qosst-bob

Release 0.10.0

Yoann Piétri Valentina Marulanda Acosta
Ilektra Karakosta-Amarantidou Matteo Schiavon

Apr 29, 2024

INTRODUCTION

`qosst-bob` is the module that contains the Digital Signal Processing (DSP), client, GUI, parameters estimation and automatised scripts of Bob.

This project is part of QOSST.

`qosst-bob` provides the following functionalities (more information *here*):

- *Digital Signal Processing for Bob*;
- *Client for Bob*;
- *GUI for Bob*;
- *Parameters estimation*;
- *Automatised scripts for Bob*

CHAPTER

ONE

GETTING STARTED

1.1 Hardware requirements

1.1.1 Operating System

The QOSST suite does not required a particular software and should work on Windows (tested), Linux (tested) and Mac (not tested).

The actual operating system requirement will come down to the hardware used for the experiment since some of them don't have interfaces with Windows.

1.1.2 Python version

QOSST if officially supporting any python version 3.9 or above.

1.2 Installing the software

There are several ways of installing the software, either by using the PyPi repositories or using the source.

1.2.1 Installing the required software for Bob

To install the required software for Bob you can simply run the command

```
$ pip install qosst-bob
```

This will automatically install `qosst-bob` (along with other required dependencies).

Alternatively, you can clone the repository at https://github.com/qosst/qosst-bob and install it by source.

1.3 Checking the version of the software

`qosst-core` will be automatically installed as it is a dependency of `qosst-bob` provides the `qosst` command from which the whole documentation can be found .

You can check the version by issuing the command

```
$ qosst info










This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.
python version: 3.11.6 (main, Feb  1 2024, 16:47:41) [GCC 11.4.0]

QOSST versions
qosst_core: 0.10.0
qosst_hal: 0.10.0
qosst_alice: Not installed
qosst_bob: 0.10.0
qosst_skr: 0.10.0
qosst_pp: Not installed
```

If the `qosst` command was not installed in the path, it also possible to run the following command:

```
$ python3 -m qosst_core.commands info
```

or

```
$ python3 -c "from qosst_core.infos import get_script_infos; print(get_script_infos())"
```

In the following we will assume that you have access to the qosst (and other) commands. If not you can replace the instructions similarly to above.

If this works and have the newest versions, you should be ready to go !

CHAPTER

TWO

FUNCTIONALITIES

On this page, we quickly review the different functionalities of `qosst-bob`, but they are fully explained in the "understanding" section.

2.1 Bob client

Alice is acting as a server and hence, Bob is acting as client. The format of the control protocol is based on the following principle: Bob send a message and Alice answers. For this reason, a class was implemented to represent Bob: *qosst_bob.bob.Bob*, which is mainly sending messages to Alice, interacting with the hardware and calling other functions as the one for the DSP or the one for parameters estimation.

Bob client is the starting point of every thing in Bob as it is the link between every function and the backbone of the GUI and the scripts.

Using Bob client is quite straightforward as shown in the following example:

```python
from qosst_bob.bob import Bob

bob = Bob("config.toml")

# Initialize the hardware
bob.open_hardware()

# Load the data of the electronic noise samples
bob.load_electronic_noise_data()

# Connect to Alice
bob.connect()

# Identification and authentication process
bob.identification()

# Start a new CV-QKD frame
bob.initialization()

# Acquire the data of the shot noise samples
bob.get_electronic_shot_noise_data()

# Proceed to the quantum information exchange
bob.dsp()

```

(continues on next page)

(continued from previous page)

```
# Make the parameters estimation
bob.parameters_estimation()

print(bob.skr)

# Close bob
bob.close()
```

More details on the client can be found *here*.

2.2 Bob DSP

Bob DSP is probably the most important code of the Bob package as it provides a way to process the data that was acquired by the hardware, and using the parameters in the configuration file, will output the recovered symbols.

The DSP is a complex bit of code that performs:

- Frame synchronisation;
- Clock recovery;
- Carrier frequency recovery;
- Frequency unshift;
- Match filtering;
- Optimal downsampling;
- Relative and global phase correction.

The DSP is explained more in details *here*.

2.3 GUI

The Graphical User Interface (GUI) of Bob is one of the ways to use Bob client. It provides an easy to use interface to perform the different steps of CV-QKD. Under the hood it mainly uses the same call as above, but also gets the information from Bob client to be able to plot them.

More information on the GUI can be found *here*.

2.4 Parameters estimatiom

The parameters estimation step corresponds to the action of taking Bob symbols, along with other information such as the electronic equivalent symbols, the electronic and shot noise equivalent symbols, Alice photon number, and a portion of Alice symbol and estimate the required parameters to estimation the key rate. In the case of Gaussian modulation, one has to estimation the transmittance T and the excess noise ξ of the channel, which is done by the code in the parameters estimation module.

The estimation of the parameters is explained in more details *here*.

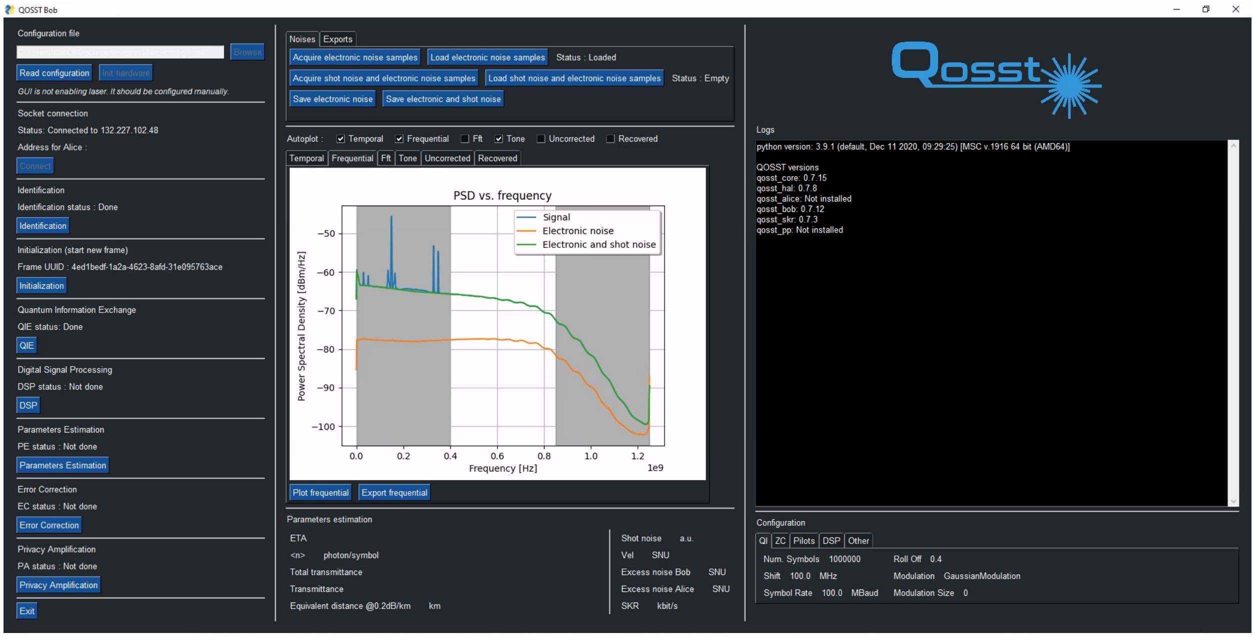

Fig. 1: Image of the GUI

2.5 Scripts

The other way to use Bob client is through already programmed scripts. There are mainly 3 scripts that are described below:

- `excess_noise` that can be called through the `qosst-bob-excess-noise` command will repeat the quantum information exchange, DSP, and parameters estimation steps a certain number of time that is given in as a parameter through the command line. The output of this script is basically the list of the excess noises;
- `transmittance` that can be called through the `qosst-bob-transmittance` command will change the applied voltage on a VOA to emulate a channel and repeat the frame exchange a certain number of times given in parameter for each attenuation. This script is therefore able to test the exchange at several distances. The output of this script is basically the excess noise and the estimated distance for each attenuation;
- `optimize` that can be called through the `qosst-bob-optimize` command will repeat the exchange of frames a certain number of times, and then will change one parameter in the configuration file of Alice and Bob. This parameter can be chosen from an established list. The output of this list is basically the excess noise and the value of the parameter of each frame. This script can be used to find the most suitable parameter for a given setting.

The documentation of the scripts can be found *here* and more information can be found *here*.

CHAPTER

THREE

USING QOSST-BOB

This section will explain how to start using `qosst-bob`. This procedure is also explained in the

3.1 Creating the configuration file and filling it

`qosst-bob` is not shipped with a default configuration file but the default configuration file of QOSST can be generated using the following command

```
$ qosst configuration create
```

This will create the configuration file at the `config.toml` default location. The documentation of this command can be found at the page of the `qosst-core` documentation.

Once the default configuration file is created, the whole `[alice]` section can be removed. The `[bob]` and `[frame]` sections must then be completed to reach the expected behaviour, and to connect to the good hardware. Here are some link that can be useful for filling these sections:

- ;
- ;
- ;
- *explanation on Bob's DSP*

3.2 Using the GUI

The GUI can then be launched with the `qosst-bob-gui` command. The good configuration file can then be loaded using the top left panel. After clicking on the "load" configuration button, the configuration parameters should now be displayed on the bottom right tabs. The hardware can be loaded by clicking on the button "Init hardware" and the communication with Alice can start, if Alice is running, by clicking on "Connect". Then clicking on "Identification" and "Initialization" will perform the initialization steps and the actual frame exchange can be done by clicking on "QIE".

To run the DSP and the parameters estimation step however, it is required to have the electronic noise and electronic and shot noise. The electronic noise must be acquired before, and the electronic and shot can be either be acquired before or have an automatic calibration. This is explained in more details *here*.

3.3 Using a script

We take here the example of the `qosst-bob-excess-noise` but the procedure is similar with the other.

The script can then be started using the simple command

```
$ qosst-bob-excess-noise -f config.toml 200
```

200 means that the script will sequentially perform 200 exchanges of frame.

It can also be useful to get the more logs by adding one or several `-v`:

```
$ qosst-bob-excess-noise -f config.toml -vv 200
```

with the following relation:

- No `-v`: Only print errors and critical errors;
- `-v`: Same as above with warnings added;
- `-vv`: Same as above with info added;
- `-vvv`: all logs.

More information on the command line can be found *here*.

Note: Alice server must be running before starting the scripts. It is also necessary to perform the calibration steps as explained *here*.

CHAPTER

FOUR

DIGITAL SIGNAL PROCESSING

The goal of this section is to give a general understanding of the Digital Signal Processing algorithm, while also giving links to where to look to for more information.

4.1 Structure of the DSP

The python structure is the following: the main code of the DSP is in the `qosst_bob.dsp.dsp` module but also call functions from modules including `qosst_bob.dsp.pilots`, `qosst_bob.dsp.resample` and `qosst_bob.dsp.zc`.

Note: The code contained in the `qosst_bob.dsp.equalizers` is not currently in use, but the code for an equalizer based on the Constant Modulus Algorithm (CMA) is available there.

The entrypoint for python is the `dsp_bob()` function that takes as input the data and the configuration object and returns the recovered symbols, the parameters for the special DSP and some debug information for the DSP.

This function actually calls the `dsp_bob_params()` which takes as parameters the data and all the DSP parameters (20+ parameters).

This last function actually calls one of four functions, depending on actual setup (clock reference shared or not, local oscillator transmitter or not).

If the clock reference is not shared, and the local oscillator is not transmitted, the `_dsp_bob_general()` function is called.

Warning: Here, we only talk about the general DSP, that should be used whenever possible. While some simplifications are possible in special cases, the other DSP algorithms should be considered unsafe.

The internal steps of the `_dsp_bob_general()` function are listed below:

- Find an approximative start of the Zadoff-Chu sequence
- Find the pilots
- Correct clock difference
- Find the pilots again with the good clock
- Estimate the beat frequency
- Find the Zadoff-Chu sequence
- Estimate the beat frequency (per subframe)

- Find one pilot (per subframe)
- Unshift the quantum signal (per subframe)
- Apply matched RRC filter (per subframe)
- Downsample (per subframe)
- Correct relative phase noise (per subframe)

We give a more involved description of the steps below.

4.2 Clock recovery

The first 3 steps above are actually done for one thing: correct the clock mismatch between Alice and Bob.

As Alice and Bob are not actively sharing a clock, their "definition of a Hertz" might vary. Even a slight variation will have big impacts at the considered rates. This is the reason why two pilots are needed in the general case: the difference of them gives us a reference for the Hertz.

The correction algorithm goes like this: given that the 2 pilots are emitted with frequencies $f_{pilot,1}$ and $f_{pilot,2}$ at Alice side, and found at $\tilde{f}^B_{pilot,1}$ and $\tilde{f}^B_{pilot,2}$, then we can estimate the clock mismatch with

$$\Delta f = \frac{\tilde{f}^B_{pilot,2} - \tilde{f}^B_{pilot,1}}{f_{pilot,2} - f_{pilot,1}}$$

This gives the "deviation of Bob's Hertz compared to Alice's Hertz".

Hence, to be able to correct the clock, we need to get the frequencies of two pilot. This can be done using the Fourier Transform or the Power Spectral Density but is more efficient if done on a small chunk of data where we know we have the pilots. This is true for two reasons: the first is the fact that on the whole data, we don't have the pilots all the time and the second is that the frequency of the beat between the two laser is moving a bit, meaning that the two pilots are also moving slightly in frequency. To be able to get a clean estimation, the time scale need to be low.

This is done by finding an estimate of the Zadoff-Chu sequence (it cannot be found precisely before clock recovery and carrier frequency estimation) with a uniform 1D filter. Once the beginning of the sequence, a small chunk of data is taken from what we know has the pilots.

4.3 Carrier frequency estimation

Once the clock is recovered, we want to estimate the beat frequency between the two lasers. Indeed, the way balanced detection works is that if the signal and the local oscillator have a frequency difference (a wavelength difference), then the acquired signal will be displaced in frequency by an amount f_{beat} corresponding to the frequency difference between the two lasers.

Even a wavelength difference of tens of pico-meters will induce shift in the order of tens of MHz, at telecom wavelength.

Once the clock has been recovered, we can find again the frequency of one of the tones, for instance the first one, $f^B_{pilot,1}$ and compare it to the emitted tone to get the beat frequency

$$f_{beat} = f^B_{pilot,1} - f_{pilot,1}$$

This beat frequency is needed to find the Zadoff-Chu sequence.

4.4 Frame synchronisation

To perform frame synchronisation, the whole data is first unshifted by the amount f_{beat}, *i.e* multiplied by a complex exponential. This has the effect of bringing the Zadoff-Chu sequence in baseband, the same was emitted.

The beginning of the Zadoff-Chu sequence, and hence the beginning of the quantum data, is found by cross-correlating the data with a locally generated Zadoff-Chu sequence, with adjusted rate.

4.5 Subframe processing

The rest of the DSP is done as subframes. This means that will recover the symbols in a small chunk of data and repeat the analysis. This is done for an already exposed reason: the beat frequency is changing overtime, and not taking this change into account gives bad result.

The size of the frame can be configured in the DSP and is given as the number of symbols that should be recovered in each frame.

4.5.1 Carrier frequency estimation

The first step is to get a proper estimation of f_{beat} from the frame. This is done the same way as before: the pilot frequency $f^B_{pilot,1}$ is estimated in the frame and the beat frequency is obtained as

$$f_{beat} = f^B_{pilot,1} - f_{pilot,1}$$

4.5.2 Pilot recovery

Then we filter the pilot with a FIR window. This pilot data will will be used later for the relative phase recovery. The cutoff frequency of the filter can be configured in the configuration.

4.5.3 Unshift signal

The whole signal is then unshifted by the amount $f_{beat} + f_{shift}$ by a multiplication by a complex exponential of the form

$$\exp(-2\pi j(f_{beat} + f_{shift})t)$$

After this operation the center frequency of the quantum data should be zero.

4.5.4 Matched filter

We then proceed to apply a matched RRC filter on the data. The same parameters are used, in particular the roll-off factor and the symbol rate.

More information on the RRC filter can be found on the .

The resulting data after this operation would correspond to the output of a raised cosine filter, which is known to minimise inter symbols interference.

