CNRS
Centre National de la Recherche Scientifique

INFN
Istituto Nazionale di Fisica Nucleare

Data quality model for Advanced Virgo

The Virgo detector characterization group

VIR-0261A-15
Date : June 2, 2015

Abstract: This note describes the model developed to deliver data quality (DQ) products to Advanced detectors gravitational-wave searches. This model differs from what was achieved during the initial Virgo era. In particular, in the case of transient searches, the model has been re-visited, generalized, optimized and simplified in many aspects. This note introduces a new terminology and, in this context, defines the DQ products the Virgo detchar group plans to produce. This document also specifies the DQ implementation which is foreseen for online and offline analysis pipelines. This DQ model greatly relies on a close interaction between the Virgo detchar group and the search groups. This note presents what is expected from search groups to successfully implement the Advanced Virgo DQ model.

VIRGO Collaboration
EGO - Via E. Amaldi - I-56021 S. Stefano a Macerata, Cascina (Pisa)
Secretariat: Telephone (39) 050 752 511 - FAX (39) 050 752 550 - e-mail perus@ego-gw.it

Contents

1 Introduction

When performing a GW search, 2 data quality (DQ) inputs are required: the time segments over which to run the pipeline and a system able to identify what events result from noise and should be discarded. The Advanced Virgo DQ model was designed to provide these two inputs to search pipelines.

The first ingredient is common to all searches. The valid segments are, more or less[1], independent from the analysis which is conducted. The data segments labeled as "science" correspond to time periods when the detector is locked and the when the detector operator declared this lock to be stable. These segments needs to be slightly adjusted to insure a nominal sensitivity of the detector. This is achieved by removing data segments suffering from severe noise disturbances resulting from major detector malfunctions[2]. Section 2 describes the state-flag system used to perform this adjustment. It also presents how this system is implemented for low-latency and offline searches.

The second ingredient is search-dependent. One must distinguish transient searches (Burst/CBC) from analyses searching for continuous signals (CW/Stochastic). The first category is limited by short-duration noise events while the second category is affected by long-lasting spectral features. A veto system can be constructed to discard background events limiting the sensitivity of transient searches. Many transient searches are conducted in the LIGO-Virgo search groups, every time the background is different. Vetoes must be optimized to match the search specificities. For this reason, the Virgo detchar group's policy is to no longer provide pre-built veto segments. Instead, veto optimization tools were designed to generate search-specific rejection procedures. These tools primarily use the information provided by auxiliary signals monitoring the detector and its environment. Combined with the information contained in the background of a given GW search, powerful vetoes can be tailored to surgically discard spurious events. This veto strategy deeply relies on a close collaboration between the Virgo detchar group and the search groups. Section 3 defines this veto model and explains how such a strategy can be implemented for both offline and online searches.

The sensitivity to stochastic and continuous GW signals is impacted by the presence of narrow-band noise features called spectral lines. The Virgo detchar group aims at detecting, following over time, characterizing, and identifying the source of all spectral lines present in the Virgo data spectrum. This work is presented in Sec. 4. This section also explains how this information can be accessed by search groups.

2 State flags

2.1 State flag definitions

The state flags are primarily used to define time segments in which the Virgo detector is known to operate in nominal conditions and when the data can be safely processed by GW search

[1] Valid segments might differ due a different pipeline response to hardware injections.

[2] In the past, CAT1 DQ flags were used to adjust science segments.

pipelines. A selection of state flags must be combined to define valid data segments over which to conduct a GW search. Some additionnal state flags are produced to monitor the environment (seismic, acoustic...) and some detector precesses. A state flag can take three values: 1 = the flag is active, 0 = the flag is not active, -1 = the flag value is unknown. State flags can be divided into four categories:

1. The **sub-system status** flags (SSS-flags) indicate when a detector sub-system is in a given operating mode. In general, a 'negative' logic is adopted, meaning that flags are active when a sub-system is in a degraded mode. The name of a SSS-flag indicates the name of the sub-system and the mode which is monitored (e.g. the input mode-cleaner not being locked: `V1:IMC_NOT_LOCKED`). Some SSS-flags can be defined as a combination of multiple SSS-flags, for example, `V1:ITF_LOCKED` or `V1:ITF_SCIENCEMODE`.

