ORBS, a data reduction software for the imaging Fourier transform spectrometers SpIOMM and SITELLE

T. Martin[a], L.Drissen[a] and G. Joncas[a]

[a]Dépt. de physique, de génie physique et d'optique, Université Laval, Québec, QC, Canada G1K 7P4 and Centre de recherche en astrophysique du Québec (CRAQ)

ABSTRACT

SpIOMM (*Spectromètre-Imageur de l'Observatoire du Mont Mégantic*) is still the only operational astronomical Imaging Fourier Transform Spectrometer (IFTS) capable of obtaining the visible spectrum of every source of light in a field of view of 12 arc-minutes. Even if it has been designed to work with both outputs of the Michelson interferometer, up to now only one output has been used. Here we present ORBS (*Outils de Réduction Binoculaire pour SpIOMM/SITELLE*), the reduction software we designed in order to take advantage of the two output data. ORBS will also be used to reduce the data of SITELLE (*Spectromètre-Imageur pour l'Étude en Long et en Large des raies d'Émissions*) – the direct successor of SpIOMM, which will be in operation at the Canada-France-Hawaii Telescope (CFHT) in early 2013. SITELLE will deliver larger data cubes than SpIOMM (up to 2 cubes of 34 Go each). We thus have made a strong effort in optimizing its performance efficiency in terms of speed and memory usage in order to ensure the best compliance with the quality characteristics discussed with the CFHT team. As a result ORBS is now capable of reducing 68 Go of data in less than 20 hours using only 5 Go of random-access memory (RAM).

Keywords: Fourier transform spectrometry, hyperspectral imagery, data reduction

1. INTRODUCTION

First of all we would like to remind that a Michelson interferometer delivers a changing interference pattern depending on the optical path difference between the interfering beams which is controlled by the position of one of the mirror. As this mirror is being shifted along the optical path the collected signal at the output of the interferometer produces an interferogram. In an ideal world a simple Fourier transform of this interferogram would give us the spectrum we are looking for.

The core of SpIOMM[1,2] and SITELLE[13] is a four-ports interferometer with two outputs instead of one in order to offer the possibility of correcting for the variations of the sky transmission. During an observation sequence which lasts for hours, airmass variation and transparency modifications (clouds) are acting as a slowly varying modulation on the interferogram. This results in errors in the computed spectrum (cf. figure 1). For an ideal instrument (with a perfect optical transmittance) in a four-port configuration: 100% of the incident flux is collected on the two cameras at the output ports. Accordingly, without any transmission variation of the sky the sum of the light collected on each camera remains a constant and is a measure of the total intensity of the source. We can then correct for the sky transmission by dividing the two recorded interferograms by their sum. Keeping in mind that SpIOMM produces spectral cubes, for a spectrum is *attached* to each pixel of the observed field, it follows that the preceding assertion is valid for any part of this field and consequently for each pixel of the cubes.

Following the notation used by Davis[3] and considering a monochromatic incident light of wavenumber σ and intensity I_0 interfering with an optical path difference x (cf. figure 2), the intensity viewed by the camera of the first port (the camera A) is*:

$$I_A = I_0 \eta \frac{1 + \cos(2\pi\sigma x)}{2} \tag{1}$$

Further author information: (Send correspondence to T.M.)
T.M.: E-mail: thomas.martin.1@ulaval.ca, Telephone: 1-418 656 2131 # 5641

*η represents the transmission coefficient of the optical system

The remaining energy must be found on the second output, where lies the camera B ($I_A + I_B = I_0$) thus:

$$I_B = I_0 \eta \frac{1 - \cos(2\pi\sigma x)}{2} + \text{cste} \tag{2}$$

We will now consider the other important advantage of using both output ports for we collect two times more photons which increases ideally the signal-to-noise ratio by $\sqrt{2}$. From the equations above we see immediately that we have to combine them by taking the difference of the interferograms to keep the modulation and remove most of the undesired constant terms. We then have:

$$I_A - I_B \propto I_0 \eta \cos(2\pi\sigma x) \tag{3}$$

We can now write down the core equation of the merging process of ORBS which permits to double the signal strength and correct for the variations of the sky transmission:

$$I_{\text{merged}} = \frac{I_A - I_B}{I_A + I_B} \tag{4}$$

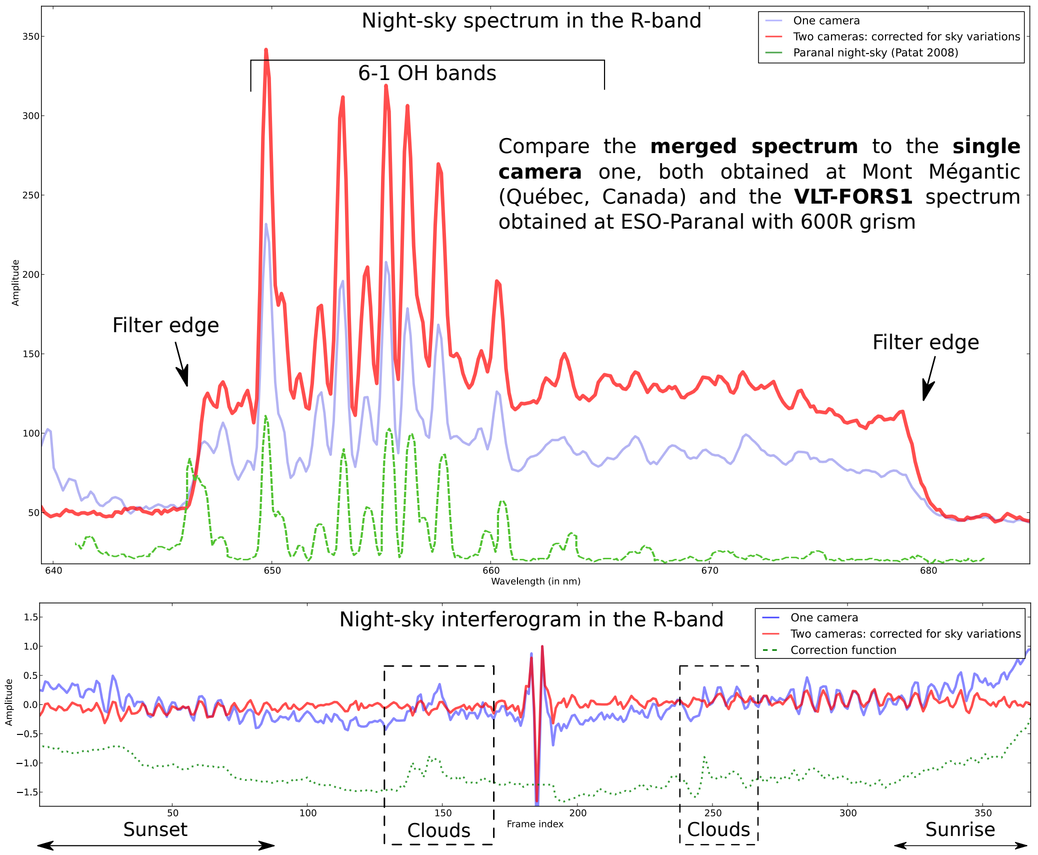
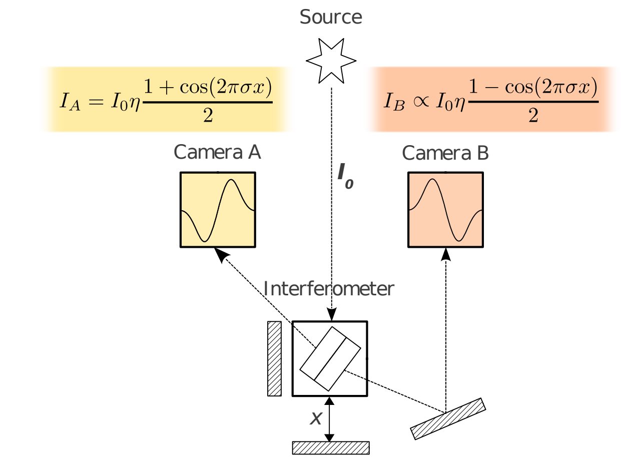