4.5.5 Optimal downsampling

Now we have a mixture of all symbols. The way to get the good symbols, we need to find the good sampling point. Again you can refer to this to understand why.

If we sample at the good point, the variance of the symbols will be maximum. This is because, if we don't the best sampling point, all the symbols will be sampled with a lower amplitude.

Also the number of possible sampling point is finite because it has to be lower than the symbol period. For instance if the symbol rate is 100 MBaud, and the ADC rate is 2.5 GSamples/s, then the number of samples per symbol (SPS) at Bob side is 25, meaning that we have 25 possible starting point. The general idea is to compute

```
sps = adc_rate/symbol_rate
np.argmax([np.var(symbols[i::sps]) for i range(sps)])
```

In fact this is not that simple since, after clock correction the SPS is usually not an integer. Therefore, we have a function to downsample with a float SPS: `qosst_bob.dsp.resample._downsample_float()`.

Once the best downsampling point is found, we downsample and we get the symbols with phase error.

4.5.6 Correct relative phase

The correction of the relative phase is done in the following way. The phase error is computed as the phase difference between the filtered pilot and a perfect complex exponential at frequency $f^B_{pilot,1}$.

This phase error is then used to cancel the error on the symbols.

The actual DSP function stops here, and return a list of arrays of symbols (one array for each subframe).

Warning: As the global phase for each subframe is not the same, it is not yet possible to combine the different arrays.

4.5.7 Correct global phase

The correction of the global phase requires Alice symbols. Hence, one the main DSP function is done, Bob client will go again through the subframes and will request symbols from Alice. The ratio of symbols that will be used for global phase recovery and parameters estimation vs the symbols that will be used for key generation can be set in the configuration.

Once Bob gets the symbol, the global phase correction is found by computing the covariance between the rotated version of Bob symbols (by a global phase) and Alice symbols. The maximal covariance corresponds to the best angle for global phase correction.

After this Bob client will merge all the subframes data in one array.

4.6 Special DSP

We quickly discuss here the special DSP. The DSP transformation has to be applied on the electronic noise and the electronic and shot noise, in order to have a correct normalisation.

The special DSP only applies part of the DSP (phase is not relevant, and not need to find again the pilots and the Zadoff-Chu sequence). The operations are

- Unshift;
- RRC filter;
- Downsample.

The parameters for this DSP are outputted by the general DSP in a `qosst_bob.dsp.dsp.DSPDebug` and can directly be given to the `qosst_bob.dsp.dsp.special_dsp()` function.

CHAPTER

FIVE

CLIENT

Bob client is represented by a python class *qosst_bob.bob.Bob* with methods and attributes.

Usually the *public* methods represent the different steps of the protocol, that also corresponds to the different actions in the GUI. The attributes correspond to the variables that Bob needs to pass to the different methods such as the recovered symbols, Alice symbols, etc... but also the hardware.

Here we give a short introduction on how this client works and does. We also give the python instructions with a fully working example at the end.

In general, more details about the control protocol can be found on the .

5.1 Initializing the client and the hardware

The client is to be initialized like any other python instance. The required parameter is the location of the configuration path:

```
from qosst_bob.bob import Bob

bob = Bob("config.toml")
```

This call does almost nothing, apart from reading the configuration and setting some attributes to their default values. Once this is done, the hardware can be initialized with the following command

```
bob.open_hardware()
```

5.2 Load the calibration data

We now need to load the calibration data, in particular the electronic noise data, this can be done with the following command:

```
bob.load_electronic_noise_data()
```

The location that will be used to load the electronic shot is the one set in the configuration path in `bob.electronic_noise.path`.

Note: It is possible to acquire the electronic noise data with Bob client using the following code

```
bob.get_electronic_noise_data()
```

However this code will only make the acquisition so you need to check that the local oscillator is off, and that the input signal is switched off. Also it is possible to make the acquisition from the GUI.

It is also possible to save the electronic noise data with

```
bob.save_electronic_noise_data()
```

The location will be the one set in the configuration file in `bob.electronic_noise.path`.

You might also want, if you don't use one of the automatic calibration (see *here*) to load the electronic and shot noise data

```
bob.load_electronic_shot_noise_data()
```

5.3 Connect, identify and initialize the frame

Once the hardware is set up, and the calibration is loaded, it's time to connect to Alice. Make sure Alice server has started and execute the following code

```
bob.connect()
bob.identification()
bob.initialization()
```

The first command will establish the connection with Alice.

In the second command, Bob will send its serial number and the version of QOSST, so that Alice can verify that everything is in order. It's also during this step that the authentication is initialized and that Alice and Bob agree on a configuration.

In the last command, Bob generates a UUID and send it to Alice. This starts a new CV-QKD frame.

5.4 Quantum Information Exchange

The next step is to proceed to the Quantum Information Exchange. This step is triggered with the following code

```
bob.quantum_information_exchange()
```

Here we describe the steps that are involved when this method is called:

1. Check if slow automatic shot noise calibration has to be performed. If yes, perform the calibration by switching off the signal input, making an acquisition, and switching on the signal input. If no, do nothing;
2. Send the `QIE_REQUEST` message to Alice.;
3. Check if fast automatic shot noise calibration has to be performed. If yes, switch off the signal input. If no, do nothing.
4. When Alice answers with `QIE_READY`, start the acquisition;
5. Check if fast automatic shot noise calibration has to be performed. If yes, wait for the amount of time specified and switch back on the signal input. If no, don't wait (and don't switch bask as the switch off never happened in the first place).
6. Send the message `QIE_TRIGGER` to Alice;

7. Upon reception of the message QIE_EMISSION_STARTED from Alice, start a timer;
8. When the timer has ended, send the message QIE_ACQUISITION_ENDED to Alice;
9. Upon reception of the message QIE_ENDED from Alice, finish this action.

5.5 Digital Signal Processing

The digital signal processing step can be started with the code

```
bob.dsp()
```

This will actually do a few things. First it will do the actual DSP, as explained *here* and then, it will also ask the symbols to Alice, to be able to correct the global angle (and then later to estimate the parameters), and it will also apply the DSP on the electronic noise data and electronic and shot noise data.

5.6 Parameters estimation

Performing the parameters estimation can be done with the following code

```
bob.parameters_estimation()
```

that will execute a code as described *here*. In particular, after this step, the *skr* attribute is updated and can be displayed

```
print(bob.skr)
```

During this step, the average number of photons per symbol $\langle n \rangle$ is requested to Alice . The results of parameters estimation are also sent to Alice.

5.7 Error correction and privacy amplification

Those steps are not implemented yet and calling one of *error_correction()* or *privacy_amplification()* will result in raising the NotImplemented exception.

5.8 Closing Bob

Whenever it is possible, it is better to properly close Bob with the following code

```
bob.close()
```

This will properly close the socket and hardware connections.

5.9 Full example

```python
from qosst_bob.bob import Bob

bob = Bob("config.toml")

# Initialize the hardware
bob.open_hardware()

# Load the electronic noise data
bob.load_electronic_noise_data()

# Connect to Alice
bob.connect()

# Identification
bob.identification()

# Initialization
bob.initialization()

# QIE
bob.quantum_information_exchange()

# DSP
bob.dsp()

# Parameters estimation
bob.parameters_estimation()
print(bob.skr)

# Close Bob
bob.close()
```

GUI

Here is a picture of the GUI:

Fig. 1: Image of the GUI, with numbered sections

1. Configuration loading and hardware initialization
2. Actions
3. Noises and exports
4. Figures
5. Parameters estimation results
6. Logs
7. Configuration parameters

6.1 Configuration loading and hardware initialization

The configuration file can be chosen either by directly typing the path in the text field, or by using the Browse button, that will open an explorer. When the file has been selected, the configuration can be loaded by clicking on the "Read configuration" button.

The text field will become gray and cannot be modified without exiting. The "Init hardware" button will become available.

The "Read configuration" button does not become disabled. Indeed it is possible to read again the configuration file to update the parameters without restarting the GUI.

> **Warning:** If you reload the configuration after clicking on the `Init hardware` button, please note that any modification on the hardware will have no effect.

Clicking on the "Init hardware" button will open the hardware (ADC, and switch), and perform the basic configuration.

> **Warning:** The GUI is never enabling the laser by itself. When using the GUI, the laser should be set manually.

6.2 Actions

There are 8 possible actions that are described below.

6.2.1 Connect

This action will initialize the `QOSSTClient` socket on connect to Alice's socket.

> **Note:** Make sure that Alice is connected before this step.

6.2.2 Identification

This button will perform the identification procedure, which in practice, initialize the authentication.

6.2.3 Initialization

This button will perform the initialization for a new frame. In particular, it will generate a UUID for the frame and send to Alice.

> **Note:** While in theory, this step should be performed each time before starting a new frame. In practice however, the software won't complain starting a new quantum information exchange without having re-initialized the frame.

6.2.4 QIE

This button will start the Quantum Information Exchange (QIE) procedure. This step will involve sending a message to Alice to get ready, and when she is ready, start the acquisition and trigger Alice.

At the end of this step, the figures are plotted if the autoplot is enabled.

6.2.5 DSP

This button will apply the DSP on the quantum data, on the electronic noise data and on the electronic and shot noise data. It will also ask Alice to send a portion of its symbols.

At the end of this step, the recovered symbols, the electronic noise symbols and the electronic and shot noise symbols are available.

At the end of this step, the figures are plotted if the autoplot is enabled.

6.2.6 Parameters estimation

This button will perform the parameters estimation step after getting the number of photon at Alice's output by a request to Alice.

At the end of this step, the section 5 of the GUI, with the parameters estimation is updated with the values.

6.2.7 Error correction

This step is not yet implemented.

6.2.8 Privacy amplification

This step is not yet implemented.

6.3 Noises and exports

6.3.1 Noises

This tab has button to acquire, load and save the electronic noise and the electronic and shot noise. You can refer to the *documentation on Bob calibration* for more information.

6.3.2 Exports

This section has button to export the electronic noise, the electronic and shot noise and the signal.

Note: One might ask what is the different between the export and save features for the noises. When saving, the save is operated through QOSST data containers (see *here* and for more information) and at a location set in the configuration file, meaning that saving again will in general, result in overwriting the date. When exporting, the numpy array is saved using the numpy `save` function in a directory set in the configuration (`config.bob.export_directory`) with a unique timestamp. The save method is more useful for use inside the QOSST ecosystem, and the export for outside QOSST.

6.4 Figures

The figure area allows for direct feedback from the experiment. 6 figures are available:

- Temporal: temporal representation of the acquired data. It also displays the results of frame synchronisation;
- Frequential: Power Spectral Density of the acquire data, the electronic noise and the electronic and shot noise;
- FFT: Fourier Transform of the acquired data;
- Tone: 2D heatmap of the recovered tone (one of them);
- Uncorrected: 2D heatmap of the data before relative and global phase recovery;
- Recovered: 2D heatmap of the data after the end of the DSP.

The first 3 are available after the QIE step and the last 3 are available after the DSP step.

After the QIE step and the DSP step, the GUI performs the autoplot for the selected figures. By default autoplot is enabled for temporal, frequential and tone.

It is also possible to manually plot one figure by going to the tab and click on "Plot . . . ". Finally it is also possible to export the figure with "Export . . . "

It is also possible to choose a plotting style from the one listed below:

```
* default
* paper
* old-style
```

6.5 Parameters estimation results

After the parameters estimation step, the results are printed in this section, including

- Efficiency of the detector (actually not a result);
- The number of photons estimated by Alice;
- The total transmittance, *i.e.* ηT;
- The transmittance, *i.e.* T;
- The equivalent distance in km of the attenuation with an attenuation coefficient of 0.2 dB/km;
- The shot noise variance (in arbitrary units);
- The electronic noise variance V_{el} in Shot Noise Units;
- The excess noise at Bob side ξ_B in Shot Noise Units;
- The excess noise at Alice side ξ in Shot Noise Units;
- The secret key rate in bits/s.

6.6 Logs

The logs section will display the same logs as in the console. The verbosity of the logs will be the one passed through the command line with the following relation:

- No `-v`: Only print errors and critical errors;
- `-v`: Same as above with warnings added;
- `-vv`: Same as above with info added;
- `-vvv`: all logs.

> **Warning:** Due to the limitation of blocking calls, this window is only updated at the end of an action. However the logs in the console are displaced in real time, so it's always better to monitor the logs directly from the console.

6.7 Configuration parameters

When the configuration is loaded (or reloaded), this section of tabs is updated to display the following information:

- QI (Quantum frame)
  - Number of symbols;
  - Frequency shift of the quantum data;
  - Symbol rate;
  - Roll-Off;
  - Modulation;
  - Modulation size;
- ZC (Zadoff-Chu)
  - Root of the Zadoff-Chu sequence;
  - Length of the Zadoff-Chu sequence;
  - Rate;
- Pilots
  - Frequencies;
- DSP
  - Tone cutoff;
  - Subframes size;
  - Abort clock recovery;
  - Alice DAC rate;
  - Exclusion zone;
  - Phase filtering.

CHAPTER

SEVEN

PARAMETERS ESTIMATION

The role of parameters estimation is to estimate the different values that will be used for the computation of the secret key rate. The parameters you need depends on the exact security proof you are using, and the idea of the software was that the output of the parameters estimation procedure could be plugged into the secret key rate calculator. However, this is not currently the case. This issue is also discussed in .

This is why in the following we restrict in the case of the Gaussian modulation where the channel can be assumed Gaussian since the optimal attack is Gaussian. In this setting we need to evaluate the effect on the two first moments of our coherent states: the transmittance T and the excess noise ξ.

Taking a simple model where the symbols of Alice are represented by X and the symbols of Bob by Y, then we have the relation

$$Y = \sqrt{\eta T} X + n$$

where n is some white gaussian noise $n \sim \mathcal{N}(0, 1 + V_{el} + \eta T \xi)$. Then it's possible to show that

$$\langle X^2 \rangle = V_A$$

$$\langle XY \rangle = \sqrt{\eta T} V_A$$

$$\langle Y^2 \rangle = 1 + V_{el} + \eta T V_A + \eta T \xi$$

so that

$$T = \frac{1}{\eta} \left(\frac{\langle XY \rangle}{V_A} \right)^2$$

and

$$\xi = \frac{\langle Y^2 \rangle - 1 - V_{el} - \eta T V_A}{\eta T}$$

The actual exact formula depends on the exact schema of detection and the representation of X and Y. Also, one does not usually have direct access to X, but rather X' such that $X = \alpha X'$ with α a real number, and this α might not be known. This coefficient corresponds to the transformation from the arrays on the computer to the actual string of symbols (DAC, modulator, attenuation). This coefficient could be characterized, but this is not mandatory. Instead one can deduce α by computing $\langle X'^2 \rangle$ and comparing it to the average number of photons per symbol $\langle n \rangle$.

The parameters estimation method will also output the normalized value of V_{el}. The action of the parameters estimation method can summarized as:

1. Get the shot noise value by subtracting the electronic noise value to the electronic and shot noise value;
2. Compute the normalized value of the electronic noise V_{el};
3. Measure the coefficient α by comparing the variance of Alice symbols to the photon number;

4. Measure the covariance between the symbols of Alice and of Bob, get T;
5. Measure the variance of the symbols of Bob, get ξ.

Here is an example of use for the parameters estimation:

```
from qosst_bob.parameters_estimation.base import DefaultEstimator

symbols_alice = ...
symbols_bob = ...
photon_number = ...
electronic_symbols = ...
electronic_shot_symbols = ...

t, xi, vel = DefaultEstimator.estimate(symbols_alice, symbols_bob, photon_number,␣
↪electronic_symbols, electronic_shot_symbols)
```

CHAPTER

EIGHT

CALIBRATION OF BOB

As we saw before, two parameters are needed for the parameters estimation step: the efficiency of the detector η and the electronic noise of the detector V_{el}. However, this last quantity is normalised to the shot noise, so we actually need to calibrate three quantities.

Also note that the electronic noise samples and the electronic and shot noise samples have to follow the same treatment at the quantum data, in particular the DSP, which means that those quantity needs to be estimated before the DSP step.