2. The **process** flags (PROC-flags) monitor aspects connected to the processing of detector data. An important example of such processes is the analysis of the detector output signal used to compute the calibrated strain data $h(t)$: the reconstruction process (HRec) generates PROC-flags to check the quality of the reconstruction or the validity of the calibration parameters. DAQ processes are also monitored to track missing frames or gaps in the data streams. Finally some PROC-flags are associated to coil or photodiode saturations. The PROC-flag names should be self-explanatory, defining what aspect is monitored.

3. The **environment** flags (ENV-flags) are constructed using environmental sensor signals. They tag time periods characterized by large deviations from "quiet" conditions, in which case the flag is inactive. For example, these flags monitor the seismic activity, the acoustic environment, the weather conditions etc.

4. The **hardware-injection** (HI-flags) flags are used to tag time segments where GW signals are physically injected into the detector. When set to 1, a HI-flag indicates that a signal is artificially present in the $h(t)$ data stream. The flag name must specify which kind of signal is injected. For example: `V1:INJECTION_CBC` or `V1:INJECTION_BURST`. Some HI-flags can be defined as a combination of multiple HI-flags, for example `V1:INJECTION_BLIND` = `V1:INJECTION_BLIND_CBC` + `V1:INJECTION_BLIND_BURST`

2.2 State flag providers

State flags are mostly produced by low-latency processes. Some of them can result from offline reprocessing of the data (e.g. the $h(t)$ reconstruction flags).

Many SSS-flags are provided by the Detector Monitoring System (DMS) [1]. To be completed. Other SSS-flag providers?

The $h(t)$ reconstruction process, HRec [**?**], provides flags monitoring the different steps of the reconstruction analysis. Needs to be more specific.

The ENV-flags are mostly produced from band-limited RMS measurements using data from environmental channels. The BRMSMon tool was specifically designed to compute band-limited

RMS of multiple channels and to produce ENV-flags with a low latency. ENV-flags can also be manually reported by the scientists on shift or by the people working at Cascina, typically for an earthquake event or for very bad weather conditions.

The hardware-injection flags are provided by an Automation (Alp) server monitoring the injection system. (To be completed.)

2.3 State flag access

2.3.1 Low-latency searches

Low-latency searches must be provided with valid time segments over which to run the pipelines. A selection of state flags will be combined and saved in a 16-bits integer called the Virgo state vector. The state vector is recorded at a 1 Hz sampling rate and is saved in the low-latency frames along with the $h(t)$ data stream. The Virgo state vector channel is named `V1:STATE_VECTOR`. Describe the state vector when the bit system is defined. A low-latency pipeline must therefore read the state vector and apply a bit-mask to determine the valid time segments in which the $h(t)$ data can be safely processed.

2.3.2 Offline searches

The state flags are individually uploaded in the LIGO-Virgo segment database, DQSEGDB [2], by a low-latency ($\sim$ 15 minutes) process called SegOnline [**?**] (see also Fig.1). The Virgo detchar group provides what combination of state flags ought to be used by search pipelines to define valid time segments to process. It consists of a command line (`ligolw_segment_query_dqsegdb`) using a sequence of state flags to add and to veto. This database query generates a single list of valid time segments, which should, in principle, be common to all analyses. They might differ for injection flags.

3 Transient noise vetoes

3.1 Definitions, strategy and terminology

The output of a GW-transient search is often dominated by transient noise events (or triggers) called glitches. These glitches contribute to the background of the search and limit the sensitivity to genuine GW signals. It is possible to reduce the false alarm rate by rejecting glitches using vetoes.

To illustrate the Virgo veto implementation, here is a list of typical veto examples:

- Remove every event detected when the wind speed is greater than 70 km/h.
- Reject an event if the event peak time is less than 1 s away from a transient noise detected in a magnetic sensor between 45 and 55 Hz and with a signal-to-noise ratio (SNR) larger than 40. The source of this magnetic coupling was mitigated on Oct. 15: do not use this veto after this date.

- Reject an event detected with a central frequency of 150 ± 10 Hz and with a central time less than 1 s away from a glitch in a seismic sensor at 10 ± 5 Hz (up-conversion mechanism) and with a SNR> 50.

