NAPVMI Experiment Software Documentation

Lea Hohmann

July 13, 2022

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.laserconnect()
  Checks for available laser device (over serial port) and tries to establish connection to laser. After successful connection, calls Root.laserinit() and changes connection status of Root class.

- Root.laserinit()
  Initializes laser GUI by creating an instance of lasermodule.LaserApp()

- Root.startdelayint()
  Callable by a GUI button if camera and BNC are connected. Calls Root.checkdelays() and starts a Delay integration experiment by calling Root.newwindow() to create a toplevel and then creating an instance of delayintegration.IntegrationGUI().

- Root.startkineticseries()
  Callable by a GUI button if camera and BNC are connected. Calls Root.checkdelays() and starts a Kinetic series experiment by calling Root.newwindow(), then creating an instance of kineticseries.SeriesGUI().

- Root.startwavelengthseries()
  Callable by GUI button if camera, BNC and laser are connected. Calls Root.checkdelays() and starts a wavelength series experiment by calling Root.newwindow() to create a toplevel, then creating an instance of wavelengthseries.WavelengthGui()

- 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.newwindow()
  Creates a tkinter Toplevel window, disabled the main GUI functions and the series buttons and specifies the closing event function (Root.quitseries())

- Root.quitseries()
  Called when a series toplevel is closed. Releases the series camera instance and restores the main GUI, as well as updating the BNC and laser by query before closing the series toplevel.

- 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.savedelayfile()
  Called by GUI button. Queries delay generator for current delay settings on all channels, then asks the user for a filename to save them under.

- DelayApp.loaddelayfile()
  Called by GUI button. Asks the user to choose an existing file with delay values, then changes all delays to the ones specified in the file. If there is currently a channel open on the GUI, it will update it with the current delay afterwards.

- 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.autorange()
  Sets the image zoom area to default (entire image)

- 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.setexposure()
  Called by GUI button. Retrieves the value from the exposure time manual entry box and updates the exposure slider with it.

- CameraApp.exposuretime()
  Sets the cameras exposure time according to the value obtained from the GUI slider.

- CameraApp.setgain()
  Called by GUI button. Retrieves the value from the gain manual entry box and updates the gain slider with it.

- 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 internal camera triggering)

- 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_multiframelive()
  Callable from the GUI to start a rolling average live display with user-set number of frames. 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.multiframeloop()
  Creates empty image objects and creates a separate thread for the acquisition loop. Calls on CameraApp.getmultiframeimage() within the thread to acquire a summed image (using user-set number of frames). Calls multiframeloop2 with user set framecount to track and display acquisition.

- CameraApp.multiframeloop2()
  framecount
  While there is no user interrupt, periodically checks whether framecount has been reached. When framecount is reached, initiates display of image by calling CameraApp.displayimage() and CameraApp.integrateimage(). If live acquisition is not interrupted, calls CameraApp.multiframeloop() to start a new round of acquisition.

- 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. Sets type of image to image, then calls CameraApp.capturemultiframe() to acquire the image.

- CameraApp.acquirexslice()
Called from GUI to acquire single x slice. Sets type of image to x slice, then calls CameraApp.capturemultiframe() to acquire an image.

- CameraApp.acquireyslice()
Called from GUI to acquire single y slice. Sets type of image to y slice, then calls CameraApp.capturemultiframe() to acquire an image.

- CameraApp.capturemultiframe()
Sets up an acquisition with a user-set number of frames and creates empty image objects. Starts a separate thread for acquisition, calls CameraApp.getmultiframeimage() within thread to execute acquisition. Calls CameraApp.capturemultiframe2() with user set framecount to track and display acquisition.

- CameraApp.capturemultiframe2()
framecount
While there is no user interrupt, periodically checks whether framecount is reached. If framecount is a multiple of 20, calls CameraApp.quickdisplay() to show last 20 frames. When framecount is reached, treats the data according to image type (image, x or y slice). Displays the result by calling CameraApp.displayimage() and CameraApp.integrateimage() or CameraApp.displayslice(). Finishes acquisition.

- CameraApp.getmultiframeimage()
framecount
While running, acquires a number of frames from the camera set by framecount and sums them into a single image.

- CameraApp.displayimage()
framecount
Displays the current image or the set zoom region, creates and displays a frame histogram using framecount. Checks for low or high signal and displays warnings if appropriate.

- CameraApp.quickdisplay()
displaydata
Displays the data passed to it in displaydata and creates a histogram per frame with the framecount of 20. Checks for low or high signal and dislays 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. Calls CameraApp.parametersaver() to save parameters in separate txt file.

- CameraApp.save_asimage()
  Callable from GUI. Saves the current image array as image file. Calls CameraApp.parametersaver() to save parameters in separate txt file.

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

- CameraApp.saveparameterfile()
  Callable from GUI. Asks the user for filename input and calls CameraApp.parametersaver() to save parameter file.

- CameraApp.parametersaver()
  filename
  Saves parameters in filename passed to the function.

