GEMINI

INTERFEROMETER

COMPLETE OPERATIONAL EXAMPLE

(PYTHON AND NI-DAQ)

NIREOS SRL

Via Giovanni Durando, 39 - 20158 Milan (Italy)

info@nireos.com | www.nireos.com

This manual is continuously updated by our staff to help our customers to use the GEMINI interferometer at the top of its potential. You can always find the latest version of this manual at **www.nireos.com/downloads**

We made our best to assure that the manual is clearly written and does not contain errors, but we are aware that perfection can not be reached. Please contact us at info@nireos.com in case you spot an error or in case you find hard to understand some parts of the manual.

Subject to change without notice, download the last version at
www.nireos.com/downloads
Document Version: 1.0-A

Contents

1. Introduction

The aim of this manual is to show a step by step example on how to properly acquire spectra with NIREOS GEMINI Interferometer using Python programming language. This manual is based and tested on Python 3.7 (32 bit/64 bit) on Windows operative system. This software controls the positioner inside the interferometer, and it is based on MCSControl.dll library provided by Smaract (For further details see Smaract MCS programmers Guide manual). The MCS_Installer_3.8.9 must be correctly installed to use this software. The python scripts use external packages as: Numpy, Pandas, Scipy and Matplotlib, they must be imported.

As an example, the acquisition part is done using a National Instrument Data AcQuisition (DAQ) Card (USB-6002) and the software uses DAQmx software. The DAQ card and the DAQmx software are not included in the GEMINI interferometer.

This example contains all the steps needed to acquire successfully an input light interferogram and then compute its correspondent spectrum. This manual is intended for user with a previous knowledge of the principle of Fourier transform spectroscopy and on the GEMINI interferometer operating principle. See the *Theoretical Manual* for a detailed explanation. A basic knowledge of the Python programming language is required. All the processing codes to retrieve the light spectral information are the results of year of research in the Fourier Transform spectroscopy field and have been tested and applied to a great variety of scientific applications. Please have a look on www.nireos.com to see the wide applicability of this device.

2. Python Functions

In order to move and control the positioner, the already implemented function available in the MCSControl.dll library are used. These functions are written in C programming language, and in order to use them in Python it is necessary to use the package called *ctypes.*

Ctypes is a foreign function library for Python. It provides C compatible data types, and allows calling functions in DLLs or shared libraries.

To import the library you have to use the command ctypes.CDLL(‘ PATH and name of the library’). In our case :

```
lib = ctypes.CDLL('MCSControl.dll')
```

To call a C function it is necessary to specify the input arguments that that function receives as input through the *library.function. argtypes=[ctypes.type, ctypes.type]*.

If a pointer to the variable is needed you can pass it as *POINTER(ctypes.type)*

ctypes type	C type	Python type
c_bool	_Bool	bool (1)
c_char	char	1-character bytes object
c_wchar	wchar_t	1-character string
c_byte	char	int
c_int	int	int
c_uint	unsigned int	int
c_long	long	int
c_ulong	unsigned long	int
c_float	float	float
c_double	double	float
c_longdouble	long double	float
c_char_p	char * (NUL terminated)	bytes object or None

Also the packages *Numpy and Scipy* are needed to take advantage of their mathematical functions.

Import the package *nidaqmx* to control and acquire signals from the DAQ module.

2.1 Initialization.py

This function handle the connection to the controller and initialize the positione inside the interferometer and move the interferometer to the home position, where the delay is zero. The home position is calibrated in factory and saved in the non-volatile memory of the motor driver. The home is the position where the two replicas have zero relative delay.

In order to connect to the controller the function SA_OpenSystem is exploited, it takes as input:

- *system_locator*: string that identifies the system, it is addressed as *usb:id:<sn>*, where *sn* is the serial number of the controller which is printed on it.
- *options*: options for the initialization function. List below
- reset: the controller's settings are reset on open
- async/sync: choose between asynchronous connection, async, or synchronous one, sync

If the execution was successful the function gives *the* output:

- system_index: returns a handle to the opened system.

which is the handle to the opened system, it is needed for all the other controller's functions.

The value channel_index is already set to "0", since there is just one positioner plug to the controller.