8.1 Eta

A photodiode is a device that convert photons into electrons. The perfect behaviour would be that for each incoming photon, one electron is emitted. In practice, this is not the case, and we have a finite efficiency η such that

$$n_e = \eta \cdot n_{ph}$$

where n_e is the number of emitted electrons, n_{ph} the number of incoming photons and $0 \leq \eta \leq 1$. Using the fact that the current through the photodiode is given by

$$I = n_e \cdot e$$

where e is the elementary charge and the input optical power is given by

$$P = n_{ph} \cdot \frac{hc}{\lambda}$$

where h is Planck's constant, c the speed of light and λ the wavelength, we get the relation

$$\eta = \frac{hc}{\lambda e}\frac{I}{P} = \frac{hc}{\lambda e}\mathcal{R}$$

where $\mathcal{R} = \frac{I}{P}$ is called the responsivity of the photodiode and has SI units of A/W.

At $\lambda = 1550$nm, we have the following relation

$$\eta \simeq \frac{\mathcal{R}}{1.25A/W}$$

This means, that the efficiency of the photodiode can be deduced from the ratio of the current to the optical power.

However, we are not working with a single photodiode but with an interferometric detector. The idea is then to say that the responsitivty is the ratio between the sum of all photocurrents, to the optical power of the signal before the interferometer.

For instance, we the example of an homodyne detector, with 1 beam splitter and 1 balanced detector with photocurrents I_+ and I_-, if the optical power of the signal at the entrance of Bob is P_s, then the responsivity is

$$\mathcal{R} = \frac{I_+ + I_-}{P_s}$$

This means that to get the efficiency of the whole detector, we need to measure two quantities: the sum of all the photocurrents and the input optical power on the signal side.

This can be done using the following setup:

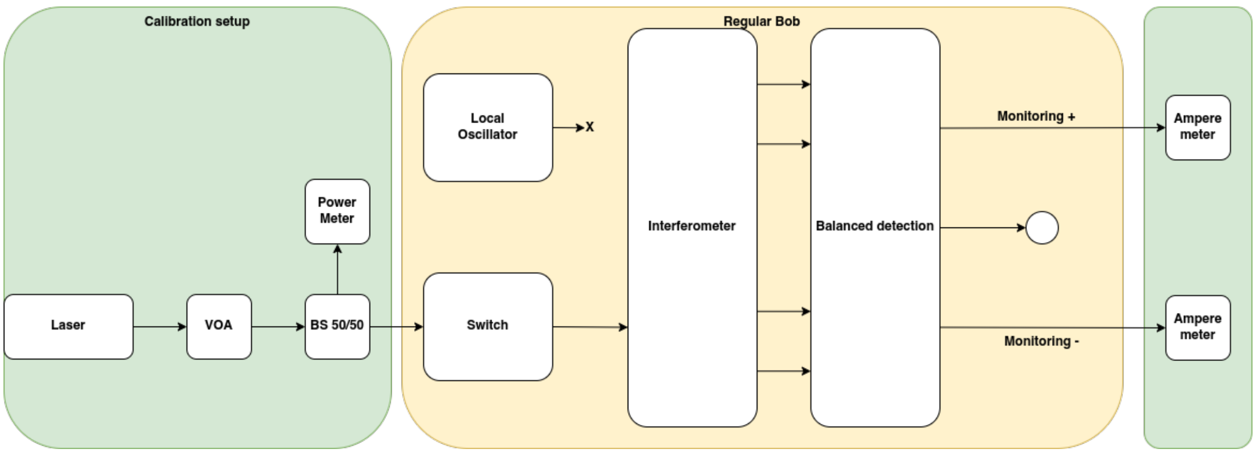

Fig. 1: Proposition of Calibration setup

Note that in the proposed scheme, the local oscillator is not plugged.

The input optical power is measured using a beam splitter and an optical power meter. The VOA allows to change the input optical power to gain in precision. The photocurrents are acquired using amperemeters.

A script in `qosst-bob` is provided to do this characterisation (assuming the output of the monitoring outputs are actually voltage outputs). The script can be called using the `qosst-bob-tools eta-voltage` command. The configuration will be done interactively in the script. More information on this script can be found *here*. The script `qosst-bob-tools eta-current` can also be used if the photocurrents are directly measured from amperemeters.

8.2 Electronic noise

Electronic noise samples must be acquired in order to estimate the electronic noise value. This must be done beforehand, with the signal input switched off and the local oscillator off.

When everything is off, it's possible to use the GUI to acquire those samples. Once the configuration has been loaded and the hardware initialized, the "acquire electronic noise samples" button can be clicked to acquire the samples. Once acquired, the samples can be saved using the "save electronic noise" button. This will save the electronic noise container (*`qosst_bob.data.ElectronicNoise`*) to the location set in the configuration file, in the parameter `bob.electronic_noise.path` (default is `electronic_noise.qosst`).

The samples can then be loaded when using the GUI, or scripts. The samples will be loaded also from the location set in `bob.electronic_noise.path`.

The electronic noise will usually not changed too much, unless the detector is changed. The temperature will have an influence in the electronic noise but the effect should not be too big in a temperature-controlled room.

8.3 Electronic and shot noise

The shot noise also needs to be calibrated as it is needed for normalisation. In practice we acquire electronic and shot noise samples and the variance of the electronic noise is subtracted to get the variance of the sole shot noise.

However, in contrast with the electronic noise, the shot noise can have quicker variations, especially since it is proportional to the power of the local oscillator, which might vary. For this reason, three methods of calibration are proposed.

In any case the calibration of the shot noise must be done with the local oscillator on and the signal input switched off.

8.3.1 One time calibration

The one time calibration of the electronic and shot noise is very similar to the one for the electronic noise. Once in the GUI, with the local oscillator on, and the signal input switched off, the samples can be acquire dby clicking on the "acquire electronic and shot noise samples", can be saved with the button "save electronic and shot noise" at the location set in `bob.electronic_shot_noise.path` and can then be loaded in the GUI or scripts, at the same path.

Warning: This method is strongly discouraged as it will not give good excess noise results.

8.3.2 Slow automatic calibration

The second option is to calibrate automatically the electronic and shot noise before each frame, with a different acquisition. This means that before each frame, the client will automatically switch off the input signal, make an acquisition and switch on the input signal again.

This can be set by choosing `bob.automatic_shot_noise_calibration = true` in the configuration file.

Note: This method is better than the first one but will still suffer from a few seconds between the two acquisitions, and it's possible to do better, as shown in the next section.

8.3.3 Fast automatic calibration

The final method for calibrating the shot noise is also automatic, and the shot noise is also calibrated for each frame, but this time the shot noise is calibrated in the same acquisition as the quantum data.

This is done in the following way:

1. Switch off the signal input
2. Start the acquisition
3. Wait some amount of time t_{switch}
4. Switch on the signal input
5. Send the trigger to Alice

With these method, the first part of the acquisition, until t_{switch} can be considered shot noise because the switch had not happened yet. Using this method the time between the end of calibration of shot noise and the beginning of the CV-QKD frame is roughly the classical communication time (in the order of tens of milliseconds).

This calibration method can be enabled by setting a non-zero value in `bob.switch.switching_time`, which will represent t_{switch} in seconds.

Warning: When using this method, the acquisition time has to be seen accordingly to take into account the additional time for the shot noise calibration.

CHAPTER

NINE

USING THE COMMAND LINE INTERFACE (CLI)

Here we describe the general idea of the command line interfaces that are shipped with `qosst-bob`. The full documentation of the CLI is available *here*.

The package is shipped with 5 command line interfaces:

- `qosst-bob-gui`
- `qosst-bob-excess-noise`
- `qosst-bob-transmittance`
- `qosst-bob-optimize`
- `qosst-bob-tools`

We give a brief description and usage example in the following.

9.1 Level of verbosity

Usually, the scripts are not logging a lot in the console: they are only logging errors and critical errors. To get more logs, it is possible to pass `-v` to the command line with the following relation:

- No `-v`: Only print errors and critical errors;
- `-v`: Same as above with warnings added;
- `-vv`: Same as above with info added;
- `-vvv`: all logs.

Note: The verbosity of logs saved in a file are not handled in the command line but in the configuration file, in the `logs` section.

9.2 qosst-bob-gui

This command launches the GUI, and don't take parameters apart from the verbosity level discussed above.

```
$ qosst-bob-gui -vv
```

9.3 qosst-bob-excess-noise

This script will initialize Bob client, and proceed to a number of frame exchanges that is passed as a parameter. Other important parameters is the verbosity level discussed above and the location of the config file.

```
$ qosst-bob-excess-noise -f config.toml -vv 200
```

will do 200 frames exchange. For each frame the different parameters are saved and at the end of the script the results of the experiment are saved in a `ExcessNoiseResults` container.

9.4 qosst-bob-transmittance

This script will initialize Bob client, and will apply an attenuation to a VOA to emulate a channel. For each value of attenuation, the script will exchange a number of fames that is given in parameter. Other important parameters include the verbosity and the location of the configuration file.

```
$ qosst-bob-transmittance -f config.toml -vv -n 5 0 5 0.01
```

mean that the value to apply to the VOA ranges from 0 to 5 with a step of 0.01 and for each value of attenuation, 5 frames are exchanged.

For each frame, the different parameters that were estimated are saved, and at the end of the script, the data is saved in a `TransmittanceResults` container.

9.5 qosst-bob-optimize

This script will initialize Bob client and an updater. The updater will automatically change one configuration parameter on Bob side and on Alice side (or only one of the two when the other change is not necessary). For each value of the parameter, the script will exchange a number of frames passed as a parameter.

There are 10 updaters available and the list can be found below:

```
* Frequency cutoff tone
* Conversion factor
* Xi versus va
* Baud rate
* Pilots amplitude
* Average tone size
* Roll off
* Frequency shift
* Subframe size
* Pilot difference
```

and more information *here*. The exact parameters will differ from one updater to the other, so it's important to either look at the *cli documentation* or to use the `-h` to get help directly from the command line. But some parameters are always there, such as the verbosity, the location of the configuration file and the number of repetitions per parameter value.

For instance, this is an example to optimize the value of the roll-off parameter on a range from 0 to 1 with step of 0.01 and 5 repetitions per roll-off value:

```
$ qosst-bob-optimize -f config.toml -vv -n 5 roll-off 0 1 0.01
```

For each frame, the values of the estimated parameters are saved and at the end of the script saved in a `OptimizationResults` container.

9.6 qosst-bob-tools

The last command line interface is for tools for Bob. Currently two tools are available: `eta-voltage` and `eta_current` that can be used for the calibration of eta (see *here* for more information).

9.6.1 eta voltage

The script can be started with the following command:

```
$ qosst-bob-tools eta-voltage 10
```

The important parameter is the gain (in the above example, it's 10) which is the gain between the monitoring output in V and the photocurrent in A (the gain is in V/A). Other parameters include the verbosity level and the `--no-save` options. All the other configuration will be done interactively in the script.

More information can be found *here*.

9.6.2 eta current

The script can be started with the following command:

```
$ qosst-bob-tools eta-current
```

The only parameters are the verbosity level and the `--no-save` option. All the other configuration will be done interactively in the script.

More information can be found *here*.

CHAPTER

TEN

CLI DOCUMENTATION

10.1 qosst-bob-gui

10.1.1 qosst-bob-gui - CLI interface

```
qosst-bob-gui [-h] [--version] [-v]
```

qosst-bob-gui options

- ***-h***, ***--help*** - show this help message and exit
- ***--version*** - show program's version number and exit
- ***-v***, ***--verbose*** - Level of verbosity. If none, nothing is printed to the console. -v will print warnings and errors, -vv will add info and -vvv will print all debug logs. (default: `0`)

10.2 qosst-bob-excess-noise

10.2.1 qosst-bob-excess-noise - CLI interface

```
qosst-bob-excess-noise [-h] [--version] [-v] [-f FILE] [--no-save] [--plot] num_rep
```

qosst-bob-excess-noise positional arguments

- ***num_rep*** - Number of repetitions of the experiment. (default: `None`)

qosst-bob-excess-noise options

- ***-h***, ***--help*** - show this help message and exit
- ***--version*** - show program's version number and exit
- ***-v***, ***--verbose*** - Level of verbosity. If none, nothing is printed to the console. -v will print warnings and errors, -vv will add info and -vvv will print all debug logs. (default: `0`)
- ***-f*** `FILE`, ***--file*** `FILE` - Path of the configuration file. Default : /home/docs/checkouts/readthedocs.org/user_builds/qosst-bob/checkouts/latest/docs/source/config.toml. (default: `{cwd}/config.toml`)

- ***--no-save*** - Don't save the data.
- ***--plot*** - Plot the data.

10.3 qosst-bob-optimize

10.3.1 qosst-bob-optimize - CLI interface

```
qosst-bob-optimize [-h] [--version] [-f FILE] [--no-save] [--plot] [-n NUM_REP]
                   [--voa-channel VOA_CHANNEL] [-v]
                   {xi-vs-va,roll-off,pilots-amplitude,conversion-factor,baud-rate,
↪subframe-size,frequency-cutoff-tone,frequency-shift,average-tone-size,pilot-difference-
↪tone}
                   ...
```

qosst-bob-optimize options

- ***-h***, ***--help*** - show this help message and exit
- ***--version*** - show program's version number and exit
- ***-f*** FILE, ***--file*** FILE - Path of the configuration file. Default : /home/docs/checkouts/readthedocs.org/user_builds/qosst-bob/checkouts/latest/docs/source/config.toml. (default: `{cwd}/config.toml`)
- ***--no-save*** - Don't save the data.
- ***--plot*** - Plot the data.
- ***-n*** NUM_REP, ***--num-rep*** NUM_REP - Number of repetitions of the experiment. (default: `10`)
- ***--voa-channel*** VOA_CHANNEL - Attenuation in V to apply to the channel VOA (default: `0`)
- ***-v***, ***--verbose*** - Level of verbosity. If none, nothing is printed to the console. -v will print warnings and errors, -vv will add info and -vvv will print all debug logs. (default: `0`)

qosst-bob-optimize xi-vs-va

Compute the excess noise while varying the variance.

```
qosst-bob-optimize xi-vs-va [-h] begin_variance end_variance step_variance
```

qosst-bob-optimize xi-vs-va positional arguments

- ***begin_variance*** - Value for the first variance (default: `None`)
- ***end_variance*** - Value for the last variance (excluded) (default: `None`)
- ***step_variance*** - Value for the step of the variance (default: `None`)

qosst-bob-optimize xi-vs-va options

- *-h*, *--help* - show this help message and exit

qosst-bob-optimize roll-off

Compute the excess noise while varying the roll-off.

```
qosst-bob-optimize roll-off [-h] begin_roll_off end_roll_off step_roll_off
```

qosst-bob-optimize roll-off positional arguments

- *begin_roll_off* - Value for the first roll-off (default: None)
- *end_roll_off* - Value for the last roll-off (excluded) (default: None)
- *step_roll_off* - Value for the step of the roll-off (default: None)

qosst-bob-optimize roll-off options

- *-h*, *--help* - show this help message and exit

qosst-bob-optimize pilots-amplitude

Compute the excess noise while varying the amplitude of the pilots.

```
qosst-bob-optimize pilots-amplitude [-h] begin_amplitude end_amplitude step_amplitude
```

qosst-bob-optimize pilots-amplitude positional arguments

- *begin_amplitude* - Value for the first amplitude (default: None)
- *end_amplitude* - Value for the last amplitude (excluded) (default: None)
- *step_amplitude* - Value for the step of the amplitude (default: None)

qosst-bob-optimize pilots-amplitude options

- *-h*, *--help* - show this help message and exit

qosst-bob-optimize conversion-factor

Compute the excess noise while varying the conversion factor of Alice.

```
qosst-bob-optimize conversion-factor [-h] initial_value begin_error end_error step_error
```

qosst-bob-optimize conversion-factor positional arguments

- ***initial_value*** - Initial value of the conversion factor. (default: `None`)
- ***begin_error*** - Value for the first error (in %) (default: `None`)
- ***end_error*** - Value for the last error (in %, excluded) (default: `None`)
- ***step_error*** - Value for the step of the error (in %) (default: `None`)

qosst-bob-optimize conversion-factor options

- ***-h***, ***--help*** - show this help message and exit

qosst-bob-optimize baud-rate

Compute the excess noise while varying the baud rate of Alice. WARNING: Make sure to have enough frequency space before doing so.

```
qosst-bob-optimize baud-rate [-h] baud_rates [baud_rates ...]
```