- Reject an event if it overlaps (in time) more than 50% of the duration of a glitch detected in an acoustic sensor.

- Reject an event if it occurs when the beam position is more than 10 mm away from its nominal position.

- Reject an event occuring during the Sunday church service when the bell is ringing at 10h00 and when the wind is blowing South.

From these examples, we see that **3 ingredients must be given to fully define a veto**:

1. A **veto recipe** listing conditions to be checked. These conditions only rely on parameters derived from instrument auxiliary signals, environmental sensing data and/or human inputs. When all conditions are met, the veto is said to be "ON". If one of the conditions fails, the veto is said to be "OFF".

2. A **veto procedure** explaining how to apply the veto to events collected by a GW search. When a veto is ON, a search event is rejected if the event parameters match what is prescribed by the veto procedure.

3. A **veto validity period** specifying when the veto recipe/procedure should be applied. Noise in the detector is changing all the time. Therefore a veto is valid only for a limited time.

In the past, the veto procedure has, more or less, always been the same. It consisted of rejecting an event if the event peak time was contained in some time window centered around the time when the veto was ON. This strategy is sub-optimal and should be re-visited for Advanced detector searches. For example, many searches target transient signals lasting several minutes (e.g. long bursts, or CBC). It is clear that rejecting a 3-minute long event because the peak time is 1 second away from a glitch lasting 2 ms in an auxiliary channel does not make sense. In other words, a veto can be ON but an event which does not match the veto procedure should not be discarded.

For Advanced detector searches, the Virgo detchar group recommends to explore new veto procedures. For example, a veto procedure could consist of requiring a minimal overlap between the event duration and the time segment when the veto is ON. Or, the veto procedure could be defined using some event parameters other than the peak time, like the event frequency, the chirp mass or the duration. Finally, the veto procedure should explicitely specify at which stage of the pipeline the veto is applied: before/after the selection cuts (like the χ^2 consistency tests)? The veto procedure depends on the output of search pipelines. As a consequence, **the elaboration of veto procedures should be studied within search groups** and communicated to the Virgo detchar group.

Another important difference in the new Virgo veto model is the absence of dead-time associated to a veto. A veto should be seen as an additional rejection cut implemented in a search pipeline: **a veto rejects an event, not some live-time**. The final live-time of the search is not affected by the vetoes application and the resulting false-alarm rate can only be reduced.

The Virgo detchar group developed several analysis tools to elaborate veto recipes (see Sec. 3.2). These tools process auxiliary signals and elaborate the best "recipe" to target glitches of a given search. The glitch background of a given search is unique as it results from a sequence of search algorithms (match-filtering/power-excess, coincidence/coherence, noise rejection cuts...). To account for this background diversity, **the Virgo veto strategy is to run the veto tools for every search**. To achieve this, **two inputs are required from the search groups**:

1. The search background triggers are to be provided as they are used by the tools to tune the veto recipe.

2. The veto procedure must be defined by the search groups. It is used when running the veto recipe tools.

Developing veto recipes for all LIGO-Virgo searches represents a large amount of work. For this reason the involvement of search groups is mandatory. The Virgo detchar group commits to provide user-friendly veto tools, detailed documentation, and assistance when running the tools.

Finally, the Virgo detchar group recommends to have a veto validity period as short as possible. The noise in a GW detector keeps changing over time; a veto recipe performing well at a given day may perform poorly the next day. However, elaborating a veto recipe often requires a reasonnably long stretch of data. A good balance is to tune and use a veto over 2-3 days of science data. An optimal segmentation of the data is determined by the structure of the lock segments, the presence of maintenance breaks and a monitoring of the noise stationarity. At the end of a science run, the Virgo detchar group compiles all these parameters and proposes a segmentation to be used to tune the vetoes.

3.2 Veto recipes

The Virgo detchar group developed several tools to generate veto recipes. The most important ones are the use-percentage veto (`UPV`) and `EXCAVATor`. These tools are looking for complementary noise coupling mechanisms. It has been shown that all the vetoes developed in the past Virgo science runs can be replaced by recipes optimized with `UPV` and `EXCAVATor`[3]. In this section we list the different approaches which can be adopted to develop and tune a veto recipe.

[3]In fact, it has been shown that the combination `UPV`+`EXCAVATor` gave better glitch rejection performance than standard Virgo vetoes.