Figure 1. Comparison between raw data and data corrected for the variations of sky transmission. The lower panel shows the interferogram in each case and the upper panel shows the resulting spectrum. A spectrum obtained at the Very Large Telescope (ESO-Paranal, Chile) by the VLT-FORS1 spectrometer with a 600R grism[4] has been added to show the much better resolution of the corrected spectrum

2. ORBS

2.1 General reduction process

Computing the spectrum of a merged and corrected interferogram is one of the easiest and fastest part of the process (see Table 1). In fact the essential part of the process consists in carefully merging the raw interferograms


Source
$I_A = I_0\eta\frac{1+\cos(2\pi\sigma x)}{2}$
$I_B \propto I_0\eta\frac{1-\cos(2\pi\sigma x)}{2}$
Camera A
Camera B
I_0
Interferometer
x
Figure 2. The four ports configuration of SpIOMM and SITELLE

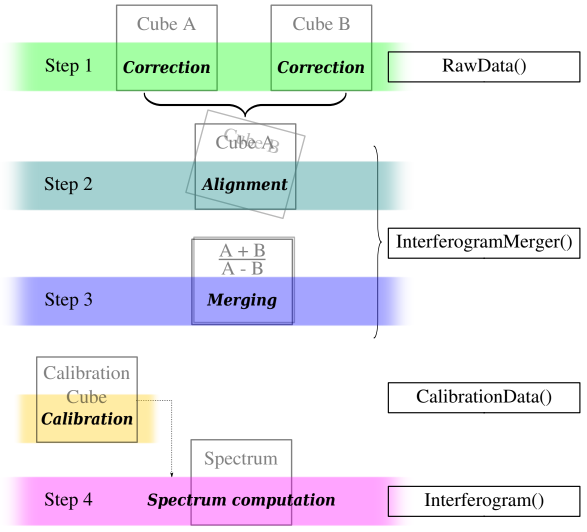


Figure 3. ORBS general reduction process

captured on the two output ports of the interferometer. The general reduction process of ORBS walks through four steps (see figure 3):

1. **Correction** of each image for bias, dark current, flat field, cosmic rays and independent alignment of the images in each data cube (along the x and y axes).
2. **Alignment** of the two data cubes in order to have a perfect correspondence between the region observed and the pixel map of each camera. There are five degrees of freedom : two shifts along the x and y axes (dx and dy), one rotation angle (dr) and two "tip-tilt" angles ($d\alpha$ and $d\beta$). This is followed by a geometrical transformation of the second cube relatively to the first one.
3. **Merging** process using equation (4) for each pixel. We thus obtain the merged and corrected interferogram.
4. **Spectrum computation** of the merged interferogram using a Fast Fourier Transform (FFT) algorithm.

2.2 Specifications

ORBS will be used by the technical team of the CFHT to reduce SITELLE's data and it thus comes with a set of specific constraints, regarding the autonomy and the computation time, with the global aim of delivering the best possible data to the scientific community. Using the set of software characteristics classified in the standard ISO/IEC 25010:2011[5] we will list the different specifications and the choices we made to ensure ORBS's conformity.

2.2.1 Functional correctness

Functional correctness (or accuracy) is certainly the most important aspect of a reduction pipeline. For ORBS accuracy means obtaining the most accurate spectral cube possible given the raw data cubes of images delivered by the cameras.

Data precision for calculation and storage Each step needs the most accurate computation and even if we are dealing with large data arrays all calculations are made using double-precision floating-point numbers (64-bit float)[6]. To avoid taking too much space on the hard drives the data generated by the different reduction processes is then stored as simple-precision floating-point numbers (32-bit float) using the FITS format[7].

Alignment processes There are two alignment processes: the first one is for aligning images of a cube. Telescope guiding is never perfect and thus light never falls in the same pixel. The other one – and certainly the most delicate one – concerns the alignment of the cubes (see 2.1). We use the Powell's optimisation method[8] over Gaussian fitted stars for both processes. The result is a mean positional difference around one tenth of a pixel over all the selected stars (generally more than 15 stars are used). Once the alignment parameters have been found the images are aligned using linear interpolation because of the errors on stars for higher interpolation degrees.