Two communication modes can be used to establish the connection to the controller, synchronous and asynchronous. In this guide we will deal only with the synchronous case, even though it is less flexible it is simpler and for our purposes it suites better (for further knowledge about asynchronous mode see MCS - Modular Control System Programmer's Guide).

2.2 Movements.py

This file contains the methods necessary to move the positioner and get values from it.

2.2.1 move_absolute

This function moves the motor in an absolute position chosen by the user and exploits the function SA_GotoPositionAbsolute_S and it accepts as input:

- *system_index*: that is the handle to the opened system
- *channel_index*: selects the channel of the selected system
- *movement*: absolute position to move to in nanometers.

2.2.2 move_relative

This function moves the motor in a relative position with respect to the current one. It exploits the function SA_GotoPositionRelative_S and takes as input:

- *system_index*: that is the handle to the opened system
- *channel_index*: selects the channel of the selected system
- *movement*: relative position to move to in nanometers.

2.2.3 move_home

This function moves the motor in the home position, where the delay between the two replicas is zero and the absolute position is zero. It exploits the function SA_GotoPositionAbsolute_S as well, in which the absolute position as input is set to zero. It accepts as input:

- *system_index*: that is the handle to the opened system
- *channel_index*: selects the channel of the selected system
- *movement*: set to "0"

2.2.4 get_position

This function gives the current position of the positioner, in nanometers, with respect to the home position set. It exploits the function SA_GetPosition_S, it takes as input:

- *system_index*: that is the handle to the opened system
- *channel_index*: selects the channel of the selected system

If the execution of the function was successful it returns:

- the value of current position

2.2.5 get_status

This function gives the current position of the positioner, in nanometers, with respect to the home position set. It exploits the function SA_GetStatus_S, it takes as input:

- *system_index*: that is the handle to the opened system.
- *channel_index*: selects the channel of the selected system.

If the execution of the function was successful it returns:

- An integer value that describes the current status of the positioner: see MCS Programmers Guide.pdf to understand the meaning of the codes.

2.2.6 get_scale

This function retrieves the currently configured scale of the controller, in nanometers, the distance between the physical zero of the positioner and the one of the interferometer. It exploits the function SA_GetScale_S, it takes as input:

- *system_index*: that is the handle to the opened system.

- *channel_index*: selects the channel of the selected system.

If the execution of the function was successful it returns:

- An integer value that is the scale, in nanometers.

2.3 Data_acquisition

The package *nidaqmx* has to be imported. The full and detailed guide can be found at https://nidaqmx-python.readthedocs.io/en/latest/.

We will exploit the *nidaqmx.task.Task* object. Usually it is named as *task.*

The inherent function task.ai_channels.add_ai_voltage_chan("Dev_name/channel", terminal_config = TerminalConfiguration.XXX) initializes the connection and add the value of the channel to the buffer. Then the value is read throughout the function data=task.read(number_of_samples_per_channel=”x”) that return a list.

N.B. This function work only with the data acquisition systems from National Instruments. If you have another data acquisition system, you have to replace this part of the code with the one provided by the manufacturer.

2.4 Find_scan_range

The *find scan range* function returns the maximum excursion needed to achieve a specific spectral resolution at a certain wavelength. To compute the maximum excursion it uses the formula:

$$Excursion_{max} = \frac{\lambda^2}{\Delta n(\lambda)\Delta\lambda \sin\alpha}$$

Where λ is the wavelength. $\Delta n(\lambda)$ is the wavelength dependent birefringence difference between the ordinary axis and extraordinary one. α is the wedge apex angle. The birefringence is taken from a database from a .csv file. Since the values in the file are discretized, the birefringence at the wavelength λ is retrieved by interpolation.

INPUTS:

- Resolution [nm] (Float): Desired resolution at the selected wavelength in nanometers.
- Wavelength [nm] (Float): Wavelength at which the resolution is associated.

OUTPUTS:

- Start [mm] (Float): Starting position of the scan to achieve the desired Resolution [nm] at the selected Wavelength [nm]
- End [mm] (Float): Ending position of the scan to achieve the desired Resolution [nm] at the selected Wavelength [nm]

2.5 Spectral_calibration

The *Spectral Calibration* returns the polynomial coefficients of the fitting function that converts from $\frac{1}{Wavelength}$ [nm^{-1}] to spatial frequency [mm^{-1}] and vice-versa. It is possible to modify the calibration by editing the lookup table inside the file *"*_parameters_cal.txt",* where the * stands for the serial number. In the lookup table each wavelength in nanometers is associated univocally to a frequency in mm^{-1}.

