GoldenEye™ Hyperspectral Imaging Camera

SDK

User's Guide

Version 1.0.2

Document No. 94-101301

BaySpec, Inc.

1101 McKay Drive, San Jose, CA 95131

Tel. (408) 512-5928 • Fax: (408) 512-5929

Web: www.bayspec.com

December 2023

Table of Contents:

1. Overview

BaySpec GoldenEye Software Development Kit (SDK) supplies the interface for the software developer to access the BaySpec GoldenEye™ Visible-Near-Infrared (NIR) GoldenEye hyperspectral imager. The SDK provides a set of functions that allow the user to configure and control the camera, acquire live images, and acquire and process stacks of up to 141 wavelength bands at full image sensor resolution. The callable Python functions in the SDK can be integrated into Python or LabVIEW programs (see Appendix III) as needed. A sample code running in Python is also included in this SDK. The library is compatible with Windows 10/11. The SDK is currently available for both Python 3.7 (SDK v1.0.1) and Python 3.10 (SDK v1.0.2).

Figure 1: GoldenEye™ hyperspectral imager with a standard f=8 mm (40° FOV) lens, in an ultra-compact package

2. System Requirements

2.1 Hardware

The minimum requirements for the computer are listed below:

OS: Microsoft Windows 10 / 11

CPU: Intel® Core™ i5 or above

RAM: 16 GB or higher (min 32 GB for high resolution option)

USB Port: USB 2.0 / 3.0

Display: 1280 × 1024 pixels or better

2.2 Software

Python version 3.7 or Python 3.10 is required.

Python 3.7: Required packages

- Ximea_api
- Threading
- Serial
- Numpy
- Crypto.Cipher
- Crypto.random
- Struct
- Lmfit
- Scipy.interpolate
- Os
- Sys
- Configparser

Python 3.10: Required packages

- Ximea_api
- Threading
- Serial
- Numpy
- pycryptodome
- Struct
- Lmfit
- Scipy.interpolate
- Os
- Sys
- Configparser

Note: The SDK includes a virtual environment file with the minimum required packages. It can be used with the ***conda*** package manager for the new environment creation. The command line instruction is "***conda env create -f sdk_env.yml***".

2.3 Required files

Provided (and necessary) scripts

- setup_acquisition.py
- Ximea API scripts and DLLs

3. Functions

3.1 **MyCamera()** – Connect and Identify Camera

Usage: **cam_id=MyCamera()**

Description: Establish connection to the camera. If succeed, assign the camera identification to the variable "***cam_id***". If connection failed, the error code 56 "No Devices Found" will be issued.

Parameters: None

Return value: camera identifier value if succeed, or "No Device Found" error if failed

Requirement: have imported MyCamera package ("***from setup_acquisition import MyCamera***")

3.2 **setup_cam_acquisition()** - Initialize Camera for Image Acquisition

Usage: **setup_acquisition.setup_cam_acquisition(cam_id, exposure, gain, high_resolution, acquisition length, wavelengths_in)**

Description: Initialize the system

Parameters: ***cam_id***: camera identifier

exposure: camera exposure time in µs (settable range: 7000-50,000)

gain: camera gain in dB (settable range: -3.5→+7.4)

high_resolution: a True/False Boolean indicating camera resolution mode. True: camera runs in full resolution: 1280 × 1024; False: camera runs in ¼ resolution: 640 × 512. Running the camera in the lower resolution mode will significantly increase the acquisition/processing speed.

Note: this parameter is only valid for newer version of GoldenEye with high resolution cameras, will be ignored if an older version of GoldenEye (pixel resolution: 648 × 488) is connected.

acquisition_length: number of frames to be acquired each time (frames are then processed into a single hyperspectral cube). The settable range is 50-200. The maximum value 200 (default) provides the highest spectral resolution for the cube, while the minimum value 50 requires the shortest image acquisition time (comes with lowest spectral resolution).

wavelengths_in (Optional): list of wavelengths of interest to be passed to the camera. The wavelengths have to be within 400 to 1100 nm range. If they are outside this range, the camera will default to 400-1000 nm interval with 5 nm steps. List is a numpy array. An example is: "wavelengths_in= np.linspace (400,1100,141)" Wavelength step interval is defined by the passed list. The resulting wavelengths will be slightly shifted depending on the exact camera calibration parameters. If no input is given for "wavelengths_in", the program will check for the Wavelengths.csv file to determine the interval. If the file is also missing, the default will be again 400-1100 nm.