3.2.1 State flags

A basic veto recipe consists of compiling time periods when one or a combination of state flags are active (cf. Sec 2). For example, some PROC-flags are monitoring saturations of coil currents or in the photo-diode signals. These saturations are often a source of intense glitching. Glitches can also be caused by extreme environmental disturbances. In that case, the ENV-flags could be used in a veto recipe.

3.2.2 Use-percentage veto (`UPV`)

The use-percentage veto algorithm [3, 4] isolates significant correlations between two trigger samples. The first sample is composed of triggers of a given search and the second contains noise glitches detected in an auxiliary channel (using the `Omicron` algorithm). The auxiliary glitches are binned into the frequency parameter, then, `UPV` performs a time coincidence between the two trigger samples in each bin. The coupling between the two samples is said to be significant if, in one or several frequency bins, more than half the auxiliary glitches are involved in a coincidence. A SNR threshold is applied to auxiliary glitches in each frequency bin. The threshold is incrementally raised to potentially amplify the presence of a noise coupling and the `UPV` algorithm is repeated. The output of `UPV` is a veto recipe given by a frequency-dependent SNR threshold to apply to Omicron auxiliary glitches.

For now, only a time coincidence is implemented in the `UPV` algorithm. As underlined in Sec.3.1, it might be necessary to introduce new veto procedures to match search triggers. If required by the search group, the `UPV` algorithm can be upgraded accordingly.

3.2.3 `EXCAVATor` veto

Several coupling mechanisms are known to create noise events detected by a given search, without causing any glitch in auxiliary signals. In this situation, `UPV` is blind and unable to provide any veto recipe. The `EXCAVATor` algorithm [5] was developed to look for different kinds of coupling and to complement `UPV`-based veto recipes. It compares the time of the noise event with the value (or the derivative value) of an auxiliary data stream at that time. Then, it is able to tell whether a value (or a derivative value) is systematically associated to an event found by the search. As for `UPV`, a threshold is adjusted to maximize this association. The output of `EXCAVATor` is a veto recipe corresponding to set of thresholds to apply to the auxiliary signal values and/or derivative.

For now, only a time coincidence is implemented in the `EXCAVATor` algorithm. As underlined in Sec.3.1, it might be necessary to introduce new veto procedures to match search triggers. If required by the search group, the `EXCAVATor` algorithm can be upgraded accordingly.

3.2.4 Ad-hoc veto

In some cases, it might be required to tailor specific veto recipes which are not covered by the algorithms developed by the Virgo detchar group. This often results from dedicated noise investigations and event follow-up. Any veto recipe not already covered by the standard veto

tools are of great interest for the Virgo detchar group. Indeed new veto tools could be developed to systematically search for such new coupling mechanisms. Please, provide feedback!

3.2.5 Veto performance

After a veto recipe is created, it is often convenient to measure its performance. The detchar group has a long history with quantifying veto performance [6]. A dedicated tool, called `DQperf`, has been developed to extensively test a veto recipe (and/or a veto procedure) against a set of triggers. It measures many different figures of merit and produces plots to visualize the veto's impact on the trigger set. In particular, the veto rejection performance is measured using the time-slide approach used by GW searches.

3.2.6 Veto safety

Finally, a veto recipe must be insensitive to effects caused by a GW crossing the detector, in which case the veto is said to be safe. **The veto safety should be systematically checked** by making sure that the veto recipe never makes use of channels known to be coupled to the detector's main output signal.

More quantitatively, the veto safety can be assessed *a posteriori*, using hardware injections. The number of injections matching the veto recipe must not exceed what you would expect from accidental coincidences between two Poisson distributions. The Virgo detchar group provides a tool, called `Safety`, which performs this test and indicates whether a veto recipe can be considered as safe.

3.3 Veto implementation for searches

3.3.1 Low-latency searches

The veto information is delivered to low-latency searches in the frames, along with the $h(t)$ data stream. To limit the amount of information to convey (there could be hundreds of veto recipes), veto recipes have already been applied, combined and simplified. The final product is a veto data stream as a function of time, indicating whether the veto is ON or OFF. With such a structure, it is clear that the low-latency pipeline has no other choice but to adopt a basic veto procedure solely based on time. If needed, this approach could be re-visited in the future.