N.B. the GEMINI interferometer is already calibrated when shipped. The GEMINI interferometer is designed to be an ultra-stable device; therefore, the calibration remains stable over long periods of time. Please see the *Theoretical Manual* for more details on the calibration procedure of the interferometer.

INPUTS:

- [NONE]

OUTPUTS:

- P_freq2wave [from freq. to 1/wave] (Array of float): This array contains a set of polynomial coefficients in ascending order of power. These are the polynomial coefficients of the function that converts from frequencies in mm^{-1} to 1/wavelength in nm^{-1}. Mathematically, using the set of polynomial coefficients to convert from a specific frequency f_0 to the correspondent wavelength in nm one obtains:

$$\frac{1}{Wavelength(f_0)} = P_0 + P_1 f_0$$

- P_wave2freq [from 1/wave to freq.] (Array of float): This array contains a set of polynomial coefficients in ascending order of power. These are the polynomial coefficients of the function that converts from 1/wavelength in nm^{-1} to frequencies in mm^{-1}. Mathematically, using the set of polynomial coefficients to convert from a specific wavelength $Wavelelngth_0$ to the correspondent frequency in mm^{-1} one obtains:

$$f = P_0 + P_1 \frac{1}{Wavelength_0}$$

2.6 Get_calibrated_position_axis

This function receives the encoder position array and the motor current scale value and returns the interferometric calibrated position axis in millimeters. This function eliminates artifacts due to motor linear slide imperfections. To measure the actual scale of the motor inside GEMINI interferometer use the *get_scale* function in *Movements.py*.

Thanks to the GEMINI high reproducibility, to have an interferometric calibrated position axis, it is enough to measure the interferogram of a high temporal coherent source (e.g. Lasers), in our case it is a He-Ne, over the whole scan range of the motor. The measured interferogram together with the correspondent position array as recorded by the encoder should be saved in a file with the name: "*_parameters_int.txt", * stands for the serial number. This file must be placed in the same folder of the python script.

N.B. Please note that the GEMINI interferometer is shipped already calibrated and there is in general no need to recalibrate the interferometer over a long period of time. Please Contact NIREOS customer service at info@nireos.com for hints on the calibration procedure.

The script firstly reads the positions and signal from the file "*_parameters_int.txt" i.e. the reference interferogram. It then calculates the ratio between the measured positions and the reference positions, this factor will be used for oversample the measured positions throughout interpolation. Later it interpolates the reference on the oversampled measured positions. Thanks to the function *get_real_position_axis it* calculates the calibrated axis from a reference interferogram. At the end the calibrated positions are downsampled and normalized.

INPUTS:

- Position axis [mm] (Array of float): This array contains all the position values correspondent to each acquired interferogram point.
- Scale (Integer): this is the motor scale value obtained through the *get_scale* function. This value is needed to reference the motor position to the motor physical reference.

OUTPUTS:

- Calibrated_position_axis [mm] (Array of float): This array contains the interferometrically calibrated position values.

2.7 Get_spectrum_dft

This function contains all the steps to convert an interferogram into a spectrum depending on wavelength [nm] and spatial frequency [mm^{-1}]. First the coefficients to convert from wavelength and spatial frequency are retrieved, throughout the function *spatial_calibration.* Then the interferogram mean value is subtracted from the interferogram, using a moving average method, which creates a series of averages of different subsets of the full data set, thanks to the function *movmean* . The result is multiplied by a gaussian function (the so-called apodization window). The normalized standard deviation of the gaussian window with respect to the number of points of the interferogram is a value that has to be set in the script, the variable *apodization width*. Then it calculates the Fourier transform of the interferogram as a function of the position axis to get the spectrum. It gives as output the arrays of the spatial frequencies, wavelengths and the spectrum.