Apodization Apodization is still a sensitive question for anyone working with Fourier transforms of interferograms. Several apodization functions have been implemented in the code and the user can choose which one to use (e.g. Hann, Hamming, Kaiser and Blackmann window functions) during the Fourier transform process. But we would like to emphasize the fact that up to now no apodization seems to return a better precision on the amplitude of the emission lines. One good test can be done by looking to the [N II]λ6548 / [N II]λ6583 ratio which can be far from three on an apodized spectrum. The major disadvantage of an unapodized spectrum is the "sinc" line shape.

2.2.2 Usability

One specific constraint imposed by the CFHT team is the total autonomy of the reduction pipeline: there can't be any check or action taken by the technical team over the reduction process. The whole software must then run by itself, automatically getting inputs of data and reduction parameters, detecting stars for alignment and creating its own file architecture.

Star detection The autonomy mostly depends on an automatic star detection procedure. Basically we first look at all the most intense points in an high pass filtered image. Those points are then fitted to ensure that they are stars and not hot pixels or cosmic rays. To align the cube B relatively to the cube A we must make sure that the detected stars are the same. We then walk through three alignment steps:

1. Find a rough guess for dx, dy and dr (we assume the tip-tilt angles $d\alpha$ and $d\beta$ to be negligible) then try to find some "coupled" stars over both lists of detected stars in camera A and B and finally compute a first full set of alignment parameters.
2. Look another time for coupled stars (generally detect much more coupled stars) and compute a more precise set of alignment parameters.
3. Go through a longer optimisation routine which fits stars over the transformed image.

This routine is now robust enough to always find enough coupled stars (we set it to a minimum of 15 stars) in both cubes to ensure a perfect alignment.

inputs and outputs The `Orbs()` class (cf. figure 5) manages the inputs and outputs at all the levels of the reduction process. The reduction parameters (folding order, filter used, etc.) are stored in an option file which can then be used at each step. It also generates the file architecture. All the data created by ORBS which can be useful for debugging or checking the reduction quality (e.g. computed function of correction for the variations of sky transmission) or even scientific data (e.g. deep images) can be found in a file folder attached to the reduction process. Moreover, in order to reduce the memory load (see section 2.2.3), each reduction function writes its own outputs which can then be read by the next function.

2.2.3 Performance efficiency

The second set of constraints imposed by the CFHT concerns, on the first hand, the time required for a full reduction process (which must not exceed a day) and on the other hand, the maximum size of the data to handle during a whole process. In this section we will talk about the methods used to comply with those constraints and the results obtained for 1) the reduction of a regular SpIOMM data cube and 2) the reduction of a simulated large data cube taken as an example for SITELLE. A full-size data cube will also be taken as an extreme case. Those three sets of data have the following characteristics:

1. The "small"-size data cube corresponds to a regular SpIOMM data cube in terms of both spatial and spectral resolution. As the cameras are different the numbers of pixels is not exactly the same for the camera A (446×433, binned 3×3) and B (511×511, binned 4×4). We have chosen a cube taken with the red filter which has a total of 416 points over the given spectral range (which means 416 frames in the interferogram cube)[†].
2. The "large"-size data cube can be considered as a large cube even for SITELLE. It has an average spatial resolution in SITELLE's case (which cameras are 2048×2048 pixels wide) with a 2×2 binning (i.e. 1024×1024 pixels) but a high spectral resolution with 1088 points over the spectral range.
3. The "full"-size data cube has at the same time a high spatial resolution (no binning, 2048×2048 pixels) and a high spectral resolution of a thousand steps.