qosst-bob-optimize baud-rate positional arguments

- ***baud_rates*** - The list of baud rates to try. (default: `None`)

qosst-bob-optimize baud-rate options

- ***-h***, ***--help*** - show this help message and exit

qosst-bob-optimize subframe-size

Compute the excess noise while varing the size of the subframe of the DSP at Bob side.

```
qosst-bob-optimize subframe-size [-h] sizes [sizes ...]
```

qosst-bob-optimize subframe-size positional arguments

- ***sizes*** - The list of subframe sizes to try. (default: `None`)

qosst-bob-optimize subframe-size options

- ***-h***, ***--help*** - show this help message and exit

qosst-bob-optimize frequency-cutoff-tone

Compute the excess noise while varying the cutoff for the filtering of the tone on Bob side.

```
qosst-bob-optimize frequency-cutoff-tone [-h] begin_cutoff end_cutoff step_cutoff
```

qosst-bob-optimize frequency-cutoff-tone positional arguments

- ***begin_cutoff*** - Value for the first cutoff (default: `None`)
- ***end_cutoff*** - Value for the last cutoff (excluded) (default: `None`)
- ***step_cutoff*** - Value for the step of the cutoff (default: `None`)

qosst-bob-optimize frequency-cutoff-tone options

- ***-h***, ***--help*** - show this help message and exit

qosst-bob-optimize frequency-shift

Compute the excess noise while varying the frequency shift of Alice. WARNING: Make sure to have enough frequency space before doing so.

```
qosst-bob-optimize frequency-shift [-h] frequency_shifts [frequency_shifts ...]
```

qosst-bob-optimize frequency-shift positional arguments

- ***frequency_shifts*** - The list of frequency shifts to try. (default: `None`)

qosst-bob-optimize frequency-shift options

- ***-h***, ***--help*** - show this help message and exit

qosst-bob-optimize average-tone-size

Compute the excess noise while varying the size for the averaging of the pilot at phase correction.

```
qosst-bob-optimize average-tone-size [-h] sizes [sizes ...]
```

qosst-bob-optimize average-tone-size positional arguments

- ***sizes*** - The list of sizes to try. (default: `None`)

qosst-bob-optimize average-tone-size options

- ***-h***, ***--help*** - show this help message and exit

qosst-bob-optimize pilot-difference-tone

Compute the excess noise while varying the difference between the two pilots. The first pilot will be left unchanged.

```
qosst-bob-optimize pilot-difference-tone [-h] begin_difference end_difference step_
↪difference
```

qosst-bob-optimize pilot-difference-tone positional arguments

- ***begin_difference*** - Value for the first difference (default: `None`)
- ***end_difference*** - Value for the last difference (excluded) (default: `None`)
- ***step_difference*** - Value for the step of the difference (default: `None`)

qosst-bob-optimize pilot-difference-tone options

- ***-h***, ***--help*** - show this help message and exit

10.4 qosst-bob-transmittance

10.4.1 qosst-bob-transmittance - CLI interface

```
qosst-bob-transmittance [-h] [--version] [-v] [-f FILE] [--no-save] [--plot] [-n NUM_REP]
                        start_voa end_voa step_voa
```

qosst-bob-transmittance positional arguments

- ***start_voa*** - Start value of the VOA in V. (default: `None`)
- ***end_voa*** - End value of the VOA in V (not included). (default: `None`)
- ***step_voa*** - Step value of the VOA in V. (default: `None`)

qosst-bob-transmittance options

- ***-h***, ***--help*** - show this help message and exit
- ***--version*** - show program's version number and exit
- ***-v***, ***--verbose*** - Level of verbosity. If none, nothing is printed to the console. -v will print warnings and errors, -vv will add info and -vvv will print all debug logs. (default: `0`)
- ***-f*** `FILE`, ***--file*** `FILE` - Path of the configuration file. Default : /home/docs/checkouts/readthedocs.org/user_builds/qosst-bob/checkouts/latest/docs/source/config.toml. (default: `{cwd}/config.toml`)
- ***--no-save*** - Don't save the data.
- ***--plot*** - Plot the data.
- ***-n*** `NUM_REP`, ***--num_rep*** `NUM_REP` - Number of repetitions of the experiment. Default : 1 (default: 1)

10.5 qosst-bob-tools

10.5.1 qosst-bob-tools - CLI interface

```
qosst-bob-tools [-h] [--version] [-v] {eta-voltage,eta-current} ...
```

qosst-bob-tools options

- ***-h***, ***--help*** - show this help message and exit
- ***--version*** - show program's version number and exit
- ***-v***, ***--verbose*** - Level of verbosity. If none, nothing is printed to the console. -v will print warnings and errors, -vv will add info and -vvv will print all debug logs. (default: `0`)

qosst-bob-tools eta-voltage

Compute eta using voltage

```
qosst-bob-tools eta-voltage [-h] [--no-save] gain
```

qosst-bob-tools eta-voltage positional arguments

- ***gain*** - Gain of the TIA of the monitoring outputs. (default: `None`)

qosst-bob-tools eta-voltage options

- ***-h***, ***--help*** - show this help message and exit
- ***--no-save*** - Don't save the results.

qosst-bob-tools eta-current

Compute eta using current.

```
qosst-bob-tools eta-current [-h] [--no-save]
```

qosst-bob-tools eta-current options

- ***-h***, ***--help*** - show this help message and exit
- ***--no-save*** - Don't save the results.

CHAPTER ELEVEN

BOB

Client code for QOSST Bob.

class qosst_bob.bob.**Bob**(*config_path: str*, *enable_laser: bool = True*)

Client class that will interact with the sockets and the hardware.

Parameters

- **config_path** (*str*) – configuration path.
- **enable_laser** (*bool, optional*) – if True, the laser will be enabled by the client. If False, no interaction is made with the laser. Useful for the GUI. Defaults to True.

photon_number: float

Mean photon number at Alice's side.

polarisation_controller: GenericPolarisationController | None

For polarisation recovery, polarisation controller for Bob.

powermeter: GenericPowerMeter | None

For polarisation recovery, powermeter for Bob.

notifier: QOSSTNotifier

Notifier for Bob.

config_path: str

Location of the configuration path.

is_connected: bool

True if the client is connected to the server.

config: Configuration | None

Configuration object.

electronic_noise: *ElectronicNoise* **| None**

Electronic noise object.

electronic_shot_noise: *ElectronicShotNoise* **| None**

Electronic shot noise object.

electronic_symbols: ndarray | None

Array of electronic noise symbols after DSP.

electronic_shot_symbols: ndarray | None

Array of electronic+shot noise symbols after DSP.

end_electronic_shot_noise: int
End of the electronic shot noise data in case of automatic calibration.

indices: ndarray | None
Indices to ask for to Alice

alice_symbols: ndarray | None
Symbols of Alice

transmittance: float
Total transmittance estimated.

excess_noise_bob: float
Excess noise estimated at Bob.

vel: float
Normalised electronic noise.

skr: float
Secret key rate.

adc_data: list[numpy.ndarray] | None
List of arrays containing current ADC

signal_data: list[numpy.ndarray] | None
List of arrays containing the signal data.

begin_data: int | None
Indice of beginning of data

end_data: int | None
Indice of end of data

received_tone: ndarray | None
Received stone

quantum_data_phase_noisy: ndarray | None
Array of symbols with phase noise

quantum_symbols: ndarray | None
Array of corrected symbols.

enable_laser: bool
Enable the laser if True. Meant to be False when using the GUI.

laser: GenericLaser | None
Laser device for Bob.

switch: GenericSwitch | None
Switch device for Bob.

adc: GenericADC | None
ADC device for Bob.

frame_uuid: UUID | None
UUID of the current frame.

`load_configuration()` → None

Load or reload the configuration.

`open_hardware()` → None

Open the ADC, the switch and the laser (if enable laser is True).

`close_hardware()` → None

Close the ADC, the switch and the laser (if it was open).

`_init_socket()`

Init the QOSST socket.

`_init_data_adc()`

Configure the acquisition of the ADC.

`_get_adc_data()`

Get the data from the ADC.

`_init_notifier()`

Initialize the notifier.

`connect()` → bool

Connect the client to the server.

Returns
True if the operation succeded. False otherwise.

Return type
bool

`close()` → None

Close the socket and Bob.

`identification()` → bool

Identify the client to the server and get the server identification.

Returns
True if the identification was successful. False otherwise.

Return type
bool

`initialization()` → bool

Start a new frame and initializalize to the server.

Returns
True if the initialization was successful. False otherwise.

Return type
bool

`quantum_information_exchange()` → bool

Start the Quantum Information Exchange (QIE).

Returns
True if the QIE was successful. False otherwise.

Return type
bool

`dsp()` → bool

Apply the Digital Signal Processing (DSP) algorithm.

Returns
True if the DSP was successful, False otherwise.

Return type
bool

`parameters_estimation()` → bool

Run the parameters estimation algorithm.

Returns
True if the parameters estimation wen well, False otherwise.

Return type
bool

`error_correction()` → bool

Apply error correction on the data.

Raises
`NotImplementedError` – This function is not yet implemented.

Returns
True if the operation was successful, False otherwise.

Return type
bool

`privacy_amplification()` → bool

Apply privacy amplification on the data.

Raises
`NotImplementedError` – This function is not yet implemented.

Returns
True if the operation was successful, False otherwise.

Return type
bool

`_start_acquisition()`

Start the acquisition.

`_stop_acquisition()`

Stop the acquisition.

`_do_dsp()`

Actually apply the DSP to the data.

Also request the symbols to Alice, in order to correct the global phase.

`_wait_for_timer()`

Timer for the acquisition.

`get_electronic_noise_data()`

Acquire electronic+shot noise data using the configured adc.

It will make an acquisition, and compute the noise density.

load_electronic_noise_data()

Load electronic noise from a numpy file.

save_electronic_noise_data(*detector: str = ''*, *comment: str = ''*) → None

Save the electronic noise data as numpy file.

get_electronic_shot_noise_data() → None

Acquire electronic+shot noise data using the configured adc.

It will switch the state of the optical switch to the calibration state, make an acquisition, switch back, and finally compute the noise density.

load_electronic_shot_noise_data() → None

Load electronic+shot noise from a numpy file.

save_electronic_shot_noise_data(*detector: str = ''*, *power: float | None = None*, *comment: str = ''*) → None

Save the electronic+shot noise data as numpy file.

get_alice_photon_number() → float

Request variance to Alice.

Raises
`Exception` – if the received message is not the expected message.

Returns
the variance of Alice's symbols.

Return type
float

request_parameter_change(*parameter: str*, *new_value: Any*) → None

Request a parameter change to the server.

Parameters

- **parameter** (*str*) – full module name of the parameter.
- **new_value** (*Any*) – the requested value for the parameter.

_optimal_polarisation_finding()

The goal of this function is to minimize the power on the powermeter, that corresponds to the vertical polarisation.

This is done by minimizing for the three paddles.

CHAPTER

TWELVE

DATA

Module for QOSST data specific to Bob.

class qosst_bob.data.**ElectronicNoise**(*data: List[ndarray]*, *detector: str | None = None*, *comment: str | None = None*)

QOSST data class to hold electronic noise data.

Parameters

- **data** (*List[np.ndarray]*) – list of data, each element is a ndarray corresponding to a channel.
- **detector** (*Optional[str], optional*) – name of the detector. Defaults to None.
- **comment** (*Optional[str], optional*) – comment on the acquisition. Defaults to None.

data: List[ndarray]

The actual data that was acquired.

detector: str | None

Optional detector that was used for this electronic noise.

comment: str | None

Optional comment.

date: datetime

Datetime of the electronic noise acquisition.

class qosst_bob.data.**ElectronicShotNoise**(*data: List[ndarray]*, *detector: str | None = None*, *power: float | None = None*, *comment: str | None = None*)

QOSST data class to hold electronic and shot noise data.

Parameters

- **data** (*List[np.ndarray]*) – list of data, each element is a ndarray corresponding to a channel.
- **detector** (*Optional[str], optional*) – name of the detector. Defaults to None.
- **comment** (*Optional[str], optional*) – comment on the acquisition. Defaults to None.

data: List[ndarray]

The actual data that was acquired.

detector: str | None

Optional detector that was used for this electronic noise.

power: float | None
Optional power.

comment: str | None
Optional comment.

date: datetime
Datetime of the electronic nois acquisition.

class qosst_bob.data.**ExcessNoiseResults**(*configuration: Configuration*, *num_rep: int*, *excess_noise_bob: ndarray*, *transmittance: ndarray*, *photon_number: ndarray*, *datetimes: ndarray*, *electronic_noise: ndarray*, *shot_noise: ndarray*, *source_script: str*, *command_line: str*)

Data class for the results of a qosst-bob-excess-noise measurement.

Parameters

- **configuration** (Configuration) – configuration that was used for the experiment.
- **num_rep** (*int*) – number of repetitions in the experiment.
- **excess_noise_bob** (*np.ndarray*) – array of estimated excess noise bob results (in SNU).
- **transmittance** (*np.ndarray*) – array of estimated transmittance.
- **photon_number** (*np.ndarray*) – array of photon numbers.
- **datetimes** (*np.ndarray*) – list of datetime for each point.
- **electronic_noise** (*np.ndarray*) – array of estimated electronic noise (in SNU).
- **shot_noise** (*np.ndarray*) – array of estimated shot noise (in SNU).
- **source_script** (*str*) – source script that was used for the experiment.
- **command_line** (*str*) – command line that was used for the experiment.

configuration: Configuration
Configuration that was used for the experiment.

num_rep: int
Number of repetitions for this experiment.

excess_noise_bob: ndarray
Array of excess noise results.

transmittance: ndarray
Array of transmittance results.

photon_number: ndarray
Array of photon numbers.

datetimes: ndarray
List of datetimes for each point of the experiment.

electronic_noise: ndarray
Array of electronic noise (in SNU).

shot_noise: ndarray
Array of shot noise (in SNU).

source_script: str

Script that was used for this experiment.

command_line: str

Command line that was used for this experiment.

date: datetime

Datetime of the experiment.

class qosst_bob.data.**TransmittanceResults**(*configuration: Configuration*, *num_rep: int*, *excess_noise_bob: ndarray*, *transmittance: ndarray*, *photon_number: ndarray*, *datetimes: ndarray*, *electronic_noise: ndarray*, *shot_noise: ndarray*, *source_script: str*, *command_line: str*, *attenuation_values: ndarray*)

Data class for the results of a qosst-bob-transmittance measurement.

Parameters

- **configuration** (Configuration) – configuration that was used for the experiment.
- **num_rep** (*int*) – number of repetitions in the experiment.
- **excess_noise_bob** (*np.ndarray*) – array of estimated excess noise bob results (in SNU).
- **transmittance** (*np.ndarray*) – array of estimated transmittance.
- **photon_number** (*np.ndarray*) – array of photon numbers.
- **datetimes** (*np.ndarray*) – list of datetime for each point.
- **electronic_noise** (*np.ndarray*) – array of estimated electronic noise (in SNU).
- **shot_noise** (*np.ndarray*) – array of estimated shot noise (in SNU).
- **source_script** (*str*) – source script that was used for the experiment.
- **command_line** (*str*) – command line that was used for the experiment.
- **attenuation_values** (*np.ndarray*) – array of attenuation values for the tranmisttance experiment.

attenuation_values: ndarray

Array of attenuations for this particular experiment.

class qosst_bob.data.**OptimizationResults**(*configuration: Configuration*, *num_rep: int*, *excess_noise_bob: ndarray*, *transmittance: ndarray*, *photon_number: ndarray*, *datetimes: ndarray*, *electronic_noise: ndarray*, *shot_noise: ndarray*, *source_script: str*, *command_line: str*, *parameters: Dict*)

Data class for the results of a qosst-bob-optimize measurement.

Parameters

- **configuration** (Configuration) – configuration that was used for the experiment.
- **num_rep** (*int*) – number of repetitions in the experiment.
- **excess_noise_bob** (*np.ndarray*) – array of estimated excess noise bob results (in SNU).
- **transmittance** (*np.ndarray*) – array of estimated transmittance.
- **photon_number** (*np.ndarray*) – array of photon numbers.