Return value: None.

Requirements: (Optional) band definition file: "wavelengths.csv" in the App root folder, which determines the bands/wavelengths for the cube. If this file is missing, the default is 141 bands in the range of 400-1100 nm with an even interval of 5nm between the bands.

3.3 **acquire_cube() -** Acquire and process images

Usage: **Cube_information, band_images, wavelengths = setup_acquisition.**
acquire_cube (ext_trigger)

Description: Acquire images and obtain the hyperspectral cube data

Parameters: ***ext_trigger***: a True/False Boolean. True if camera will be externally triggered. If externally triggered, the camera will wait indefinitely for the external trigger to start the acquisition and then processing.

Return value: ***Cube_information***: all information needed about the hyperspectral cube. Used for most ENVI environment hyperspectral image loading. See Appendix I for an example. This information enables most ENVI-compatible software to automatically label and organize the hyperspectral data cube.

Band_images: hyperspectral cube of the size (#bands, samples, lines) where [#bands] is the number of bands/wavelengths (up to 141). [Samples × lines] is the image resolution of the band images. The pixel values of these images represent relative spectral intensity (raw sensor measurements). They are not radiometrically calibrated. To obtain radiometrically calibrated results, see Appendix II.

wavelengths: a list of wavelengths associated with the hyperspectral images. Although these are already contained within the ***Cube_information***, they are part of output for convenience and easy plotting.

Requirement: setup_acquire.py; Ximea API scripts and DLLs; wavelengths.csv.

4. Python sample program: "sdk_example.py"

A Python sample program is included in this SDK for demonstration of the usages of the functions, and to streamline user development. This sample program also includes a short example code section showing live image preview for alignment and focusing (done by calling the XIMEA API). Below screenshots show how to run this sample program.

Note: The SDK and its accompanying sample program are dependent on specific Python versions. The sample program included in SDK v1.0.1 is designed to run under Python 3.7, whereas the one packaged with SDK v1.0.2 should be executed using Python 3.10.