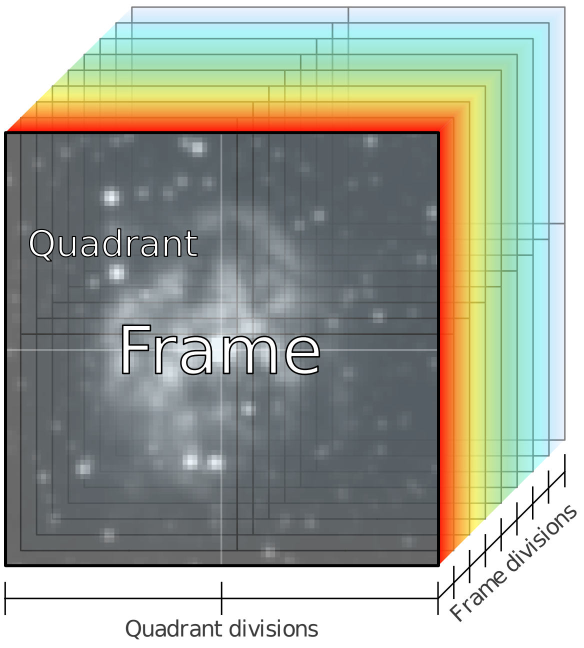


Figure 4. Type of cube divisions made to process it

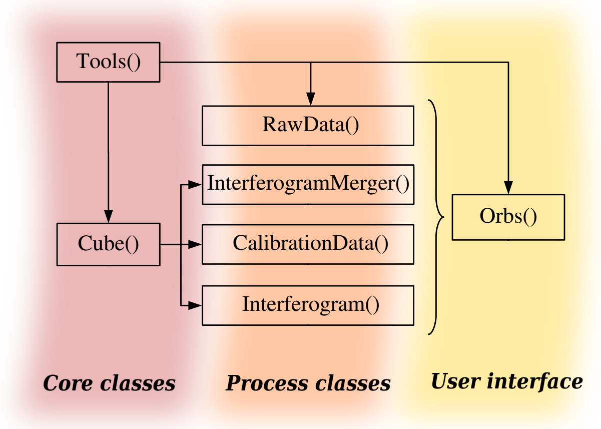


Figure 5. ORBS inheritance diagram and "onion"-like architecture.

Resource utilisation All calculations are made with a 64-bit floating point precision (see section 2.2.1). In this case a full-size SITELLE's data cube requires up to 34 Go of memory[‡] to be entirely loaded. It is clear that such an amount of data is just impossible to handle by a regular machine. We have thus chosen to process data

[†]Note that for a given number of points the resolution depends on the filter used. $R_B \sim 650$ in the blue band with 250 steps, $R_R \sim 1900$ in the red band with 416 steps and $R_V \sim 850$ in the green band with 450 steps

[‡]i.e. 2048×2048 pixels $\times$ 1000 steps of double-precision floating-point numbers (64-bit, 8 octets)

Process designation * = parallelized † = data divided by quadrants	SpIOMM (small data cube) 446×433×416 (511×511×416) Time consumption % of 34 min	 Maximum memory load % of 643 (869) Mo	SITELLE (large data cube) 1024×1024×1088 (×2) Time consumption % of 4.6 hours	 Maximum memory load % of 9.1 Go
Alignment parameters	9.1 (9.6)	5.8 (4.9)	3.7 × 2	1.0
Cosmic ray detection*†	2.6 (3.5)	22.7 (20.8)	20.3 × 2	15
Interferogram correction*	0.7 (22.9[a])	23.3 (20.9)	1.6 × 2	7.0
Total for step 1: Correction	**48.4**	**23.3**	**51.2**	**15.0**
Alignment param. of A vs B	5.7	5.8	0.4	1.0
Transformation of B*	36.7	19.3	20.9	6.7
Total for step 2: Alignment	**42.4**	**19.3**	**21.3**	**6.7**
Total for step 3: Merging*	**1.8**	**25.1**	**3.7**	**7.5**
Total for step 4: Spectrum computation*†	**7.6**	**18.4**[b]	**23.8**	**4.9**

Table 1. Time consumption analysis for the reduction of SpIOMM and SITELLE's data cubes. Data have been collected for 1) a regular-size set of data for SpIOMM (camera A: 446 × 433 × 416 and camera B: 511 × 511 × 416 – 643 Mo and 869 Mo in 64-bit float) 2) a larger set of data corresponding to a large data cube for SITELLE (1024 × 1024 × 1088 – 9.1 Go each in 64-bit float). The numbers in parenthesis stands for camera B which is different in SpIOMM case. The processor used is a 64-bit quad-core (8 threads). The total processing time is of 34 minutes for SpIOMM's data set and 4.6 hours for SITELLE's one. Note that specific processing is required to correct the images of the camera B. SITELLE's time consumption is based on the fact that the cameras are the same and that no specific processing has to be done for camera B. The maximum memory used is expressed in percentage of the maximum computational size of a whole data cube in memory (as if it was entirely loaded during the execution of the process): number of pixels × memory size of a 64-bit float (8 octets).