- **datetimes** (*np.ndarray*) – list of datetime for each point.
- **electronic_noise** (*np.ndarray*) – array of estimated electronic noise (in SNU).
- **shot_noise** (*np.ndarray*) – array of estimated shot noise (in SNU).
- **source_script** (*str*) – source script that was used for the experiment.
- **command_line** (*str*) – command line that was used for the experiment.
- **parameters** (*Dict*) – dict of updated parameters.

parameters: Dict

Dict of updated parameters.

CHAPTER

THIRTEEN

DSP

Digital Signal Processing module for Bob.

13.1 DSP

Main module for the DSP algorithm.

Warning: the DSP _dsp_bob_shared_clock_shared_lo, _dsp_bob_shared_clock_unshared_lo and _dsp_bob_unshared_clock_shared_lo are adapated versions of old DSP and might not work. There are untested.

class qosst_bob.dsp.dsp.**DSPDebug**(*begin_zadoff_chu: int = 0*, *end_zadoff_chu: int = 0*, *begin_data: int = 0*, *end_data: int = 0*, *tones: ~typing.List[~numpy.ndarray] = <factory>*, *uncorrected_data: ~typing.List[~numpy.ndarray] = <factory>*, *real_pilot_frequencies: ~typing.List[float] = <factory>*, *beat_frequency: float = 0*, *delta_frequency_pilots: float = 0*, *equi_adc_rate: float = 0*)

Dataclass for debug information for the DSP.

begin_zadoff_chu: int = 0

Beginning of the Zadoff-Chu sequence.

end_zadoff_chu: int = 0

End of the Zadoff-Chu sequence.

begin_data: int = 0

Beginning of useful data.

end_data: int = 0

End of useful data.

tones: List[ndarray]

List of arrays containing filtered tone used for phase recovery.

uncorrected_data: List[ndarray]

List of arrays containing data before phase correction.

real_pilot_frequencies: List[float]

List of recovered pilot frequencies.

delta_frequency_pilots: float = 0

Difference of frequency between the two tones.

equi_adc_rate: float = 0

Equivalent ADC rate in case the clock is not shared.

class qosst_bob.dsp.dsp.**SpecialDSPParams**(*symbol_rate: float*, *adc_rate: float*, *roll_off: float*, *frequency_shift: float*, *schema: DetectionSchema*)

Dataclass for the parameters to give to the special DSP function for the elec and shot noise.

symbol_rate: float

Symbol rate in Symbols/s,.

adc_rate: float

Symbol rate in Samples/s, recovered in case clock is not shared.

roll_off: float

Roll off of the RRC filter.

frequency_shift: float

Frequency shift of the data, recovered in case clock is not shared and/or LLO setup.

schema: DetectionSchema

Detection schema to know how to interpret the data.

qosst_bob.dsp.dsp.**dsp_bob**(*data: ndarray*, *config: Configuration*) → Tuple[List[ndarray] | None, *SpecialDSPParams* | None, *DSPDebug* | None]

DSP function for Bob, given the data and the configuration.

Parameters

- **data** (*np.ndarray*) – the data on which to apply the DSP.
- **config** (Configuration) – the configuration object.

Returns

array of symbols, SpecialDSPParams containing data to apply the exact same DSP to other data and DSPDebug containing debug information,.

Return type

Tuple[Optional[List[np.ndarray]], Optional[*SpecialDSPParams*], Optional[*DSPDebug*]]

qosst_bob.dsp.dsp.**dsp_bob_params**(*data: ~numpy.ndarray*, *symbol_rate: float*, *dac_rate: float*, *adc_rate: float*, *num_symbols: int*, *roll_off: float*, *frequency_shift: float*, *num_pilots: int*, *pilots_frequencies: ~numpy.ndarray*, *zc_length: int*, *zc_root: int*, *zc_rate: float*, *shared_clock: bool = False*, *shared_lo: bool = False*, *process_subframes: bool = False*, *subframe_length: int = 0*, *fir_size: int = 500*, *tone_filtering_cutoff: float = 10000000.0*, *abort_clock_recovery: float = 0*, *excl: ~typing.List[~typing.Tuple[float, float]] | None = None*, *pilot_phase_filtering_size: int = 0*, *num_samples_fbeat_estimation: int = 100000*, *schema: ~qosst_core.schema.detection.DetectionSchema = <qosst_core.schema.detection.DetectionSchema object>*, *debug: bool = False*) → Tuple[List[ndarray] | None, *SpecialDSPParams* | None, *DSPDebug* | None]

Apply the DSP to the data given the DSP parameters.

Parameters

- **data** (*np.ndarray*) – data on which to apply the DSP.
- **symbol_rate** (*float*) – symbol rate in Symbols per second.
- **dac_rate** (*float*) – DAC rate in Hz.
- **adc_rate** (*float*) – ADC rate in Hz.
- **num_symbols** (*int*) – number of sent symbols.

- **roll_off** (*float*) – roll off value for the RRC filter
- **frequency_shift** (*float*) – frequency shift of the quantum data in Hz.
- **num_pilots** (*int*) – number of pilots.
- **pilots_frequencies** (*np.ndarray*) – list of pilot frequencies, in Hz.
- **zc_length** (*int*) – length of the Zadoff-Chu sequence.
- **zc_root** (*int*) – root of the Zadoff-Chu sequence.
- **zc_rate** (*float*) – rate of the Zadoff-Chu sequence.
- **shared_clock** (*bool, optional*) – if the clock is shared between Alice and Bob. Defaults to False.
- **shared_lo** (*bool, optional*) – if the local oscillator is shared between Alice and Bob. Defaults to False.
- **process_subframes** (*bool, optional*) – if the data should be processed at subframes. Defaults to False.
- **subframe_length** (*int, optional*) – if the previous parameter is True, the length, in samples, of the subframe. Defaults to 0.
- **fir_size** (*int, optional*) – FIR size. Defaults to 500.
- **tone_filtering_cutoff** (*float, optional*) – cutoff for the FIR filter for the pilot filtering, in Hz.
- **abort_clock_recovery** (*float, optional*) – Maximal mismatch allowed by the clock recovery algorithm before aborting. If 0, the algorithm never aborts. Defaults to 0.
- **excl** (*Optional[List[Tuple[float, float]]], optional*) – exclusion zones for the research of pilots (i.e. frequencies where we are sure the pilots are not), given as a list of tuples of float, each elements defining excluded segment (start frequency, stop frequency).
- **pilot_phase_filtering_size** (*int, optional*) – Size of the uniform1d filter to apply to the phase of the recovered pilots for correction. Defaults to 0.
- **num_samples_fbeat_estimation** (*int, optional*) – number of samples to estimate the beat frequency between the two lasers. Defaults to 100000.
- **schema** (*DetectionSchema, optional*) – detection schema to use for the DSP. Defaults to qosst_core.schema.emission.SINGLE_POLARISATION_RF_HETERODYNE.
- **debug** (*bool, optional*) – wether to return a debug dict. Defaults to False.

Returns

array of symbols, SpecialDSPParams containing data to apply the exact same DPS to other data and DSPDebug containing debug information,.

Return type

Tuple[Optional[np.ndarray], Optional[*SpecialDSPParams*], Optional[*DSPDebug*]] `qosst_bob.dsp.dsp._dsp_bob_shared_clock_shared_lo`(*data: ~numpy.ndarray*, *symbol_rate: float*, *dac_rate: float*, *adc_rate: float*, *num_symbols: int*, *roll_off: float*, *frequency_shift: float*, *num_pilots: int*, *pilots_frequencies: ~numpy.ndarray*, *zc_length: int*, *zc_root: int*, *zc_rate: float*, *process_subframes: bool = False*, *subframe_length: int = 0*, *fir_size: int = 500*, *tone_filtering_cutoff: float = 10000000.0*, *pilot_phase_filtering_size: int = 0*, *schema: ~qosst_core.schema.detection.DetectionSchema = <qosst_core.schema.detection.DetectionSchema object>*, *debug: bool = False*) → Tuple[List[ndarray] | None, *SpecialDSPParams* | None, *DSPDebug* | None]

DSP in the case of a shared clock and a shared local oscillator.

This simplifies a lot the DSP, since there is no clock difference or beat frequency.

The procedure is the following:

- Recovery of the Zadoff-Chu sequence
- Recovery of the pilot (per subframe)
- Unshift signal (per subframe)
- Apply match filter (per subframe)
- Downsample (per subframe)
- Correct relative phase noise (per subframe)

The output has still a global phase noise.

Parameters

- **data** (*np.ndarray*) – data received by Bob.
- **symbol_rate** (*float*) – symbol rate for the quantum data, in Symbols per second.
- **dac_rate** (*float*) – DAC rate, in Hz.
- **adc_rate** (*float*) – ADC rate in Hz.
- **num_symbols** (*int*) – number of symbols.
- **roll_off** (*float*) – roll-off for the RRC filter.
- **frequency_shift** (*float*) – frequnecy shift in Hz for the quantum data.
- **num_pilots** (*int*) – number of pilots.
- **pilots_frequencies** (*np.ndarray*) – list of frequencies of the pilots.
- **zc_length** (*int*) – length of the Zadoff-Chu sequence.
- **zc_root** (*int*) – root of the Zadoff-Chu sequence.
- **zc_rate** (*float*) – rate of the Zadoff-Chu sequence.
- **process_subframes** (*bool, optional*) – if True, data is processed as subframes. Defaults to False.
- **subframe_length** (*int, optional*) – number of symbols to recover in each subframe. Defaults to 0.
- **fir_size** (*int, optional*) – size for the FIR filters. Defaults to 500.

- **tone_filtering_cutoff** (*float, optional*) – cutoff, in Hz, for the filter of the tone. Defaults to 10e6.
- **pilot_phase_filtering_size** (*int, optional*) – size of the uniform1d filter to filter the phase correction. Defaults to 0.
- **schema** (*DetectionSchema, optional*) – detection schema to use for the DSP. Defaults to qosst_core.schema.emission.SINGLE_POLARISATION_RF_HETERODYNE.
- **debug** (*bool, optional*) – if True, a debug dict is returned. Defaults to False.

Returns

list of np.ndarray, each one corresponding to the recovered symbols for a subframe, SpecialDSPParams object to give to the special dsp, and DSPDebug object if debug was true.

Return type

Tuple[List[np.ndarray], *SpecialDSPParams*, Optional[*DSPDebug*]]

qosst_bob.dsp.dsp.**_dsp_bob_shared_clock_unshared_lo**(*data: ~numpy.ndarray, symbol_rate: float, dac_rate: float, adc_rate: float, num_symbols: int, roll_off: float, frequency_shift: float, num_pilots: int, pilots_frequencies: ~numpy.ndarray, zc_length: int, zc_root: int, zc_rate: float, process_subframes: bool = False, subframe_length: int = 0, fir_size: int = 500, tone_filtering_cutoff: float = 10000000.0, excl: ~typing.List[~typing.Tuple[float, float]] | None = None, pilot_phase_filtering_size: int = 0, schema: ~qosst_core.schema.detection.DetectionSchema = <qosst_core.schema.detection.DetectionSchema object>, debug: bool = False*) → Tuple[List[ndarray] | None, *SpecialDSPParams* | None, *DSPDebug* | None]

DSP in the case of a shared clock and an unshared local oscillator.

This simplifies the DSP, since there is no clock difference.

The procedure is the following:

- Estimation of f_beat (to find the Zadoff-Chu sequence)
- Recovery of the Zadoff-Chu sequence
- Recovery of the pilot (per subframe)
- Estimation of f_beat (per subframe)
- Unshift signal (per subframe)
- Apply match filter (per subframe)
- Downsample (per subframe)
- Correct relative phase noise (per subframe)

The output has still a global phase noise.

Parameters

- **data** (*np.ndarray*) – data measured by Bob.
- **symbol_rate** (*float*) – symbol rate in Symbols per second.

- **dac_rate** (*float*) – DAC rate, in Hz.
- **adc_rate** (*float*) – ADC rate, in Hz.
- **num_symbols** (*int*) – number of symbols.
- **roll_off** (*float*) – roll off factor for the RRC filter.
- **frequency_shift** (*float*) – frequency shift of the quantum symbols in Hz.
- **num_pilots** (*int*) – number pilots.
- **pilots_frequencies** (*np.ndarray*) – list of the frequencies of the pilots.
- **zc_length** (*int*) – length of the Zadoff-Chu sequence.
- **zc_root** (*int*) – root of the Zadoff-Chu sequence.
- **zc_rate** (*float*) – shift, in Hz, of the Zadoff-Chu sequence.
- **process_subframes** (*bool, optional*) – if True, process the data with subframes. Defaults to False.
- **subframe_length** (*int, optional*) – number of symbols to recover in each subframe. Defaults to 0.
- **fir_size** (*int, optional*) – size of the FIR filters.. Defaults to 500.
- **tone_filtering_cutoff** (*float, optional*) – cutoff, in Hz, for the filtering of the pilots.. Defaults to 10e6.
- **excl** (*Optional[List[Tuple[float, float]]], optional*) – exclusion zones for the research of pilots (i.e. frequencies where we are sure the pilots are not), given as a list of tuples of float, each elements defining excluded segment (start frequency, stop frequency). Defaults to None.
- **pilot_phase_filtering_size** (*int, optional*) – size of the uniform1d filter to filter the phase correction. Defaults to 0.
- **schema** (*DetectionSchema, optional*) – detection schema to use for the DSP. Defaults to qosst_core.schema.emission.SINGLE_POLARISATION_RF_HETERODYNE.
- **debug** (*bool, optional*) – if True, the DSPDebug object is returned. Defaults to False.

Returns

list of np.ndarray, each one corresponding to the recovered symbols for a subframe, SpecialDSPParams object to give to the special dsp, and DSPDebug object if debug was true.

Return type

Tuple[List[np.ndarray], *SpecialDSPParams*, Optional[*DSPDebug*]] `qosst_bob.dsp.dsp._dsp_bob_unshared_clock_shared_lo`(*data: ~numpy.ndarray*, *symbol_rate: float*, *dac_rate: float*, *adc_rate: float*, *num_symbols: int*, *roll_off: float*, *frequency_shift: float*, *num_pilots: int*, *pilots_frequencies: ~numpy.ndarray*, *zc_length: int*, *zc_root: int*, *zc_rate: float*, *process_subframes: bool = False*, *subframe_length: int = 0*, *fir_size: int = 500*, *tone_filtering_cutoff: float = 10000000.0*, *pilot_phase_filtering_size: int = 0*, *schema: ~qosst_core.schema.detection.DetectionSchema = <qosst_core.schema.detection.DetectionSchema object>*, *debug: bool = False*) → Tuple[List[ndarray] | None, *SpecialDSPParams* | None, *DSPDebug* | None]

DSP in the case of an unshared clock and a shared local oscillator.

This simplifies the DSP, since there is no frequency beat.

The procedure is the following:

- Recovery of the Zadoff-Chu sequence
- Recovery of the pilot (per subframe)
- Recovery of the clock (per subframe)
- Unshift signal (per subframe)
- Apply match filter (per subframe)
- Downsample (per subframe)
- Correct relative phase noise (per subframe)

Parameters

- **data** (*np.ndarray*) – data measured by Bob.
- **symbol_rate** (*float*) – symbol rate in Symbols per second.
- **dac_rate** (*float*) – DAC rate, in Hz.
- **adc_rate** (*float*) – ADC rate, in Hz.
- **num_symbols** (*int*) – number of symbols.
- **roll_off** (*float*) – roll-off factor for the RRC filter.
- **frequency_shift** (*float*) – frequency shift of the quantum symbols, in Hz.
- **num_pilots** (*int*) – number of pilots.
- **pilots_frequencies** (*np.ndarray*) – list of the frequencies of the pilots.
- **zc_length** (*int*) – length of the Zadoff-Chu sequence.
- **zc_root** (*int*) – root of the Zadoff-Chu sequence.
- **zc_rate** (*float*) – rate of the Zadoff-Chu sequence.
- **process_subframes** (*bool, optional*) – if True, process the data in subframes. Defaults to False.