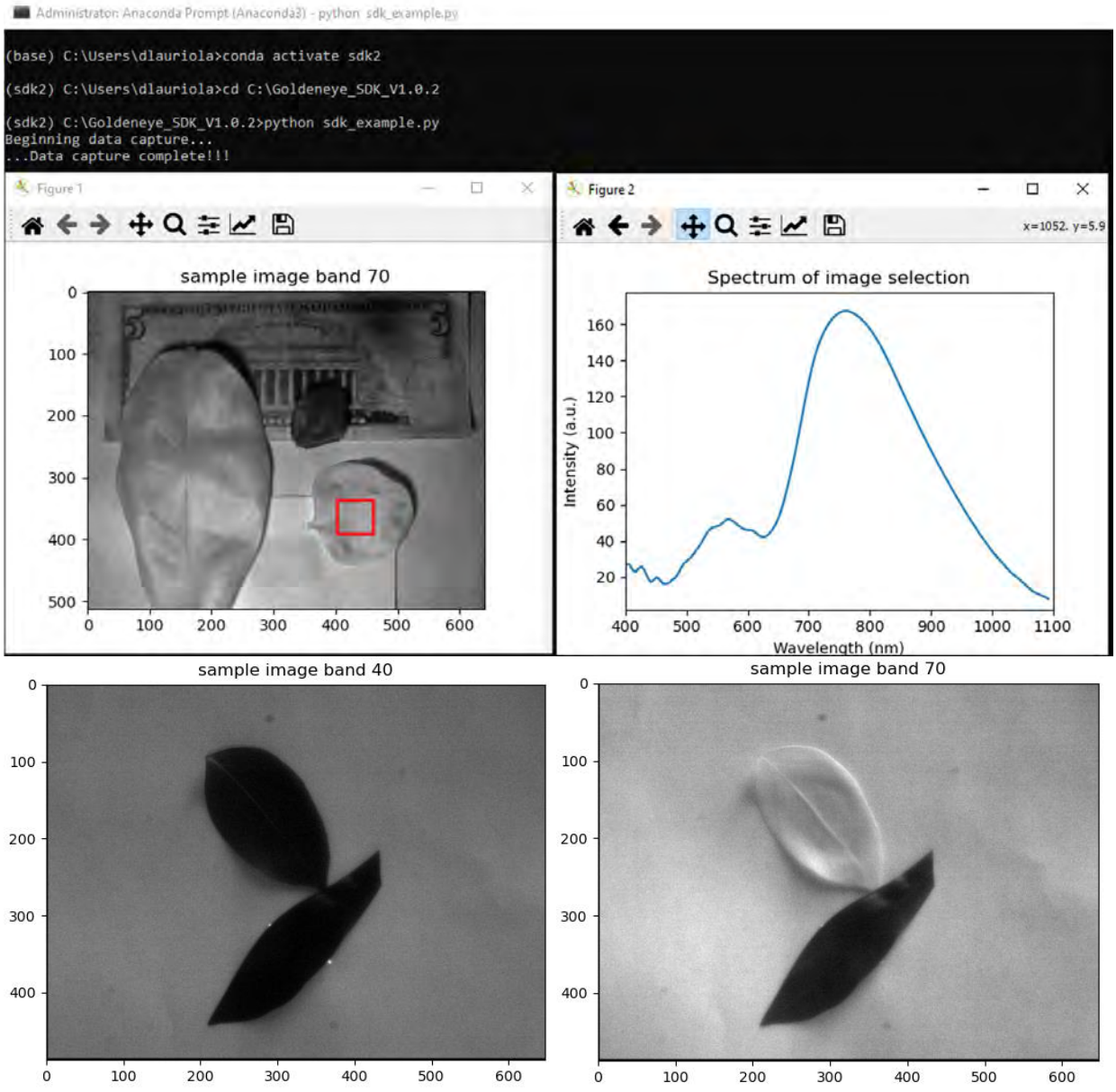

Figure 2: running the sample program

5. Files in the SDK package

Below is a list files included in the SDK package:

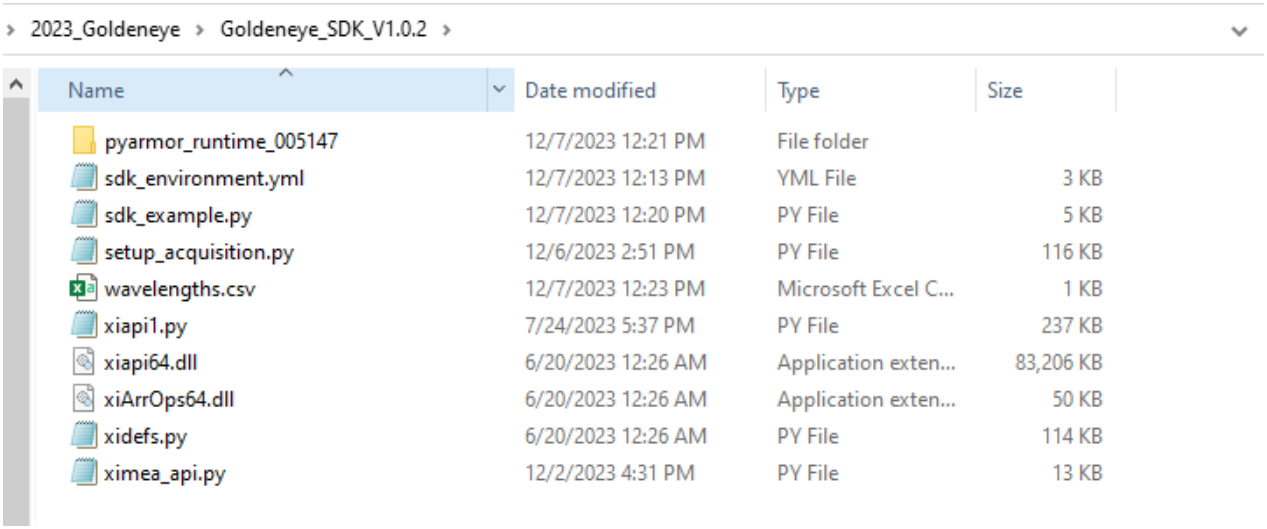

Figure 3: List of included files for the SDK

Note: ***Wavelengths.csv*** contains a list of the wavelengths of interest and can be varied to focus on specific wavelengths and/or expanded to see a broader range of wavelengths. By default, the imager is setup to measure over the full spectral range of 400 - 1100nm. It will generate 141 Hyperspectral band images in a 5nm step. If desired, users can define their own wavelength bands in any integer steps. For example, if a user only wants to see some specific wavelengths, or wants smaller band steps in shorter wavelength and larger band steps in longer wavelengths (to match the wavelength resolution), this can be done by modifying the file "***wavelengths.csv***" to specify the desired wavelengths and step sizes (**must be within the range of 400nm to 1100nm, be listed monotonously rising, the minimum step is 1nm, must be saved in "CSV (Comma delimited) (.csv)" format**). The software will read this file and generate Hyperspectral band images accordingly.

Alternatively, the user can pass the wavelengths as a list within the python function, instead of having the Wavelengths.csv file.

Appendix I: ENVI Header Information Example

ENVI
bands = 141
samples = 640
lines = 512
interleave = bsq
data type = 12
byte order = 0
wavelength units = nm
wavelength = { 400.93, 405.96, 410.98, 416.00, 421.03, 426.05, 431.07, 436.08, 441.10, 446.12, 451.13, 456.15, 461.16, 466.17, 471.18, 476.19, 481.20, 486.20, 491.21, 496.21, 501.21, 506.21, 511.22, 516.21, 521.21, 526.21, 531.20, 536.20, 541.19, 546.18, 551.17, 556.16, 561.15, 566.14, 571.12, 576.11, 581.09, 586.07, 591.05, 596.03, 601.01, 605.99, 610.97, 615.94, 620.91, 625.89, 630.86, 635.83, 640.79, 645.76, 650.73, 655.69, 660.65, 665.62, 670.58, 675.54, 680.50, 685.45, 690.41, 695.36, 700.32, 705.27, 710.22, 715.17, 720.12, 725.06, 730.01, 734.95, 739.90, 744.84, 749.78, 754.72, 759.65, 764.59, 769.53, 774.46, 779.39, 784.32, 789.25, 794.18, 799.11, 804.04, 808.96, 813.88, 818.81, 823.73, 828.65, 833.56, 838.48, 843.40, 848.31, 853.22, 858.14, 863.05, 867.95, 872.86, 877.77, 882.67, 887.58, 892.48, 897.38, 902.28, 907.18, 912.07, 916.97, 921.86, 926.76, 931.65, 936.54, 941.43, 946.31, 951.20, 956.08, 960.97, 965.85, 970.73, 975.61, 980.49, 985.36, 990.24, 995.11, 999.98, 1004.86, 1009.72, 1014.59, 1019.46, 1024.32, 1029.19, 1034.05, 1038.91, 1043.77, 1048.63, 1053.49, 1058.34, 1063.20, 1068.05, 1072.90, 1077.75, 1082.60, 1087.45, 1092.29}
fwhm =
{6.84,7.14,7.44,7.74,8.05,8.36,8.67,8.98,9.30,9.62,9.95,10.28,10.61,10.94,11.28,11.62,11.97,12.31,12.66,13.02,13.37,13.73,14.10,14.46,14.83,15.20,15.58,15.96,16.34,16.72,17.11,17.50,17.90,18.29,18.69,19.10,19.50,19.91,20.32,20.74,21.16,21.58,22.00,22.43,22.86,23.30,23.74,24.18,24.62, 25.06,25.51,25.97,26.42,26.88,27.34,27.81,28.27,28.75,29.22,29.70,30.18,30.66,31.14,31.63,32.13,32.62,33.12,33.62,34.12,34.63,35.14,35.65,36.17,36.69,37.21,37.74,38.26,38.79,39.33,39.86,40.40,40.95,41.49,42.04,42.59,43.15,43.71,44.27,44.83,45.40,45.97,46.54,47.12,47.69,48.28,48.86, 49.45,50.04,50.63,51.23,51.82,52.43,53.03,53.64,54.25,54.86,55.48,56.10,56.72,57.35,57.97,58.61,59.24,59.88,60.52,61.16,61.80,62.45,63.10,63.76,64.41,65.07,65.74,66.40,67.07,67.74,68.42,69.09,69.77,70.45,71.14,71.83,72.52,73.21,73.91,74.61,75.31,76.02,76.73,77.44,78.15}