[a]Specific processing is required to correct the images of the camera B

[b]In this case the spectrum data cube is saved in 81 quadrants. If the spectrum data cube is saved in one file the maximum memory load reaches 123% of the computational size of one data cube

by part as we can always divide a data cube to process it (cf. figure 4). We can divide the cube either by frame (the cube is sliced along the z axis) or by "quadrant" (the cube is sliced along the x and y axes). Moreover, data is stored during the reduction process which avoid keeping a large amount of data in memory. In addition data cubes created during the reduction (outputs of the different functions) are stored as sets of individual frames which avoid the problem of loading a whole cube prior to process it. This is what we call the *frame-divided* cube concept. This is implemented by the `Cube()` class. Following this concept a data cube becomes a virtual entity which associates altogether a particular set of frames. Using this class one can get the data as if the cube was entirely loaded in memory. Data is collected from the files opened one by one. The drawback of this concept is obviously the loading time which can be longer in some processes than the calculation time (but we would like to enhance the fact that manipulating smaller data cubes speeds up array operations). You can see in Table 1 that the peak in the amount of memory utilisation occurs during the merging process and is around 15% of the computational size of a data cube in memory for a large cube. We use this percentage to express the fact that a simple process would load a whole data cube prior to process it. For ORBS, as the data is computed by part, the maximum amount of memory needed is more than 6 times smaller. The reduction of a full-size data cube would then take up to 5.1 Go of memory and only 1.4 Go for the large-size one. These results depends widely on the number of "quadrant" divisions chosen and the number of threads used: if more threads are running more memory is used[§] and for a larger number of quadrants the maximum amount of memory load is reduced but the time needed for loading data becomes very important. Those parameters must be adjusted to fit with the computer characteristics and get the best possible performances for each size of cube. Finally we would like to point out that if the user wants to have a spectral cube in only one file the last spectrum computation process must take at least an amount of memory equal to the total computational size of the processed cube. To avoid

[§]Our tests have been made using a 8 threads processor this problem for very large cubes an option has been implemented in the spectrum computation process to save the spectral cube by quadrants (only the image axes are fragmented but the spectral axis is complete so that these cubes are immediately readable by any visualisation software). When this option is used the percentage of memory used is reduced from 123% down to 19% in the case of a small data cube and can be as small as 5% in the case of a larger one.

Time behaviour By dividing the arrays of data in smaller parts we have been able to parallelize all the time consuming processes. We have chosen to work with Parallel Python module[9] for Python. The parallelization have been implemented to use as much threads[¶] as possible. For a regular number of threads (we have not tested it for more than 8 threads) the processing time is simply divided by the number of available threads. The drawback is that the amount of memory needed is also multiplied by the number of threads. A time processing analysis have been made on the data collected from the reduction of SpIOMM's data and a simulated SITELLE's data cube (cf. Table 1). On this basis we have tried to anticipate the time needed to reduce SITELLE's data assuming that there will be no need for a particular correction of the images for any of the cameras. The processing time for a large data cube is thus of 4.6 hours and up to 20 hours for a full-size data cube (which contains four times more pixels).

2.2.4 Operability

As the whole process of reduction is fully autonomous a simple command line is required for its execution. The needed inputs must be written in an option file which will then be read by the `Orbs()` class. The inputs are of two kinds:

- Files paths to the raw data frames of camera A and B, the calibration files, and the correction files (bias, dark, flat)
- Observational parameters (step size, folding order, etc.) which could be read directly in the headers of the generated files

2.2.5 Maintainability

The maintainability is an important quality for a software which is going to be used by the scientific community.

Modularity The reduction operations may change and each astronomer may be interested in creating its own reduction process to fit its own needs. ORBS has thus not only been designed in the aim of being able to execute a full reduction process for the reduction of the data obtained at the CFHT but also to permit a great variety of different reduction patterns. Orbs modularity is based on an object-oriented "onion"-like architecture (cf. figure 5). Anyone can change the reduction pattern at its convenience by using the `Orbs()` class which acts as a user-interface implementing the high level reduction methods. Most of the parameters of the processing classes can be tuned directly in `Orbs()` method calls.