- **subframe_length** (*int, optional*) – number of symbols to recover in each subframe. Defaults to 0.
- **fir_size** (*int, optional*) – size of the FIR filters. Defaults to 500.
- **tone_filtering_cutoff** (*float, optional*) – cutoff, in Hz, for the filtering of the tone. Defaults to 10e6.
- **pilot_phase_filtering_size** (*int, optional*) – size of the uniform1d filter to filter the phase correction. Defaults to 0.
- **schema** (*DetectionSchema, optional*) – detection schema to use for the DSP. Defaults to qosst_core.schema.emission.SINGLE_POLARISATION_RF_HETERODYNE.
- **debug** (*bool, optional*) – if True, the DSPDebug object is returned. Defaults to False.

Returns
list of np.ndarray, each one corresponding to the recovered symbols for a subframe, SpecialDSP-Params object to give to the special dsp, and DSPDebug object if debug was true.

Return type
Tuple[List[np.ndarray], *SpecialDSPParams*, Optional[*DSPDebug*]]

`qosst_bob.dsp.dsp.`**`_dsp_bob_general`**(*data: ~numpy.ndarray, symbol_rate: float, dac_rate: float, adc_rate: float, num_symbols: int, roll_off: float, frequency_shift: float, num_pilots: int, pilots_frequencies: ~numpy.ndarray, zc_length: int, zc_root: int, zc_rate: float, process_subframes: bool = False, subframe_length: int = 0, fir_size: int = 500, tone_filtering_cutoff: float = 10000000.0, abort_clock_recovery: float = 0, excl: ~typing.List[~typing.Tuple[float, float]] | None = None, pilot_phase_filtering_size: int = 0, num_samples_fbeat_estimation: int = 100000, schema: ~qosst_core.schema.detection.DetectionSchema = <qosst_core.schema.detection.DetectionSchema object>, debug: bool = False*) → Tuple[List[ndarray] | None, *SpecialDSPParams* | None, *DSPDebug* | None]

General DSP.

The steps are the following:

- Find an approximative start of the Zadoff-Chu sequence
- Find the pilots
- Correct clock difference
- Find the pilots again with the good clock
- Estimate the beat frequency
- Find the Zadoff-Chu sequence
- Estimate the beat frequency (per subframe)
- Find one pilot (per subframe)
- Unshift the quantum signal (per subframe)
- Apply matched RRC filter (per subframe)
- Downsample (per subframe)
- Correct relative phase noise (per subframe)

The output has stil a global phase difference.

Parameters

- **data** (*np.ndarray*) – data measured by Bob.
- **symbol_rate** (*float*) – symbol rate in symbols per second.
- **dac_rate** (*float*) – DAC rate, in Hz.
- **adc_rate** (*float*) – ADC rate, in Hz.
- **num_symbols** (*int*) – number of symbols.
- **roll_off** (*float*) – roll-off factor for the RRC filter.
- **frequency_shift** (*float*) – frequency shift of the quantum symbol, in Hz.
- **num_pilots** (*int*) – number of pilots.
- **pilots_frequencies** (*np.ndarray*) – list of the frequencies of the pilots.
- **zc_length** (*int*) – length of the Zadoff-Chu sequence.
- **zc_root** (*int*) – root of the Zadoff-Chu sequence.
- **zc_rate** (*float*) – rate of the Zadoff-Chu sequence.
- **process_subframes** (*bool, optional*) – if True, process the data with subframes. Defaults to False.
- **subframe_length** (*int, optional*) – number of symbols to recover in each subframes. Defaults to 0.
- **fir_size** (*int, optional*) – size of the FIR filters. Defaults to 500.
- **tone_filtering_cutoff** (*float, optional*) – cutoff, in Hz, for the pilot filtering. Defaults to 10e6.
- **abort_clock_recovery** (*float, optional*) – maximal mismatch allowed by the clock recovery algorithm before aborting. If 0, the algorithm never aborts.. Defaults to 0.
- **excl** (*Optional[List[Tuple[float, float]]], optional*) – exclusion zones for the research of pilots (i.e. frequencies where we are sure the pilots are not), given as a list of tuples of float, each elements defining excluded segment (start frequency, stop frequency). Defaults to None.
- **pilot_phase_filtering_size** (*int, optional*) – size of the uniform1d filter to filter the phase correction. Defaults to 0.
- **num_samples_fbeat_estimation** (*int, optional*) – number of samples for the estimation of fbeat. Defaults to 100000.
- **schema** (*DetectionSchema, optional*) – detection schema to use for the DSP. Defaults to qosst_core.schema.emission.SINGLE_POLARISATION_RF_HETERODYNE.
- **debug** (*bool, optional*) – if True, the DSPDebug object is returned. Defaults to False.

Returns

list of np.ndarray, each one corresponding to the recovered symbols for a subframe, SpecialDSPParams object to give to the special dsp, and DSPDebug object if debug was true.

Return type

Tuple[Optional[List[np.ndarray]], Optional[*SpecialDSPParams*], Optional[*DSPDebug*]] qosst_bob.dsp.dsp.**find_global_angle**(*received_data: ndarray*, *sent_data: ndarray*, *precision: float = 0.001*) → Tuple[float, float]

Find global angle between received and sent data by exhaustive search.

The best angle is found when the real part of the covariance is the highset between the two sets. A certain number of angles will be tested to statisfy the required precision. In fact the number of tested points will ceil(2*pi/precision) with an actual precision of 2*pi/(number of points) with a precision lower or equal to the targeted precision.

The returned value is an angle in radian, between -pi and pi.

Parameters

- **received_data** (*np.ndarray*) – the symbols received by Bob after the DSP.
- **sent_data** (*np.ndarray*) – the send symbols by Alice.
- **precision** (*float, optional*) – the precision wanted on the angle, in radians. Defaults to 0.001.

Returns
the angle that maximises the covariance, in radians, and the maximal covariance.

Return type
Tuple[float,float]

qosst_bob.dsp.dsp.**special_dsp**(*elec_noise_data: List[ndarray]*, *elec_shot_noise_data: List[ndarray]*, *params: SpecialDSPParams*) → Tuple[ndarray, ndarray]

Special DSP to apply on the electronic and electronic and shot noises.

Parameters

- **elec_noise_data** (*List[np.ndarray]*) – list of arrays (for each channel) of electronic noise data.
- **elec_shot_noise_data** (*List[np.ndarray]*) – list of arrays (for each channel) of electronic and shot noise data.
- **params** (SpecialDSPParams) – the dictionnary returned by the DSP containing the required parameters.

Returns
the electronic symbols and electronic and shot symbols.

Return type
Tuple[np.ndarray, np.ndarray]

qosst_bob.dsp.dsp.**_special_dsp_params**(*elec_noise_data: ndarray*, *elec_shot_noise_data: ndarray*, *symbol_rate: float*, *adc_rate: float*, *roll_off: float*, *frequency_shift: float*, *_schema: DetectionSchema*) → Tuple[ndarray, ndarray]

Special DSP to apply the electronic and electronic and shot noises taking the parameters.

Parameters

- **elec_noise_data** (*np.ndarray*) – array of the electronic noise.
- **elec_shot_noise_data** (*np.ndarray*) – array of the electronic and shot noise.
- **symbol_rate** (*float*) – symbol rate of the quantum symbols, in Symbols per second.
- **adc_rate** (*float*) – ADC rate, in Hz.
- **roll_off** (*float*) – roll-off factor of the RRC filter.

- **frequency_shift** (*float*) – frequency shift of the quantum data, in Hz.
- **schema** (*DetectionSchema*) – schema to know how to interpret the data.

Returns
the electronic symbols and electronic and shot symbols.

Return type
Tuple[np.ndarray, np.ndarray]

13.2 Equalizers

Module for equalization.

class qosst_bob.dsp.equalizers.**CMAEqualizer**(*length: int*, *step: float*, *p_param: int = 2*, *q_param: int = 2*, *target_radius: float = 1*, *error_threshold: float = 0.02*)

Channel equalizer based on the Constant Modulus Algorithm (CMA).

Initialize the CMA equalizer.

Parameters

- **length** (*int*) – length of the equalizer.
- **step** (*float*) – step of the equalizer.
- **p_param** (*int, optional*) – p parameter of the equalizer. Defaults to 2.
- **q_param** (*int, optional*) – q parametwer of the equalizer. Defaults to 2.
- **target_radius** (*float, optional*) – target radius of the equalizer. Defaults to 1.
- **error_threshold** (*float, optional*) – error threshold. The training will stop wen the desired error is reached. Setting 0 will make the algorithm run on all the data. Defaults to 0.02.

length: int

Length of the equalizer.

step: float

Step of the equalizer.

p_param: int

P parameter of the equalizer.

q_param: int

Q parameter of the equalizer.

target_radius: float

Target radius of the equalizer.

error_threshold: float

Error threshold (training stop when this value is reached).

weights: ndarray | None

Weights of the equalizer.

train(*train_data: ndarray*) → Tuple[ndarray, ndarray | None, ndarray | None]

Train the CMA on the train data and updates the weights.

Parameters
train_data (*np.ndarray*) – the data that should have a constant modulus.

Returns
a tuple containing the corrected tain data (only before the error threshold was reached), the errors vector and the weights vector.

Return type
Tuple[np.ndarray, Optional[np.ndarray], Optional[np.ndarray]]

apply(*data: ndarray*) → ndarray

Apply the equalizer to the data.

Parameters
data (*np.ndarray*) – the data to apply the equalizer on.

Returns
the equalized data.

Return type
np.ndarray

13.3 Pilots

Pilots processing and related functions.

qosst_bob.dsp.pilots.**recover_tones**(*data: ndarray*, *frequencies: List[float]*, *rate: float*, *fir_size: int*, *cutoff: float = 10000000.0*) → List[ndarray]

Recover all the tones within data given the list of frequencies of the tones.

This will get the tones by applying a FIR of size fir_size and cut-off cutoff on the data.

Parameters

- **data** (*np.ndarray*) – data on which the tones should be recovered.
- **frequencies** (*List[float]*) – list of the frequencies of the tones.
- **rate** (*float*) – rate of the data, in Samples per second.
- **fir_size** (*int*) – the size of the FIR.
- **cutoff** (*float, optional*) – the cut-off frequency of the filter, in Hz. Defaults to 10e6.

Returns
the list of received tones.

Return type
List[np.ndarray]

qosst_bob.dsp.pilots.**recover_tone**(*data: ndarray*, *frequency: float*, *rate: float*, *fir_size: int*, *cutoff: float = 10000000.0*) → ndarray

Recover the tone within data given the frequency of the tone.

This will get the tones by applying a FIR of size fir_size and cut-off cutoff on the data.

Parameters

- **data** (*np.ndarray*) – the data on which the tones should be recovered.

- **frequencies** (*float*) – the frequency of the tone.
- **rate** (*float*) – the rate of the data.
- **fir_size** (*int*) – the size of the FIR.
- **cutoff** (*float, optional*) – the cut-off frequency of the filter. Defaults to 10e6.

Returns
the received tone.

Return type
np.ndarray

qosst_bob.dsp.pilots.**find_one_pilot**(*data: ndarray*, *rate: float*, *excl: List[Tuple[float, float]] | None = None*) → float

Find the frequency of one pilot by looking at maximums in the FFT.

Parameters

- **data** (*np.ndarray*) – the data to analyze.
- **rate** (*float*) – the sampling rate, in Samples per second.
- **excl** (*Optional[List[Tuple[float, float]]], optional*) – List of exclusion zones. Each tuple will be considered as (beginning of exclusion zone in Hz, end of exclusion zone in Hz). Defaults to None.

Returns
frequency of the pilot, in Hz.

Return type
float

qosst_bob.dsp.pilots.**find_two_pilots**(*data: ndarray*, *rate: float*, *tone_excl: float = 5000000.0*, *excl: List[Tuple[float, float]] | None = None*) → Tuple[float, float]

Find the frequency of two pilots by looking at maximums in the FFT.

Parameters

- **data** (*np.ndarray*) – the data to analyze.
- **rate** (*float*) – the sampling rate, in Samples per second.
- **tone_excl** (*float, optional*) – exclusion zone after first pilot is found (f_pilot1 - tone_excl, f_pilot1 + tone_excl). Defaults to 5e6.
- **excl** (*List[Tuple[float, float]], optional*) – List of exclusion zones. Each tuple will be considered as (beginning of exclusion zone in Hz, end of exclusion zone in Hz). Defaults to None.

Returns
frequency of the first pilot and frequency of the second pilot, in Hz. freq1 < freq2.

Return type
Tuple[float, float]

qosst_bob.dsp.pilots.**equivalent_adc_rate_one_pilot**(*data: ndarray*, *frequency: float*, *rate: float*, *fir_size: int*, *cutoff: float = 10000000.0*) → float

Find the equivalent ADC rate with a linear fit on the angle difference of the received tone.

Parameters

- **data** (*np.ndarray*) – the received data.

- **frequencies** (*float*) – the frequency of the pilot tone.
- **rate** (*float*) – the rate of the ADC in Hz.
- **fir_size** (*int*) – the fir size of the filters for the tone recovery.
- **cutoff** (*float, optional*) – the cut-off frequency of the filter. Defaults to 10e6.

Returns
the equivalent ADC rate in Hz.

Return type
float

qosst_bob.dsp.pilots.**phase_noise_correction**(*received_tone: ndarray*, *frequency: float*, *rate: float*) → ndarray

Return the phase difference to apply to correct the relative phase noise.

The phase noise is computed as the difference between the received tone and the expected tone.

Parameters

- **received_tone** (*np.ndarray*) – the received tone.
- **frequency** (*float*) – the frequency of the received tone, in Hz.
- **rate** (*float*) – the rate of the data, in Samples per second.

Returns
the array of phase difference.

Return type
np.ndarray

qosst_bob.dsp.pilots.**correct_noise**(*data: ndarray*, *sampling_point: int*, *sps: float*, *received_tone: ndarray*, *frequency: float*, *rate: float*, *filter_size: int = 0*) → ndarray

Correct phase noise using phase reference.

Parameters

- **data** (*np.ndarray*) – data to correct.
- **sampling_point** (*int*) – the best sampling point found.
- **sps** (*float*) – the value of samples per symbol.
- **received_tone** (*np.ndarray*) – the data for the received tone.
- **frequency** (*float*) – the frequency of this received tone, in Hz.
- **rate** (*float*) – the sampling rate, in Samples per second.
- **filter_size** (*int, optional*) – size of the uniform1d filter to apply to the phase. Defaults to 0.

Returns
the corrected data.

Return type
np.ndarray

13.4 Resample

Modules to resample data (mostly downsample) and associated functions.

`qosst_bob.dsp.resample.`**`downsample`**(*data: ndarray*, *start_point: int*, *downsampling_factor: float*) → ndarray

Downsample data starting with start_point and with a downsampling factor of downsampling_factor.

If the downsampling_factor is an integer, it uses the standard slice method.

Parameters

- **data** (*np.ndarray*) – the data to downsample.
- **start_point** (*int*) – the start point: i.e. the first point in the downsample array is data[start_point].
- **downsampling_factor** (*float*) – the downsampling factor: i.e. points in the downsampled arry are in the form data[start_point + k*downsampling_factor].

Returns

downsampled data.

Return type

np.ndarray

`qosst_bob.dsp.resample.`**`_downsample_int`**(*data: ndarray*, *start_point: int*, *downsampling_factor: int*) → ndarray

Downsample data starting with start_point and with a downsampling factor of downsampling_factor, in the case where downsampling_factor is an integer.

It uses the standard slice method.

Parameters

- **data** (*np.ndarray*) – the data to downsample.
- **start_point** (*int*) – the start point: i.e. the first point in the downsample array is data[start_point].
- **downsampling_factor** (*float*) – the downsampling factor: i.e. points in the downsampled arry are in the form data[start_point + k*downsampling_factor].

Returns

downsampled data.

Return type

np.ndarray

`qosst_bob.dsp.resample.`**`_downsample_float`**(*data: ndarray*, *start_point: int*, *downsampling_factor: float*) → ndarray

Downsample by a floating factor.

Usually the operation is to take the point start_point + k * downsampling_factor.

However, here downsampling_factor is a float so we want to compensate by approximating to the nearest integer with rint.