However, the search specificity must be preserved when delivering vetoes: **the veto data stream is provided for each pipeline running online**. When writing this document, three pipelines are considered to process low-latency data. Therefore, three veto data streams are saved in the frames. A veto data stream consists of a PROC-type channel sampled at 100 Hz. The channel name is given by the following convention: `V1:[PIPELINE]_VETO`, where `[PIPELINE]`=`MBTA`, `GSTLAL` or `CWB`. The channel can only take 3 values: "1 = the veto is ON", "0 = the veto is OFF" or "-1 = the veto is UNKNOWN".

In Sec. 3.1, it was explained that a veto recipe is optimized using background triggers of the search to which the veto is applied. To address this point, the veto tools (`UPV`+`EXCAVATor`), described in Sec. 3.2, are used to process the background triggers collected by the search pipeline

up to then. This is performed offline and veto recipes are saved to disk. The veto recipes are then loaded by online processes and are applied to auxiliary channel data. The veto channel is then populated using the combination of all veto recipes and inserted into the frames. This is possible thanks to the Virgo online architecture which is organized around a segment processing tool called `SegOnline`, as represented in Fig.1:

- `Omicron` processes are running online producing auxiliary triggers with low latency. `Omicron` was designed to load and apply veto recipes tuned with UPV. Omicron triggers are selected according to the recipe and veto segments are sent to SegOnline.
- The `VetoMon` process is running online and ingesting raw auxiliary signals. Veto recipies designed with `EXCAVATor` are applied and veto segments are sent to SegOnline.
- If ENV-flags are used in a veto recipe, the `BRMSMon` process is able to generate a veto stream with a low latency.
- The `SegOnline` process, running online, collects all the veto streams and turns them into veto channels saved in frames. Frames are sent back to the DAQ.
- A final process collects and merges many inputs: the $h(t)$ data stream, the veto channels and the state vector. Frames are then broadcasted to the LIGO-Virgo clusters.

This sequence of actions takes less than 30 seconds to complete.

A veto recipe is only valid for a limited time as explained in Sec. 3.1. To account for noise changing over time, veto tools (`UPV`+`EXCAVATor`) are run periodically[4]. The veto recipe is then updated and loaded by online processes.

The Virgo detchar group commits to maintain this architecture during the Virgo science runs. It includes the offline periodic tuning of veto recipes, the monitoring of all online processes and the sanity checks to be performed on the final veto streams. In return, **the search groups must provide a user-friendly system to periodically retrieve triggers produced by online pipelines**.

3.3.2 Offline searches

Offline searches aim at producing the final results which will be published in a journal. The detchar input ought to be optimal, in particular when tuning the veto recipes.

For low-latency searches, a *predictive* veto approach is used. This means that the veto recipes, applied to events detected at a time t, are tuned using events detected at a time $t-2$ or 3 days. With such a delay, it is possible that noise conditions changed. For offline searches it is possible to tune and apply the veto recipes over the same data set. Moreover, one can take advantage from the experience gained during the science run: the veto validity periods can be better adjusted, new veto techniques can be introduced and so on. Finally, the veto procedure for low-latency searches had to be simplified and only the time parameter could

[4]Every 2 or 3 days? To be defined.