Modifiability The "onion"-like architecture also facilitates method modifications for all higher level classes inherit of the core classes. This helps in getting an homogenized access to data and keeping programming core concepts:

- `Core()` implements basic input/output functions (FITS read/write, console prints and logfile feed)
- `Cube()` generates and manages the data access to the virtual *frame-divided* cubes (see section 2.2.3).

The processing classes have been divided in regards to the kind of data they manipulate (raw data, interferogram data, etc.) and they implements simple methods built over even simpler functions which again facilitates modifications, debugging and correction.

[¶]We prefer to use the concept of thread instead of processor because of the popular use of multi-core processors which handle multiple threads in the same processing unit

2.2.6 Portability

The possible multi-platform use of ORBS has encouraged us to use Python[10] which answers to most of the portability questions in addition to be a fully object-oriented programming language . It is also free and open source. The other libraries used are officials (PyFITS[11], Numpy[12]) or still maintained (Parallel Python[9]).

2.3 On some specific parts of the process

2.3.1 Spectrum computation

Here we describe the whole spectrum computation process. It walks through 8 steps:

1. Mean interferogram subtraction to suppress the zero-frequency term in the spectrum
2. Low order polynomial subtraction to suppress low frequency noise in the spectrum
3. Apodization (the user can choose which apodization function to use)
4. Zero-padding to have two times more points in the interferogram in order to keep the same resolution during the Fourier transform.
5. ZPD shift to correct for a non-centered ZPD.
6. Fast Fourier Transform of the interferogram
7. Phase correction (if the user chooses to get the real part of the spectrum with phase correction instead of the power spectrum)
8. Wavelength correction using the data obtained with the calibration cube.

2.3.2 Sky correction

As described in the introduction, correcting for variations of sky transmission means dividing the signal by the sum of the interferograms. We must do this for each pixel and correct each point of the corresponding interferogram. If it mathematically works perfectly for not too noisy data it appears – especially on faint regions – that dividing by noisy data adds too much undesired noise. One way to eliminate this noise is to compute the correction function on a larger region of the cube. Indeed, if we make the assumption that the transmission variations of the sky affect in the same way all the sources of the observed region (up to a square of 12×12 arc-minutes for SpIOMM and 11×11 arc-minutes for SITELLE[13]) we can thus combine altogether the correction functions calculated for each pixel and get a far less noisy general correction function[‖] that can be used to correct all the interferograms.

2.3.3 Detecting cosmic rays in interferograms

Detecting cosmic rays in an interferogram cube is easily done by the use of the third axis. That means working primarily on the interferogram of each pixel. the idea consists in dividing each particular interferogram by the mean interferogram of the region (e.g. a square on the image of about ten pixels wide). As yo can see in figure 6 it makes evident even a faint cosmic ray and reduces the importance of the zero path difference (ZPD) pattern at the center. Taking a few standard deviations (generally about four) above the mean of the "flatten" interferogram gives a good statistical threshold for detection.

ACKNOWLEDGMENTS

We acknowldege funding by the Canadian Foundation for Innovation, Québecs FQRNT, Canadas NSERC, and Université Laval.

[‖]For the signal-to-noise ratio is increased by $\sqrt{\text{number of pixels}}$

