NAPVMI Experiment Software Documentation

Lea Hohmann

June 2020

Contents

1 List of modules, classes and class methods

1.1 Main (napvmi_main.py)

Main module, upon executing, it initializes a class instance of Root()

1.1.1 class Root(tkinter.Tk)

Main GUI class, inherits from the tkinter class Tk()

- Root.bncconnect()
  Tries to establish a serial port connection to the delay generator (if connection fails, gives error message). Calls Root.bncinit() after successful connection and changes the connection status of Root() class.

- Root.bncinit()
  Initializes BNC GUI by creating an instance of the bncmodule.DelayApp() class

- Root.camerconnect()
  Tries to establish a connection to the camera by calling Root.cameraident() on each available device (if connection fails, gives error message). Calls Root.camerainit() after successful connection and changes connection status of Root() class

- Root.cameraident()
  Checks serial number of camera device passed to the function and compares to Root() serial number. If the correct camera is found, it passes it back to Root.cameraconnect()

- Root.camerinit()
  Initializes camera GUI by creating an instance of cameramodule.CameraApp()

- Root.startdelayint()
  Callable by a GUI button if camera and BNC are connected. Calls Root.checkdelays() and starts a Delay integration experiment by creating an instance of delayintegration.IntegrationGUI() (GUI opens in new window).

- Root.startkineticseries()
  Callable by a GUI button if camera and BNC are connected. Calls Root.checkdelays() and starts a Kinetic series experiment by creating an instance of kineticseries.SeriesGUI() (GUI opens in new window).

- Root.checkdelays()
  Checks all delays apart from Channel B (Laser Q-switching, main scanning channel) and stores the values in a list that is used in the experiment parameter file.

- Root.quitgui()
  Called when quitting the GUI. Calls the quit functions of the modules and releases the camera properly before closing event.

1.2 Delay generator (bncmodule.py)

1.2.1 class DelayApp(tkinter.Frame)

Class for the delay generator GUI, inherits from tkinter.Frame()

- DelayApp.intialquery()
  Sends an initial query to the delay generator to determine whether it is running; changes the ON/OFF state of the delay generator object

- DelayApp.guiinit()
  Initializes all widgets of the delay generator GUI

- DelayApp.setchannel()
  Called by GUI button. Destroys the current Channel() instance (if applicable) and establishes a new one based on the channel name chosen by the user

- DelayApp.runtriggering()
  Called by GUI button. Starts the triggering on the delay generator, checks that it is running and changes the ON/OFF button.

- DelayApp.stoptriggering()
  Called by GUI button. Stops the triggering on the delay generator, checks that it has stopped running and changes the ON/OFF button.

- DelayApp.quitapp()
  Called when quitting the GUI. Stops the triggering before the closing event.

1.2.2 class Channel(tkinter.Frame)

Class for controlling the delay channels. Inherits from tkinter.Frame()

- Channel.bncinit()
  Queries the delay generator for the channel delay and changes the channel object delay variable.

- Channel.guiinit()
  Initializes the channel GUI

- Channel.guiupdate()
  Called when making changes to the delay or querying the delay. Updates the channel GUI with current value of delay variable.

- Channel.plus()
  Defines a positive increment and calls Channel.changedelay()

- Channel.minus()
  Defines a negative increment and calls Channel.changedelay()

- Channel.changedelay()
  Changes delay using the user defined increment, step size (1, 10 or 100) and time range (ms,us,ns)

1.3 Camera (cameramodule.py)

1.3.1 class CameraApp(tkinter.Frame)

Class for the camera GUI, inherits from tkinter.Frame()

