraction of the signals measured in oscilloscopes channel 1 and channel 2 leads to the voltage drop on R_B ($V_{RB}=V_{CH1}-V_{CH2}$). Finally, to recalculate the voltage drop on the full impedance $R = R_A + R_B$, a proportional factor $R_A + R_B / R_B$ must be considered.

Code explanation:

The file ‘`Get2channels.py`’ queries the measurements channel 1 and channel 2 to an oscilloscope (remember to change the line 32 ‘`visa_address= 'USB0::0x0699::0x03C7::C020294::INSTR'`’ in the file to the proper address value) using SCPI commands through Python’s library pyvisa. Therefore, for this file to be executed, python3 software is required (i.e. anaconda release), as well as the installation of the drivers to connect to the oscilloscope.

If python is added to the environment path in the operating system, NanoDataLyzer application can take advantage of python execution (if the version is compatible) and directly query for data to the oscilloscope, automatically performing the measurements.

The file ‘`StartSingleMeasurement.ndl`’ contains the following code with Matlab syntax (editable by opening it on *Tools>Create Dataset Batch* and choosing the file with *Import from dataset* button):

```
res=inputdlg({'Value of RA:','Value of RB:'},'Write down the Impedances',[1 45;
 1 45],{'10M','10M'});
RA=replace(res{1},'M','e6');
RA=replace(RA,{'k','K'},'e3');
RA=replace(RA,'G','e9');
RB=replace(res{2},'M','e6');
RB=replace(RB,{'k','K'},'e3');
RB=replace(RB,'G','e9');
RA=str2double(RA);
RB=str2double(RB);
R=RA+RB;
 [ch1,ch2,t]=pyrunfile("Get2channels.py",{'scaled_wave1','scaled_wave2','scaled_time'});
data=[double(t); (double(ch1)-double(ch2))*R/RB]';
```

Let’s explain it a bit:

```
res=inputdlg({'Value of RA:','Value of RB:'},'Write down the Impedances',[1 45;
 1 45],{'10M','10M'});
```

‘`inputdlg`’ command creates a window dialog to input values from the user. In this case, two editable text boxes ask for the values of R_A and R_B. The default value for both is '10M', meaning 10 MΩ = 10^7 Ω.

```
RA=replace(res{1},'M','e6');
RA=replace(RA,{'k','K'},'e3');
RA=replace(RA,'G','e9');
RB=replace(res{2},'M','e6');
RB=replace(RB,{'k','K'},'e3');
RB=replace(RB,'G','e9');
```

Once the user has set the values, variables are stored in the structured object ‘res’. The next ‘replace’ commands aim to convert the prefix notation of the units to numerical values. For example after this code, ‘10M’ will be escaped as ‘10e6’, which means $10\cdot10^{6}$

```
RA=str2double(RA);
RB=str2double(RB);
R=RA+RB;
```

R_A and R_B are converted to numerical values with double precision and its sum becomes the variable ‘R’ that is going to be interpreted as the impedance value of the external load that will store the new dataset.

```
[ch1,ch2,t]=pyrunfile("Get2channels.py",{'scaled_wave1','scaled_wave2','scaled_time'});
```

The python file is executed to perform the measurement and variables ‘ch1’, ‘ch2’ and ‘t’ will store the data of voltages of each channel and time. These variables must be converted to numerical row arrays using the command ‘double’.

```
data=[double(t); (double(ch1)-double(ch2))*R/RB]';
```

Eventually, the matrix array composed of the column arrays time and voltage drop on R is created and stored as the variable ‘data’ that afterward the interpreter will add to the new dataset as the signal data. The matrix operand transpose ' was used to properly convert arrays from rows to columns. Voltage drop on R was calculated as $(V_{CH1}-V_{CH2})\cdot(R_A + R_B)/R_B$ as it was discussed before