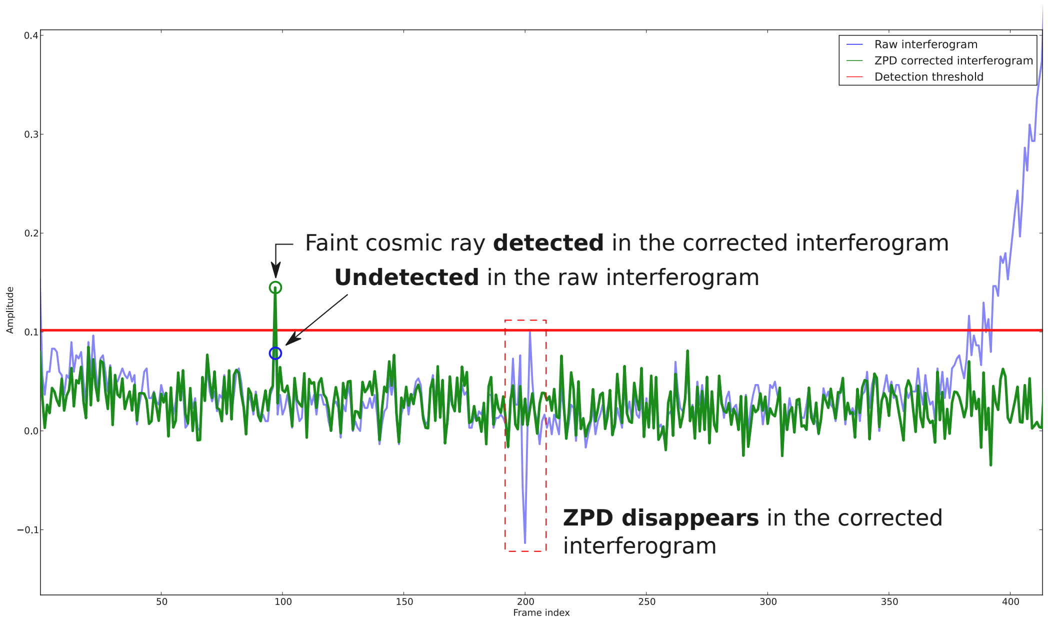


Figure 6. Detecting cosmic rays in interferograms.

REFERENCES

[1] Grandmont, F., *Développement d'un spectromètre imageur à transformée de Fourier pour l'astronomie*, PhD thesis, Université Laval (2006).

[2] Bernier, A.-P., "First results and current development of SpIOMM: an imaging Fourier transform spectrometer for astronomy," in [*Proceedings of SPIE*], **6269**, 626949–626949–9, SPIE (June 2006).

[3] Davis, S. P., Abrams, M. C., and Brault, J. W. J. W., [*Fourier transform spectrometry*], Academic Press, San Diego (2001).

[4] Patat, F., "The dancing sky: 6 years of night-sky observations at Cerro Paranal," *Astronomy and Astrophysics* **481**, 575–591 (Apr. 2008).

[5] International Organization For Standardization Iso, "ISO/IEC 25010:2011," Tech. Rep. 1, International Organization for Standardization (ISO) (2011).

[6] IEEE Computer Society, "IEEE 754 Standard for Floating-Point Arithmetic," tech. rep., IEEE Computer Society (2008).

[7] Hanisch, R. J., Farris, A., Greisen, E. W., Pence, W. D., Schlesinger, B. M., Teuben, P. J., Thompson, R. W., and Warnock, A., "Definition of the Flexible Image Transport System (FITS)," *Astronomy and Astrophysics* **376**, 359–380 (Sept. 2001).

[8] Powell, M. J. D., "An efficient method for finding the minimum of a function of several variables without calculating derivatives," *The Computer Journal* **7**, 155–162 (Feb. 1964).

[9] Vanovschi, V., "Parallel Python," (2012).

[10] Rossum, G., [*Python reference manual*], Centre for Mathematics and Computer Science, Amsterdam, The Netherlands, The Netherlands (July 1995).

[11] Barrett, P. E. and Bridgman, W. T., "PyFITS, a FITS Module for Python," in [*Astronomical Data Analysis Software and Systems IX*], Manset, N., Veillet, C., and Crabtree, D., eds., *Astronomical Society of the Pacific Conference Series* **216**, 67, ASP Conference Proceedings, Vol. 216 (2000).

[12] Ascher, D. and Dubois, P. F., "Numerical Python," *Computers in Physics* **10**(3), 262–267 (2001).

[13] Drissen, L., Bernier, A.-P., Rousseau-Nepton, L., Alarie, A., Robert, C., Joncas, G., Thibault, S., and Grandmont, F., "SITELLE: a wide-field imaging Fourier transform spectrometer for the Canada-France-Hawaii Telescope," in [*Proceedings of SPIE*], **7735**, 77350B–77350B–10 (July 2010).