Hence if L denotes the length of the list, the requirement on k is that

rint(start_point + k * downsampling_factor) < L start_point+k*downsampling_factor <= L-0.5 k <= (L-0.5-start_point)/downsampling_factor

As k is an integer

k <= np.floor((L-0.5-start_point)/downsampling_factor)

As k must take this last value to recover everything the arange is np.arange(np.floor((L-0.5-start_point)/downsampling_factor) + 1)

Note that the rint function was not used as its behavior is weird when being exactly between two integers (i.e. it rounds to the nearest integer value) which is hard to take into account for every possibility.

Instead, we use np.ceil(x-0.5) that has the same behavior as np.rint except that is always round down when between exactly two values.

Parameters

- **data** (*np.ndarray*) – the data to downsample.
- **start_point** (*int*) – the start point: i.e. the first point in the downsample array is data[start_point].
- **downsampling_factor** (*float*) – the downsampling factor: i.e. points in the downsampled arry are in the form data[start_point + k*downsampling_factor].

Returns
downsampled data.

Return type
np.ndarray

`qosst_bob.dsp.resample.`**best_sampling_point**(*data: ndarray*, *sps: float*) → int

Find the best sampling point with maximal variance, for the downsampling after the matched filter.

The best sampling point is found by testing all the possible sampling point and by taking the one with maximal variance.

Parameters

- **data** (*np.ndarray*) – the data from which to find the best sampling point.
- **sps** (*float*) – the samples per symbol value.

Returns
the best sampling point.

Return type
int

`qosst_bob.dsp.resample.`**_best_sampling_point_int**(*data: ndarray*, *sps: int*) → int

Find the best sampling point with maximal variance, with the assumption that the sps is an integer value.

The best sampling point is found by testing all the possible sampling point and by taking the one with maximal variance.

Parameters

- **data** (*np.ndarray*) – the data from which to find the best sampling point.
- **sps** (*int*) – the samples per symbol value.

Returns
the best sampling point.

Return type
int

qosst_bob.dsp.resample.**_best_sampling_point_float**(*data: ndarray*, *sps: float*) → int

Find the best sampling point with maximal variance, with the assumption that the sps is not an integer value.

The best sampling point is found by testing all the possible sampling point and by taking the one with maximal variance.

Parameters

- **data** (*np.ndarray*) – the data from which to find the best sampling point.
- **sps** (*float*) – the samples per symbol value.

Returns
the best sampling point.

Return type
int

13.5 ZC

DSP functions to deal with Zadoff-Chu sequences and synchronisation.

qosst_bob.dsp.zc.**synchronisation_zc**(*data: ndarray*, *zc_root: int*, *zc_length: int*, *resample: float = 1*, *use_abs=True*, *ratio_approx=50*) → Tuple[int, int]

Find the beginning of a Zadoff-Chu sequence in data.

This function finds the beginning and the end of a Zadoff-Chu sequence by computing the cross-correlation of the Zadoff-Chu sequence and the data.

From version 0.4.27, the behavior of this function is a little different:

first we find a first approximate of the Zadoff-Chu location by making a rolling average of the data, and then, we make the cross-correlation around this point.

Parameters

- **data** (*np.ndarray*) – the data from where the Zadoff-Chu should be found.
- **zc_root** (*int*) – the root of the Zadoff-Chu sequence.
- **zc_length** (*int*) – the length of the Zadoff-Chu sequence.
- **resample** (*float, optional*) – the optional resample to apply to the Zadoff-Chu sequence. Defaults to 1.
- **ratio_approx** (*int, optional*) – the length of the data will be divided by this value to get the window size of the rolling average for the approximation. Defaults to 50.

Returns
tuple including the beginning and the end of the Zadoff-Chu sequence.

Return type
Tuple[int, int]

CHAPTER

FOURTEEN

EXCESS NOISE

Script to measure excess noise and repeating exchanges of frames.

`qosst_bob.excess_noise._create_parser()` → ArgumentParser

Create the parser for qosst-bob-excess-noise.

Returns
parser for the qosst-bob-excess-noise.

Return type
argparse.ArgumentParser

`qosst_bob.excess_noise.main()`

Entry point of the excess noise script.

CHAPTER FIFTEEN

GUI

Module containing the Graphical User Interface (GUI) for Bob, including the layout, the interaction code and the plot code.

15.1 Bobgui

A graphical user interface for Bob.

class qosst_bob.gui.bobgui.**GUIHandler**(*window: Window*)

A log handler to print the log in the console on the GUI.

Initialize the associated StreamHandler.

emit(*record: LogRecord*)

Print the log.

Parameters
record (*logging.LogRecord*) – the log to print.

qosst_bob.gui.bobgui.**autoplot**(*bob:* Bob, *values: dict*)

Iterate through all figures and actualise plot if autoplot is enabled for this figure.

Parameters

- **bob** (Bob) – Bob object.
- **values** (*dict*) – current values of the GUI.

qosst_bob.gui.bobgui.**change_enable_status**(*window: Window*, *content:* QOSSTGUIContent | *List[*QOSSTGUIContent*]*, *disabled=False*)

Change the enable status of content.

Parameters

- **window** (*sg.Window*) – the window of the GUI.
- **content** (*Union[*QOSSTGUIContent*, List[*QOSSTGUIContent*]]*) – a GUI object or a list of GUI object to set as either enabled or disabled.
- **disabled** (*bool, optional*) – if True, the content is disabled. If False, the content is enabled. Defaults to False.

qosst_bob.gui.bobgui.**block_focus**(*window: Window*)

Block focus on every button of the window. This is used for popups.

Parameters
window (*sg.Window*) – pysimplegui window.

`qosst_bob.gui.bobgui.`**`popup_save_electronic_noise`**`(`*location: str*`)` → Tuple[bool, str, str]

Open a popup for the saving of the electronic noise.

This popup will have two fields, that can be added to the data container of electronic noise: the name of the detector and a comment.

Parameters
`location` (*`str`*) – location where the data is going to be saved.

Returns
return True if the operation was not cancelled, and if not cancelled, the name of the detector and a comment.

Return type
Tuple[bool, str, str]

`qosst_bob.gui.bobgui.`**`popup_save_electronic_shot_noise`**`(`*location: str*`)` → Tuple[bool, str, float | None, str]

Open a popup for the saving of the electronic and shot noise.

This popup will have three fields, that can be added to the data container of electronic noise: the name of the detector, the power of the local oscillator and a comment.

Parameters
`location` (*`str`*) – location where the data is going to be saved.

Returns
return True if the operation was not cancelled, and if not cancelled, the name of the detector, the power and a comment.

Return type
Tuple[bool, str, Optional[float], str]

`qosst_bob.gui.bobgui.`**`_create_parser`**`()` → ArgumentParser

Create the parser for the command line tool.

Returns
the created parser.

Return type
argparse.ArgumentParser

`qosst_bob.gui.bobgui.`**`main`**`()`

Main entrypoint for the GUI.

15.2 Figures

Code to plots figures for the GUI.

The way it actually is done in the GUI:

This module provides code for every plot through a plot function that is a Callable[[Bob, Axes], None].

This modules also provide a class for the figure with the init and plot methods. The list of figure is then initialized, in particular giving its name and the function that should be called to actually do the plot.

This list will be imported in the layout to create as many tabs and autoplot checkboxes required.

This list will also be imported in the gui to detect the different events.

`qosst_bob.gui.figures.plot_temporal`(*bob: Bob | None*, *axes: Axes*) → None

Plot the acquired data as a function of time.

Parameters

- **bob** (Bob) – Bob object.
- **axes** (*Axes*) – the axes where to plot the data.

`qosst_bob.gui.figures.plot_frequential`(*bob: Bob | None*, *axes: Axes*) → None

Plot the Power Spectral Density of the received data and, if available, of the shot noise and electronic noise.

Parameters

- **bob** (Bob) – Bob object.
- **axes** (*Axes*) – the axes where to plot the data.

`qosst_bob.gui.figures.plot_fft`(*bob: Bob | None*, *axes: Axes*) → None

Plot the FFT of the acquired data.

Parameters

- **bob** (Bob) – Bob object.
- **axes** (*Axes*) – the axes where to plot the data.

`qosst_bob.gui.figures.plot_tone`(*bob: Bob | None*, *axes: Axes*) → None

Plot the recovered tone.

Parameters

- **bob** (Bob) – Bob object.
- **axes** (*Axes*) – the axes where to plot the data.

`qosst_bob.gui.figures.plot_quantum_data`(*bob: Bob | None*, *axes: Axes*) → None

Plot the uncorrected quantum data.

Parameters

- **bob** (Bob) – Bob object.
- **axes** (*Axes*) – the axes where to plot the data.

`qosst_bob.gui.figures.plot_recovered`(*bob: Bob | None*, *axes: Axes*) → None

Plot the corrected quantum data.

Parameters

- **bob** (Bob) – Bob object.
- **axes** (*Axes*) – the axes where to plot the data.

`qosst_bob.gui.figures.draw_figure`(*canvas: Canvas*, *figure: Figure*) → FigureCanvasTkAgg

Creates and returns canvas to draw the figure on the GUI.

Parameters

- **canvas** (*sg.Canvas*) – the canvas of the GUI.
- **figure** (*Figure*) – the matplolib figure.

Returns

the tk canvas of the figure.

Return type
FigureCanvasTkAgg

class qosst_bob.gui.figures.**QOSSTBobGUIFigure**(*name: str*, *func: Callable[[*Bob *| None, Axes], None]*, *default_autoplot: bool = False*)

A class representing a GUI figure.

Parameters

- **name** (*str*) – name of the figure.
- **func** (*Callable[[*Bob*, Axes], None]*) – function to call to plot the figure.
- **default_autoplot** (*bool, optional*) – default value of the autoplot checkbox.. Defaults to False.

name: str

The name of the figure.

key: str

The key of the figure.

plot_key: str

The key of the plot button.

save_key: str

The key of the save button.

autoplot_key: str

The key of the autoplot checkbox.

figure: Figure | None

The matoplotlib figures.

axes: Axes | None

The matplotlib axes.

canvas: FigureCanvasTkAgg | None

The canvas to display in the GUI.

func: Callable[[*Bob* | None, Axes], None]

The function to plot the content.

default_autoplot: bool

The default value of the autoplot checkbox.

init_figure(*window: Window*)

Initialize the figure, the axes and make a dummy plot.

Parameters
window (*sg.Window*) – GUI window.

plot(*bob:* Bob)

Actualise the plot

Parameters
bob (Bob) – Bob object.

save(*path: PathLike*)

Save figure to path.

Parameters

path (*PathLike*) – path to save the figure.

qosst_bob.gui.figures.**all_figures = [<qosst_bob.gui.figures.QOSSTBobGUIFigure object>, <qosst_bob.gui.figures.QOSSTBobGUIFigure object>, <qosst_bob.gui.figures.QOSSTBobGUIFigure object>, <qosst_bob.gui.figures.QOSSTBobGUIFigure object>, <qosst_bob.gui.figures.QOSSTBobGUIFigure object>, <qosst_bob.gui.figures.QOSSTBobGUIFigure object>]**

List of all figures of the GUI.

15.3 Layout content

Enumerations of content in the GUI, and definition of some constants.

qosst_bob.gui.layout_content.**THEME = 'DarkGrey14'**

The used theme.

class qosst_bob.gui.layout_content.**QOSSTGUIContent**(*value*, *names=None*, *, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

A generic object for enumeration of content in the GUI.

class qosst_bob.gui.layout_content.**QOSSTGUIActions**(*value*, *names=None*, *, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Enumeration of actions in the GUI (i.e. buttons).

class qosst_bob.gui.layout_content.**QOSSTGUIText**(*value*, *names=None*, *, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Enumerator of updatable text content in the GUI.

class qosst_bob.gui.layout_content.**QOSSTGUIInput**(*value*, *names=None*, *, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Enumeration of inputs (i.e. text input, checkboxes and selects) in the GUI.

15.4 Layout

Layout for Bob gui.

CHAPTER

SIXTEEN

OPTIMIZATION

Optimization submodules of Bob.

Also contains the Updater abstract class.

class qosst_bob.optimization.**Updater**(*args: Namespace*, *bob:* Bob, *config: Configuration*)

An abstract class for updaters for optimization.

Parameters

- **args** (*argparse.Namespace*) – arguments passed to the command line.
- **bob** (Bob) – Bob object to request changes to Alice.
- **config** (Configuration) – configuration object to change the parameters on Bob side.
- **args** – arguments passed to the command line.
- **bob** – Bob object to request changes to Alice.
- **config** – configuration object to change the parameters on Bob side.

args: Namespace

The arguments of the command line.

bob: ***Bob***

The class of Bob, to request parameter changes to Alice

config: Configuration

The configuration, to change the parameter on Bob side.

round: int

A counter to keep memory of the turn

abstract _init_parameters()

This function is called at the end of init and should initialize the parameters arrays.

abstract number_of_rounds() → int

Return a number of rounds the script should do.

Returns
number of rounds of the experiment.

Return type
int

abstract update() → Dict

This function should

- update the parameter(s) on Bob side
- request the parameter(s) to be changed to Alice
- return a dict with the name of parameters as key and the new value as value

Returns
dict with the name of parameters as key and the new value as value

Return type
Dict

`abstract name()` → str

Name of the updater. To be used in the name of the saved file.

Returns
name of the updater.

Return type
str

16.1 Optimize

Script to optimize the excess noise over a DSP parameter.

`qosst_bob.optimization.optimize.optimize(args: Namespace, config_path: str)`

Launch Bob and start optimizing given a specific updater.

Save the results and plot depending on the arguments.

Parameters

- **args** (*argparse.Namespace*) – arguments int the command line.
- **config** (Configuration) – configuration object.

16.2 Commands

Entrypoint for the optimization script

`qosst_bob.optimization.commands._create_parser()` → ArgumentParser

Create the parser for the optimization module command.

Subcommands:

- xi-vs-va
- roll-off
- pilots-amplitude
- conversion-factor
- baud-rate
- subframe-size
- frequency-cutoff-tone
- frequency-shift

- pilot-difference-tone

Returns
parser for the optimization module command.

Return type
argparse.ArgumentParser

`qosst_bob.optimization.commands.main()`

Main function of the script. Entrypoint of the script.

16.3 Updaters

Module containing all the updaters for the optimize script.

16.3.1 Average tone size

Updater to optimize the excess noise while varying the size of averaging for the tone.

class `qosst_bob.optimization.updaters.average_tone_size.AverageToneSizeUpdater`(*args: Namespace*, *bob: Bob*, *config: Configuration*)

Experiments to measure the excess noise variations as a function of the average tone size for the phase correctionat Bob side.

Parameters

- **args** (*argparse.Namespace*) – arguments passed to the command line.
- **bob** (Bob) – Bob object to request changes to Alice.
- **config** (Configuration) – configuration object to change the parameters on Bob side.

`_init_parameters()`

Generate the array of sizes.

`number_of_rounds()` → int

Return the number of rounds, which is the length of the array of sizes.

Returns
number of rounds.

Return type
int

`update()` → Dict

Update the parameter by changing the phase filtering size at Bob side only.

Returns
dict with the new value of the phase filtering size.

Return type
Dict

name() → str

Name of the updater. To be used in the name of the saved file.

Returns
name of the updater.

Return type
str

16.3.2 Baud rate

Updater to optimize the excess noise while varying the baud rate.

`class qosst_bob.optimization.updaters.baud_rate.`**BaudRateUpdater**(*args: Namespace*, *bob:* Bob, *config: Configuration*)

Experiments to measure the excess noise variations as a function of the baud rate of the symbols.

Parameters

- **args** (*argparse.Namespace*) – arguments passed to the command line.
- **bob** (Bob) – Bob object to request changes to Alice.
- **config** (Configuration) – configuration object to change the parameters on Bob side.

_init_parameters()

Generate the array of baud rates.

number_of_rounds() → int

Return the number of rounds, which is the length of the array of baud rates.

Returns
number of rounds.

Return type
int

update() → Dict

Update the parameter by changing the symbol rate at Alice and Bob sides.

Returns
dict with the new value of the symbol rate.

Return type
Dict

name() → str

Name of the updater. To be used in the name of the saved file.

Returns
name of the updater.

Return type
str

16.3.3 Conversion factor

Updater to optimize the excess noise while varying the roll-off.

class qosst_bob.optimization.updaters.conversion_factor.**ConversionFactorUpdater**(*args: Namespace*, *bob: Bob*, *config: Configuration*)

Experiments to measure the excess noise variations as a function of the variance of Alice's modulation.