INPUTS:

- Apodization_width (Float): the normalized standard deviation with respect to the number of points of the interferometric signal.
- Interferogram (Array of float): the interferometric signal, the so called interferogram to be Fourier transformed.
- Position_axis [mm] (Array of float): The position array correspondent to the interferogram, each point of the array should correspond in an ordered way to the correspondent point of the interferogram.
- Start_wavelength [nm] (Float): Start Wavelength in nanometers from which the spectrum is computed.
- End_wavelength [nm] (Float): End Wavelength in nanometers up to which the spectrum is computed.
- Samples (Integer): Number of spectral points to be computed.

OUTPUTS:

- Spectrum (1D array of float): output spectrum of the interferogram with the selected number of points.
- Wave (1D array of float): wavelength axis corresponding to the spectrum.
- Freq (1D array of float): spatial frequency axis corresponding to the spectrum.

2.8 Apodization

This function multiplies the interferogram (signal) by an asymmetric gaussian window composed by a left and a right tail gaussian with the same normalized apodization width. First it find the zero position of the interferogram, the absolute minimum value. It splits the position_axis in two halves with respect to the zero found. Then it calculates the two Gaussian

windows separately and afterwards it merges them together. At the end it multiply the input interferogram with the apodization window.

INPUTS:

- Apodization width (Float): the normalized standard deviation with respect to the number of points of the interferometric signal.
- Interferogram (Array of float): the input interferogram (signal) that has to be apodized.
- Position_axis [mm] (Array of float): the position array correspondent to the interferogram, each point of the array should correspond in an ordered way to the correspondent point of the interferogram.

OUTPUTS:

- Apodized_interferogram (Array of float): it is the input interferogram multiplied by the apodization window.

2.9 Close_connection

This function is responsible for closing the connection to the controller once you finish with measurements or experiments and you want to unplug the GEMINI from your computer. As for the initialization function, it exploits the *MCSControl.dll* library from Smaract. It takes as input:

- *system_index*: that is the handle to the opened system.
- *channel_index*: selects the channel of the selected system.

3. Main.py

This script is the main method of the program, the one has to be run. Before run it, in the first part, between the comments, you have to change the parameters accordingly to your measurements.

PARAMETERS:

- Start_wave [nm]: shortest wavelength of interest of the measured light.
- End_wave [nm]: longest wavelength of interest of the measured light.
- Resolution [nm]: resolution at the central wavelength of the defined spectral band.

- Sampling_factor: minimum number of points per period of the interferogram. This control sets automatically the number of steps of the scan.
- Spectral_points: number of spectral points of the output spectra.
- Apodization_width: standard deviation of the gaussian apodization window.

First it will initialize the controller and the positioner. Then it computes the necessary scan range and number of steps. Later it performs the acquisition of the interferogram, at the end it computes the Fourier transform to retrieve the spectrums.

After all the computational part, the plots of the spectrums, one with respect to wavelength and the other with respect to the spatial frequency, will be showed.

At the end, after the closing of the plot, there is the possibility to save the interferogram in a txt file, by choosing the file name, and automatically a folder with the date is created.

Configuration Files

The GEMINI interferometer is shipped already calibrated and configurated. In order to work properly, this example software needs three configuration files that must be present in the operational example working folder together with python script.

*_parameters_int.txt

This file contains the interferogram of a highly temporal coherent source such as HeNe laser sampled over the whole scan range of the GEMINI interferometer. The file is composed of two rows the first one contains the position axis and the second one the interferogram values at the correspondent positions. This file is used to calibrate the position axis of the interferometer with an extremely high precision. The file is used inside the *Get Calibrated position axis.vi* to calculate the position axis with interferometric accuracy, therefore it is always recommended to use this file. The position axis calibration procedure is simply an interferogram acquisition of a highly temporally coherent source such as HeNe laser over the whole scan range of the motor. The Acquired interferogram should be saved together with its correspondent position axis inside the "* parameters int.txt" file. The first row of the file must contain the position axis in millimeters and the second row must contain the correspondent interferogram values. If you need further information on how to perform the position axis calibration procedure, please contact us at info@nireos.com.

*_parameters_scale.txt

This file contains the scale parameter of the positioning system and it is used to find the physical zero of the interferometer. Warning: Do not modify this file! Changing the value contained in this file require invalidate the position axis calibration procedure performed in factory.

*_parameters_cal.txt

This file contains the conversion table from mm^{-1} to wavelength and it is used to calibrate the spectral axis. The user can add or modify any of the entry and the software will automatically take into account the change to compute the spectral axis.


NIREOS