- CameraApp.guiinit()
  Initializes the widgets of the camera GUI
- CameraApp.camerasetup()
  Accesses the camera nodes for the settings (exposure time, gain) and sets the initial values that are displayed in the GUI, sets the trigger mode to the external trigger.
- CameraApp.exposuretime()
  Sets the cameras exposure time according to the value obtained from the GUI slider.
- CameraApp.gain()
  Sets the cameras gain according to the value obtained from the GUI slider.
- CameraApp.settrigger()
  Called from GUI, starts the triggered acquisition.
- CameraApp.stoptrigger()
  Called from GUI, stops the triggered acquisition (returns to automatic acquisition)
- CameraApp.setup_acquisition()
  Called by acquisition functions before starting an acquisition to set up the camera for acquisition (accessing buffer handling and acquisition mode).
- CameraApp.start_liveacquisition()
  Called by acquisition functions to start a continuous acquisition of frames on the camera.
- CameraApp.start_singleframelive()
  Callable from the GUI to start a single frame live display. Starts a live acquisition by calling on CameraApp.start_liveacquisition() and disables the other acquisition functions in the GUI. Calls on CameraApp.imageloop() to grab and display frames continuously.
- CameraApp.start_multiframelive()
  Callable from the GUI to start a multiframe (rolling average) live display. Starts a live acquisition by calling on CameraApp.start_liveacquisition() and disables the other acquisition functions in the GUI. Calls on CameraApp.multiframeloop() to grab frames and display summed images continuously.
- CameraApp.imageloop()
  Grabs next frame from camera (in triggered acquisition, if not trigger occurs after 5 seconds, displays an error message), initiates display of the frame by calling on CameraApp.displayimage(), the intensity stats and the total intensity by calling CameraApp.integrateimage(). Loops until running=False (stop button pressed)

- CameraApp.multiframeloop()
  Calls on CameraApp.getmultiframeimage() to acquire a summed image (using user-set number of frames). Initiates display of the image by calling on CameraApp.displayimage(), the intensity stats and the total intensity by calling CameraApp.integrateimage(). Loops until running=False (stop button pressed)

- CameraApp.stop_liveacquisition()
  Called from GUI (stop button), stops a live acquisition by setting running=False.

- CameraApp.acquireimage()
  Called from GUI to acquire single image with user-set number of frames. Calls CameraApp.capturemultiframe() to acquire the image, if successful calls CameraApp.displayimage() and CameraApp.integrateimage()

- CameraApp.acquirexslice()
  Calls CameraApp.capturemultiframe() to acquire an image. Sums image over y to get the integrated intensity vector and creates an x vector with the user defined values. Calls CameraApp.displayslice()

- CameraApp.acquireyslice()
  Calls CameraApp.capturemultiframe() to acquire an image. Sums image over x to get the integrated intensity vector and creates a y vector with the user defined values. Calls CameraApp.displayslice()

- CameraApp.capturemultiframe()
  Sets up an acquisition with a user-set number of frames. Calls CamerApp.getmultiframeimage() to acquire the image and only keeps the user-set xy area (partial image)

- CameraApp.getmultiframeimage()
  Acquires a set number of frames from the camera and sums them into a single image.

- CameraApp.displayimage()
  Displays the current image, creates and displays an image histogram, and x and y lines at set x and y limits. Checks for low or high signal and displays warnings if appropriate.

- CameraApp.displayslice()
  Displays an x or y slice.

- CameraApp.integrateimage()
  Integrates the entire image to get the total intensity value.

- CameraApp.save_asarray()
  Callable from GUI. Saves the current image array as numpy array file.

- CameraApp.save_asimage()
  Callable from GUI. Saves the current image array as image file.

- CameraApp.save_slice()
  Callable from GUI. Saves the current slice (intensity vector and x or y vector).

- CameraApp.saveparameterfile()
  Callable frim GUI. Saves the parameters into a text file.

- CameraApp.loadparameterfile()
  Callable from GUI. Loads a parameter file and sets the camera parameters after it.

- CameraApp.quit_cameraapp()
  Called when quitting the GUI. Releases the camera object before the closing event.

1.4 Delay integration (delayintegration.py)

1.4.1 IntegrationGui(tkinter.Toplevel)

Class that opens the window to set up and run a delay integration experiment. Inherits from tkinter.Toplevel(). Upon creating an instance of this class, the main GUI is disabled until the integration GUI is destroyed.

- IntegrationGui.guiinit()
  Initializes the widgets of the delay integration GUI.

- IntegrationGui.startacquisition()
  Acquires the experiment parameters (no of frames, delay range,...) from GUI. Runs through a loop of all delays in the range and sets each delay, then calls IntegrationGui.imageloop() to generate image, adds image to the sum image and displays the current sum and last image. After the loop, saves the image and the parameter file with the filename input by user.

- IntegrationGui.imageloop()
  Acquires the set number of frame and sums them to a single image.

- IntegrationGui.closegui()
  Called when closing the GUI. Releases the camera and restores the main GUI before the closing event.