Parameters

- **args** (*argparse.Namespace*) – arguments passed to the command line.
- **bob** (Bob) – Bob object to request changes to Alice.
- **config** (Configuration) – configuration object to change the parameters on Bob side.

_init_parameters()

Generate the array of conversion factors.

number_of_rounds() → int

Return the number of rounds, which is the length of the array of conversion factors.

Returns
number of rounds.

Return type
int

update() → Dict

Update the parameter by changing the conversion factor at Alice side only.

Returns
dict with the new value of the conversion factor.

Return type
Dict

name() → str

Name of the updater. To be used in the name of the saved file.

Returns
name of the updater.

Return type
str

16.3.4 Frequency cutoff tone

Updater to optimize the excess noise while varying the cutoff for the filter of the tone.

class qosst_bob.optimization.updaters.frequency_cutoff_tone.**FrequencyCutoffToneUpdater**(*args: Namespace*, *bob: Bob*, *config: Configuration*)

Experiments to measure the excess noise variations as a function of the cutoff for the filtering of the tone at Bob side.

Parameters

- **args** (*argparse.Namespace*) – arguments passed to the command line.
- **bob** (Bob) – Bob object to request changes to Alice.
- **config** (Configuration) – configuration object to change the parameters on Bob side.

_init_parameters()

Generate the array of cut-offs.

number_of_rounds() → int

Return the number of rounds, which is the length of the array of cutoffs.

Returns
number of rounds.

Return type
int

update() → Dict

Update the parameter by changing the cutoff for the tone filtering at Bob side only.

Returns
dict with the new value of the cutoff.

Return type
Dict

name() → str

Name of the updater. To be used in the name of the saved file.

Returns
name of the updater.

Return type
str

16.3.5 Frequency shift

Updater to optimize the excess noise while varying the frequency shift.

class qosst_bob.optimization.updaters.frequency_shift.**FrequencyShiftUpdater**(*args: Namespace*, *bob: Bob*, *config: Configuration*)

Experiments to measure the excess noise variations as a function of the frequency shift of the symbols.

Parameters

- **args** (*argparse.Namespace*) – arguments passed to the command line.
- **bob** (Bob) – Bob object to request changes to Alice.
- **config** (Configuration) – configuration object to change the parameters on Bob side.

_init_parameters()

Generate the array of frequency shifts.

number_of_rounds() → int

Return the number of rounds, which is the length of the array of frequency shifts.

Returns
number of rounds.

Return type
int

update() → Dict

Update the parameter by changing the frequency shift at Alice and Bob sides.

Returns
dict with the new value of the frequench shift.

Return type
Dict

name() → str

Name of the updater. To be used in the name of the saved file.

Returns
name of the updater.

Return type
str

16.3.6 Pilot difference

Updater to optimize the excess noise while varying the difference of pilot frequency.

class qosst_bob.optimization.updaters.pilot_difference.**PilotDifferenceUpdater**(*args: Namespace*, *bob: Bob*, *config: Configuration*)

Experiments to measure the excess noise variations as a function of the difference of frequency between the two pilots. The first pilot will be left untouched.

Parameters

- **args** (*argparse.Namespace*) – arguments passed to the command line.
- **bob** (Bob) – Bob object to request changes to Alice.
- **config** (Configuration) – configuration object to change the parameters on Bob side.

_init_parameters()

Generate the array of differences.

number_of_rounds() → int

Return the number of rounds, which is the length of the array of differences.

Returns
number of rounds.

Return type
int

update() → Dict

Update the parameter by changing the frequencies of the pilot. The first frequency stays the same but the second one is updated with the new difference. It is done at Alice and Bob sides.

Returns
dict with the new values of the frequencies of the pilots.

Return type
Dict

name() → str

Name of the updater. To be used in the name of the saved file.

Returns
name of the updater.

Return type
str

16.3.7 Pilots amplitude

Updater to optimize the excess noise while varying the amplitude of the pilots.

class qosst_bob.optimization.updaters.pilots_amplitude.**PilotsAmplitudeUpdater**(*args: Namespace*, *bob:* Bob, *config: Configuration*)

Experiments to measure the excess noise variations as a function of the amplitude of pilots.

Parameters

- **args** (*argparse.Namespace*) – arguments passed to the command line.
- **bob** (Bob) – Bob object to request changes to Alice.
- **config** (Configuration) – configuration object to change the parameters on Bob side.

_init_parameters()

Generate the array of amplitudes.

number_of_rounds() → int

Return the number of rounds, which is the length of the array of amplitudes.

Returns
number of rounds.

Return type
int

update() → Dict

Update the parameter by changing the amplitudes of pilots at Alice and Bob sides.

Returns
dict with the new value of the amplitudes.

Return type
Dict

name() → str

Name of the updater. To be used in the name of the saved file.

Returns
name of the updater.

Return type
str

16.3.8 Roll off

Updater to optimize the excess noise while varying the roll-off.

class qosst_bob.optimization.updaters.roll_off.**RollOffUpdater**(*args: Namespace*, *bob:* Bob, *config: Configuration*)

Experiments to measure the excess noise variations as a function of the variance of Alice's modulation.

Parameters

- **args** (*argparse.Namespace*) – arguments passed to the command line.
- **bob** (Bob) – Bob object to request changes to Alice.
- **config** (Configuration) – configuration object to change the parameters on Bob side.

_init_parameters()

Generate the array of roll-offs.

number_of_rounds() → int

Return the number of rounds, which is the length of the array of roll-offs.

Returns
number of rounds.

Return type
int

update() → Dict

Update the parameter by changing the roll-off factor at Alice and Bob sides.

Returns
dict with the new value of the roll-off.

Return type
Dict

`name()` → str

Name of the updater. To be used in the name of the saved file.

Returns
name of the updater.

Return type
str

16.3.9 Subframe size

Updater to optimize the excess noise while varying the size of subframes.

`class qosst_bob.optimization.updaters.subframe_size.SubframeSizeUpdater`(*args: Namespace*, *bob:* Bob, *config: Configuration*)

Experiments to measure the excess noise variations as a function of the subframe size at Bob side.

Parameters

- **args** (*argparse.Namespace*) – arguments passed to the command line.
- **bob** (Bob) – Bob object to request changes to Alice.
- **config** (Configuration) – configuration object to change the parameters on Bob side.

`_init_parameters()`

Generate the array of subframe sizes.

`number_of_rounds()` → int

Return the number of rounds, which is the length of the array of subframe sizes.

Returns
number of rounds.

Return type
int

`update()` → Dict

Update the parameter by changing the subframe size at Bob side only.

Returns
dict with the new value of the subframe size.

Return type
Dict

`name()` → str

Name of the updater. To be used in the name of the saved file.

Returns
name of the updater.

Return type
str

16.3.10 Xi versus Va

Updater to optimize the excess noise while varying the variance.

`class qosst_bob.optimization.updaters.xi_versus_va.XiVsVaUpdater(args: Namespace, bob: Bob, config: Configuration)`

Experiments to measure the excess noise variations as a function of the variance of Alice's modulation.

Parameters

- **args** (*argparse.Namespace*) – arguments passed to the command line.
- **bob** (Bob) – Bob object to request changes to Alice.
- **config** (Configuration) – configuration object to change the parameters on Bob side.

`_init_parameters()`

Generate the array of variances.

`number_of_rounds()` → int

Return the number of rounds, which is the length of the array of variances.

Returns
number of rounds.

Return type
int

`update()` → Dict

Update the parameter by changing Alice's variance at Alice side only.

Returns
dict with the new value of the variance.

Return type
Dict

`name()` → str

Name of the updater. To be used in the name of the saved file.

Returns
name of the updater.

Return type
str

CHAPTER

SEVENTEEN

PARAMETERS ESTIMATION

Module holding estimators for Bob.

17.1 Base

Define abstract class for estimators.

`qosst_bob.parameters_estimation.base.`**`complex_to_real`**(*input_data: ndarray*) → ndarray

Transform the input data of a complex np array of size n to a real np array of size 2n such that if the input data is [a_1+i*b_1, a_2+i*b_2, . . . , a_n+i*b_n] then the output array is [a_1, b_1, a_2, b_2, . . . , a_n, b_n].

Parameters

input_data (*np.ndarray*) – the input complex array of size n.

Returns

the output real array of size 2n.

Return type

np.ndarray

class `qosst_bob.parameters_estimation.base.`**`BaseEstimator`**

Base abstract estimator.

abstract static estimate(*alice_symbols: ndarray*, *bob_symbols: ndarray*, *alice_photon_number: float*, *electronic_symbols: ndarray*, *electronic_shot_symbols: ndarray*) → Tuple[float, float, float]

Estimate the transmittance and excess noise given the symbols of Alice and Bob, symbols for the shot noise and electronic noise and the avarage photon number at Alice's output.

Transmittance should be here understood as total transmittance hence eta * T.

Parameters

- **alice_symbols** (*np.ndarray*) – symbols sent by Alice.
- **bob_symbols** (*np.ndarray*) – symbols received by Bob, after DSP.
- **alice_photon_number** (*float*) – average number of photon at Alice's output.
- **electronic_symbols** (*np.ndarray*) – electronic noise data after equivalent DSP.
- **electronic_shot_symbols** (*np.ndarray*) – electronic and shot noise data, after equivalent DSP.

Returns

tuple containing the transmittance, the excess noise at Bob side and the electronic noise.

Return type
Tuple[float, float, float]

`class qosst_bob.parameters_estimation.base.DefaultEstimator`

Default estimator.

`static estimate`(*alice_symbols: ndarray*, *bob_symbols: ndarray*, *alice_photon_number: float*, *electronic_symbols: ndarray*, *electronic_shot_symbols: ndarray*) → Tuple[float, float, float]

Estimate the transmittance, excess noise and electronic noise by using the covariance method.

Parameters

- **alice_symbols** (*np.ndarray*) – symbols sent by Alice.
- **bob_symbols** (*np.ndarray*) – symbols received by Bob, after DSP.
- **alice_photon_number** (*float*) – average number of photon at Alice's output.
- **electronic_symbols** (*np.ndarray*) – electronic noise data after equivalent DSP.
- **electronic_shot_symbols** (*np.ndarray*) – electronic and shot noise data, after equivalent DSP.

Returns
tuple containing the transmittance, the excess noise at Bob side and the electronic noise.

Return type
Tuple[float, float, float]

CHAPTER

EIGHTEEN

TOOLS

Tools for Bob module of QOSST.

18.1 Calibrate eta voltage

Script to calibrate eta for detectors with monitoring output in voltage.

class qosst_bob.tools.calibrate_eta_voltage.**CalibrateEtaVoltageData**(*power_values: ndarray*, *voltage1_values: ndarray*, *voltage2_values: ndarray*)

Data container for the calibration of eta.

Parameters

- **power_values** (*np.ndarray*) – array of optical powers.
- **voltage1_values** (*np.ndarray*) – array of voltages on monitoring output 1.
- **voltage2_values** (*np.ndarray*) – array of voltages on monitoring output 2.

class qosst_bob.tools.calibrate_eta_voltage.**Configuration**(*gain: float*, *voltmeter1_device: str = 'qosst_hal.voltmeter.FakeVoltMeter'*, *voltmeter2_device: str = 'qosst_hal.voltmeter.FakeVoltMeter'*, *voltmeter1_location: str = ''*, *voltmeter2_location: str = ''*, *voltmeter1_timeout: int = 1000*, *voltmeter2_timeout: int = 1000*, *powermeter_device: str = 'qosst_hal.powermeter.FakePowerMeter'*, *powermeter_location: str = ''*, *voa_class: str = 'qosst_hal.voa.FakeVOA'*, *voa_location: str = ''*, *voa_start_value: float = 0.0*, *voa_end_value: float = 5.0*, *voa_step_value: float = 0.05*, *beam_splitter_conversion_factor_pm_to_bob: float = 1*)

Configuration object for the eta voltage script.

qosst_bob.tools.calibrate_eta_voltage.**calibration_eta_voltage**(*args: Namespace*)

Measure eta using monitoring voltage.

Parameters

args (*argparse.Namespace*) – the arguments given to the command line script.

18.2 Commands

Commands for Bob tools submodule.

`qosst_bob.tools.commands._create_parser()` → ArgumentParser

Create the parser for bob tools.

Commands:

eta_voltage gain

Returns

the main parser.

Return type

argparse.ArgumentParser

`qosst_bob.tools.commands.main()`

Main entrypoint of the command.

CHAPTER

NINETEEN

TRANSMITTANCE

Experiment to measure the transmittance while varying attenuation on the channel.

qosst_bob.transmittance.**_create_parser**() → ArgumentParser

Create parser for the transmittance experiment.

Returns
parser for the transmittance experiment.

Return type
argparse.ArgumentParser

qosst_bob.transmittance.**main**()

Main entrypoint of the script.

CHAPTER

TWENTY

UTILS

Util functions for qosst-bob.

`qosst_bob.utils.heatmap`(*data_x: ndarray*, *data_y: ndarray*, *x_label='X'*, *y_label='Y'*, *title='Heatmap'*, *cmap: Colormap | str = 'rainbow'*, *axes: Axes | None = None*, *clear: bool = True*) → Tuple[Figure, Axes]

Draw a heatmap from data_x and data_y.

Parameters

- **data_x** (*np.ndarray*) – the data to be put in the x-axis of the heatmap.
- **data_y** (*np.ndarray*) – the data to be put in the y-axis of the heatmap.
- **x_label** (*str, optional*) – the label of the x-axis. Defaults to "X".
- **y_label** (*str, optional*) – the label of the y-axis. Defaults to "Y".
- **title** (*str, optional*) – the title of the figure. Defaults to "Heatmap".
- **cmap** (*Union[matplotlib.colors.Colormap, str], optional*) – the colormap to use. Defaults to rainbow.
- **axes** (*matplotlib.axes.Axes, optional*) – use those axes to make the plot. Defaults to None.
- **clear** (*bool*) – if True and if an axe was given, clear this axe.

Returns
the figure and the axe.

Return type
Tuple[matplotlib.figure.Figure, matplotlib.axes.Axes]

`qosst_bob.utils.heatmap_complex`(*data: ndarray*, *x_label='X'*, *y_label='Y'*, *title='Heatmap'*, *cmap: Colormap | str = 'rainbow'*, *axes: Axes | None = None*, *clear: bool = True*) → Tuple[Figure, Axes]

Draw a heatmap from data, using data.real as the values for the x-axis and data.imag for the y-axis.

Parameters

- **data** (*np.ndarray*) – the complex data to be put in the x-axis (real part) and y-axis (imag part) of the heatmap.
- **x_label** (*str, optional*) – the label of the x-axis. Defaults to "X".
- **y_label** (*str, optional*) – the label of the y-axis. Defaults to "Y".
- **title** (*str, optional*) – the title of the figure. Defaults to "Heatmap".

- **cmap** (*Union[matplotlib.colors.Colormap, str], optional*) – the colormap to use. Defaults to rainbow.
- **axes** (*matplotlib.axes.Axes, optional*) – use those axes to make the plot. Defaults to None.
- **clear** (*bool*) – if True and if an axe was given, clear this axe.

Returns
the figure and the axe.

Return type
Tuple[matplotlib.figure.Figure, matplotlib.axes.Axes]

CHAPTER

TWENTYONE

LICENSE

The QOSST project is distributed under the GNU General Public License version 3. Full text below.

21.1 GNU GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. https://fsf.org/

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

21.1.1 Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

21.1.2 TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

- a) The work must carry prominent notices stating that you modified it, and giving a relevant date.
- b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".
- c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.
- d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

- a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
- b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.
- c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.
- d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.
- e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

- a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
- b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
- c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or
- d) Limiting the use for publicity purposes of names of licensors or authors of the material; or
- e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

- f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered “further restrictions” within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An “entity transaction” is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party’s predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A “contributor” is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor’s “contributor version”.

A contributor’s “essential patent claims” are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, “control” includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor’s essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a “patent license” is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To “grant” such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. “Knowingly relying” means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient’s use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is “discriminatory” if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

CHAPTER

TWENTYTWO

CONTRIBUTING

QOSST is an open source project and contributions are welcomed.

As the contribution guidelines are the same for all the QOSST projects, check out the guidelines .

PYTHON MODULE INDEX

INDEX

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X