Appendix II: Calibrated Spectral Intensities

A standard practice for many hyperspectral imaging applications, and in particular for wavelength based image classification, is to measure reflectance. Since the reflectance is dependent upon the knowledge of the ambient/impinging light, it has to be calibrated every time the light/environment changes. In many applications this is done by taking a white reference image, with an object which has a constant reflectance ~1 across the spectrum, and then factor the raw band images to it to generate the reflectance band images.

Standard steps which can be taken to obtain reflectance are:

1) Acquire a hyperspectral image of a white reference, with uniform illumination, using: ***cube_info, band_images, wl = setup_acquisition.acquire_cube (ext_trigger)***,
2) Save the *"band_images"* to a variable*: "white_reference" for example,*
3) Acquire successive hyperspectral images with the same camera settings: ***cube_info, band_images, wl = setup_acquisition. acquire_cube (ext_trigger)***,
4) For every new acquisition of *"band_images",* divide the result by the *"white_reference"* cube to obtain the absolute reflectance,
5) Take a new "white reference" image if the light conditions change.

Note: During the division process care must be taken to avoid large, physically improbable reflectance values; so, it is advisable to check for any zero-data in the white_reference image and to ensure that the acquired white_reference intensities are ~5-10 times above the noise floor threshold.

Appendix III: Run Python Code in LabView Environment

It is possible to run Python code in LabVIEW environment. There are two ways to do this:

- Using the **Python Node**: The Python Node is a LabVIEW VI that allows you to call Python code from within a LabVIEW program. The Python Node can be used to execute Python scripts, pass data to and from Python, and access Python libraries.
- Using the **System Exec.vi**: The System Exec.vi is a LabVIEW VI that allows you to execute a system-level command line. This VI can be used to launch a Python interpreter and execute a Python script from the command line.

Here are the steps on how to run Python code in LabVIEW environment using the Python Node:

1. Install Python on your computer.
2. Install the Python Node in LabVIEW.
3. Create a new LabVIEW project.
4. Add the Python Node to your project.
5. Configure the Python Node with the path to your Python interpreter and the name of your Python script.
6. Run your LabVIEW program.

Here are the steps on how to run Python code in LabVIEW environment using the System Exec.vi:

1. Install Python on your computer.
2. Create a new LabVIEW project.
3. Add the System Exec.vi to your project.
4. Configure the System Exec.vi with the path to your Python interpreter and the command line to execute your Python script.
5. Run your LabVIEW program.

For further instructions please consult NI knowledge website:
https://knowledge.ni.com/KnowledgeArticleDetails?id=kA03q000000x4PSCAY&l=en-US
You may also be able to find many coding tutorials and examples in YouTube channels.