- 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 Laser (lasermodule.py)

1.4.1 LaserApp(tkinter.Frame)

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

- LaserApp.initialquery() *Sends an initial query to the laser to update the current wavelength, changes the LaserApp.wavelength attribute.*

- LaserApp.guiinit() *Initializes all widgets of the laser GUI*

- LaserApp.plus() *Called from GUI incrementing (plus) button. Defines a positive increment (+1) and calls LaserApp.changewavelength()*

- LaserApp,minus() *Called from GUI incrementing (minus) button. Defines a negative increment (-1) and calls LaserApp.changewavelength()*

- LaserApp.changewavelength() *Changes the wavelength according to the increment (plus/minus) and step size defined by user.*

1.5 Delay integration (delayintegration.py)

1.5.1 IntegrationGui(tkinter.Frame)

Class that contains the GUI to set up and run a delay integration experiment. Inherits from tkinter.Frame().

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

- IntegrationGui.startacquisition()
  Acquires the experiment parameters (no of frames, delay range,...) from GUI. Asks for filename, sets camera parameters for acquisition and delay range vector to be iterated through and sets the running parameter. Calls delayloop() function with the first index (0).

- IntegrationGui.delayloop()
  index
  Sets the delay corresponding to current index, then calls IntegrationGui.imageloop() to generate image, adds image to the sum image and displays the current sum and last image. Increments the index and starts over if running parameter is true and the end of the delay range is not reached. Otherwise calls savedata()

- IntegrationGui.savedata()
  Saves the image and the parameter file with the filename input by user. Restores the GUI start button so that a second measurement can be run.

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

- IntegrationGui.experimentsaver()
  Saves the current experiment settings in a user defined file.

- IntegrationGui.experimentloader()
  Loads experiment settings from user defined file and updates GUI

- IntegrationGui.userinterrupt()
  Callable by stop button. Sets the running parameter as False, causing delayloop() to stop after finishing the current acquisition and instead save the partial experiment.

1.6 Kinetic series (kineticseries.py)

1.6.1 SeriesGui(tkinter.Frame)

Class that contains the GUI to set up and run a kinetic series experiment. Inherits from tkinter.Frame().

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

- SeriesGui.addrange()
  instance
  Called by GUI button. Adds a second or third delay range with separate increment to the GUI options.

- SeriesGui.removerange()
  rangenumber
  Called by GUI button. Removes a delay range from the GUI options.

- SeriesGui.startacquisition()
  Acquires the experiment parameters (no of frames, delay range,...) from GUI. Sets up camera for acquisition. Calls delayloop() to iterate through the delays.

- SeriesGui.delayloop()
  index
  Sets the delay corresponding to current index, then calls SeriesGui.imageloop() to generate image, adds the total intensity to the intensity vector and displays intensity vs time (updated after each image). Keeps incrementing the delayvector index and repeating itself if the delay has not reached the end value and if running parameter is True. Otherwise calls savedata()

- SeriesGui.savedata()
  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.experimentsaver()
  Saves the current experiment settings in a user defined file.

- SeriesGui.experimentloader()
  Loads experiment settings from user defined file and updates GUI

- SeriesGui.userinterrupt()
  Sets the running parameter to False, leading to termination of acquisition after the current image is completed.

1.7 Wavelength series (wavelengthseries.py)

1.7.1 WavelengthGui(tkinter.Frame)

Class that contains the GUI to set up and run a wavelength series experiment. Inherits from tkinter.Frame().

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

- WavelengthGui.evalentry()
  Called upon starting the acquisition, acquires the experimental parameters from GUI and checks if entries are within range, if yes calls startacquisition()

- WavelengthGui.startacquisition() *Sets up the camera for acquisition. Sets the starting wavelength and increment and calls WavelengthGui.imageloop() to generate the first image, adds the image and total intensity to the respective matrices and dispays intensity vs time After the loop, saves the series of images and the parameter file with the filename input by user.*

- WavelengthGui.wavelengthloop() *Increments the wavelength and then calls imageloop() to acquire an image, adds the image and intensities to the result matrices and updates the display. Keeps repeating itself if running parameter is True and end wavelength has not been reached. Otherwise calls savedata()*

- WavelengthGui.savedata() *Saves the data and parameters under the specified filename.*

- WavelengthGui.imageloop() *Acquires the set number of frames and sums them into a single image.*

- WavelengthGui.userinterrupt() *Sets the running parameter to false in order to interrupt acquisition after completing the current image.*

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

Laser:
Needs to be able to receive commands over serial. Written for Radiant dyes NarrowScan laser (commands are specific)

Connections:
USB3 (Camera); USB (Delay generator); RS232 or USB for virtual COM (Laser)

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 molecular beam (it is set as negative delays relative to 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).

Laser:
Connect via COM or virtual COM port. Check the COM port number on the computer (under serial ports in Windows device manager). If it is not COM1, the name (NAMEOFCOMPORT) has to be edited in the napvmi_main.py module in Root.laserconnect():

```
class Root(tk.TK):

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

The baudrate for the laser should be 115200.

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:/" ...)
```