1.5 Kinetic series (kineticseries.py)

1.5.1 SeriesGui(tkinter.Toplevel)

Class that opens the window to set up and run a kinetic series experiment. Inherits from tkinter.Toplevel(). Upon creating an instance of this class, the main GUI is disabled until the Series GUI is destroyed.

- SeriesGui.guiinit()
  Initializes the widgets of the kinetic series GUI.

- SeriesGui.startacquisition()
  Acquires the experiment parameters (no of frames, delay range,...) from GUI. Runs through a loop of all delays in the range and sets each delay, then calls IntegrationGui.imageloop() to generate image, appends image to the image series and displays the current image, adds the total intensity to the intensity

vector and displays intensity vs time (updated after each image). After the loop, saves the series of images and the parameter file with the filename input by user.

- SeriesGui.imageloop()
  Acquires the set number of frame and sums them to a single image.

- SeriesGui.closegui()
  Called when closing the GUI. Releases the camera and restores the main GUI before the closing event.

2 Hardware requirements

Camera:
Should be usable with Spinnaker software/PySpin (written for FLIR BFLY-U3)

Delay generator:
Connected via COM / USB virtual COM port. Needs to be able to receive basic SCPI commands. Written for BNC model 577

Connections:
USB3 (Camera); USB (Delay generator)

3 Software requirements

Windows:
Code was written and tested in Windows 10, no other OS were tested. Earlier versions should work as long as required python version is supported. Some windows-specific python is used and might have to be changed to work on other OS

Python:
Python 3.4 or higher. Code was written and tested in python 3.7

Python modules:
numpy
tkinter
pySerial
matplotlib
PySpin (Spinnaker Python wrapper): https://meta.box.lenovo.com/v/link/view/a1995795ffba47dbbe45771477319cc3 (choose the one for the correct python version, e.g. for 64 bit python 3.7.: cp37[...]win_amd64.zip. To check your python version, run the python –version command. To see whether you are running 32 or 64 bit python, start python, import platform and call the platform.architecture() function).

Camera drivers:
For FLIR cameras: Download Spinnaker from FLIR: https://www.flir.com/support-center/iis/machine-vision/downloads/spinnaker-sdk-flycapture-and-firmware-download/

BNC drivers:
For BNC delay generators connected via USB, download FTDI Virtual COM port drivers: https://www.ftdichip.com/Drivers/VCP.htm

4 Setup and implementing hardware changes

Camera:
Install drivers (see above) and connect camera via USB3
In case camera is changed: The code recognizes the camera by serial number. If a new camera is used, the serial number needs to be changed in the napvmi_main.py module in the Root.__init__() function and in the error message in the root.cameraconnect() function:

```
class Root(tk.TK):

    def __init__(self):
        self.serialnumber = "NEWSERIALNUMBER"

    def cameraconnect(self):
         if self.camera == "":
            messagebox.showerror("Error", "Camera
               NEWSERIALNUMBER not found. Connect the camera and
                retry.")
```

BNC:
Install VCP drivers (see above) and connect via USB
Check the name/number of the virtual COM port under serial ports in the Windows device manager. If it is not COM5, the the name (NAMEOFCOMPORT) has to be edited in the napvmi_main.py module in the root.bncconnect() function:

```
class Root(tk.TK):

    def bncconnect(self):
        try:
            self.bnc = serial.Serial("NAMEOFCOMPORT", baudrate
               =115200, bytesize=8, parity="N", stopbits=1,
               timeout=1)
```

Check that the baudrate on the delay generator is 115200. If this baudrate is not available, change it in the root.bncconnect function (see code snippet above).
If the delay generator is replaced by a different model, ensure that it can understand SCPI commands and that the commands used match those of the Model 577 (otherwise commands need to be edited)

Channel setup:
Channel B (channel number 2) is used in the code as the main scanning channel

for delay integration and kinetic series experiments. This should therefore be the channel triggering the laser q-switching. The other channels can be chosen arbitrarily. The channel reference (channel which represents the zero for the current channel's delay setting) is set manually on the delay generator and should be set once for every channel for the entire setup (as the information is not included in the parameter files).

Experiment directory:
The standard directory for saving and loading files is "C:/", if a different location is to be used as the standard directory, changes have to be made in cameramodule.py, kineticseries.py and delayintegration.py to all occurrences of the following code:

```
(intialdir="C:/" ...)
```