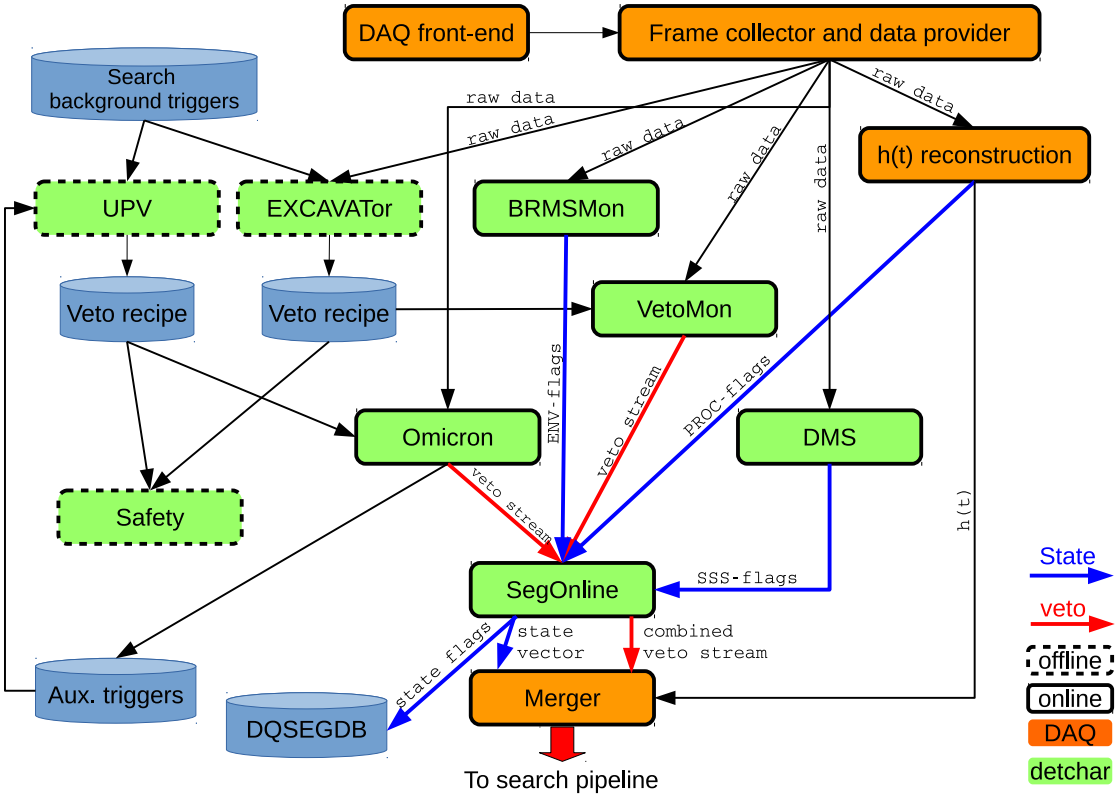

Figure 1: Online architecture used to produce data quality products for low-latency searches.

be used to reject glitches. For offline searches, it is highly recommended to develop and use multi-parameter veto procedures to better target noise glitches.

The veto implementation for offline searches can be summarized by the following steps:

1. Elaborate a veto procedure based on the search event parameters. It should also specify where, in the pipeline, the vetoes should be applied.
2. Collect background triggers where the vetoes are to be applied.
3. Divide the trigger sample into time chunks (lock segments, maintenance breaks...) defining validity periods. These time chunks are provided by the Virgo detchar group.
4. For each time chunk, measure the state flags (stored in DQSEGDB) performance with the `DQperf` algorithm and select successful veto recipes (cf. Sec. 3.2.1).
5. For each time chunk, run the veto tools (`UPV` and `EXCAVATor`). The veto recipes will be saved on disk (cf. Sec. 3.2.2 and Sec. 3.2.3).
6. Run the search pipeline applying the vetoes, following the veto procedure defined in step 1.

The Virgo detchar group plans to perform this list of tasks in the case of a few searches, searches identified as flagship analyses by the search groups. However this work will only be possible in a context of close interaction between groups. For "non-flagship" searches, it is the search groups' responsabilty to run the detchar tools and optimize the veto recipes for their analyses. The Virgo detchar group will provide as much assisitance as possible.

4 Spectral Noise Lines

To be written.

References

[1] F. Berni *et al.*, "The Detector Monitoring System: user manual," 2012. Virgo Technical Note VIR-0192A-12.

[2] M.-A. Bizouard *et al.*, "DQSEGDB Database Structure Proposal," 2013. Virgo Technical Note VIR-0420A-13.

[3] T. Isogai, "Used percentage veto for LIGO and virgo binary inspiral searches," *J.Phys.Conf.Ser.*, vol. 243, p. 012005, 2010.

[4] F. Robinet and N. Christensen, "The use-percentage veto algorithm," 2015.

[5] B. Swinkels, "EXCAVATor, an EXhaustive Correlator of Auxiliary-channel Values And Triggers," 2015.

[6] J. Aasi *et al.*, "The characterization of Virgo data and its impact on gravitational-wave searches," *Class.Quant.Grav.*, vol. 29, p. 155002, 2012.