NAPVMI Experiment Software Documentation

Lea Hohmann

November 30, 2024

Contents

1 List of modules, classes and class methods **2**
1.1 Main (napvmi_main.py) . . . 2
1.1.1 class Root(tkinter.Tk) . . . 2
1.2 Delay generator control (bncmodule.py) . . . 3
1.2.1 class DelayApp(tkinter.Frame) . . . 3
1.2.2 class Channel(tkinter.Frame) . . . 4
1.3 Camera control (cameramodule.py) . . . 4
1.3.1 class CameraApp(tkinter.Frame) . . . 4
1.4 Laser control (lasermodule.py) . . . 7
1.4.1 class LaserApp(tkinter.Frame) . . . 7
1.5 Delay integration (delayintegration.py) . . . 8
1.5.1 class IntegrationGui(tkinter.Frame) . . . 8
1.6 Kinetic series (kineticseries.py) . . . 9
1.6.1 class SeriesGui(tkinter.Frame) . . . 9
1.7 Wavelength series (wavelengthseries.py) . . . 10
1.7.1 class WavelengthGui(tkinter.Frame) . . . 10
1.8 Motor controller (motormodule.py) - Separate GUI . . . 11
1.8.1 class MotorApp(tkinter.Tk) . . . 11
1.8.2 class Motor() . . . 13

2 Hardware and software requirements **15**
2.1 Hardware . . . 15
2.2 Software . . . 15

3 Installation and implementing hardware changes **17**
3.1 Camera . . . 17
3.2 BNC delay generator . . . 17
3.3 Laser . . . 18
3.4 Motors . . . 18

4 Software repository **19**
4.1 Making changes to the software . . . 19
4.2 Creating a local copy . . . 19
4.3 Restoring a previous version of the software . . . 20

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 control (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()
  Starts a separate thread from which to run Channel.update()

- Channel.guiinit()
  Initializes the channel GUI

- Channel.update()
  Continuously running in separate thread from mainloop, queries current delay and updates the GUI label and the 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) while pausing Channel.update()

1.3 Camera control (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_acquisition()
  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()
  start,end
  Displays an x or y slice in range start-end.

- CameraApp.integrateimage()
  Integrates the entire image (or specified display zoom region) to get the total

intensity value and displays it below the image. If integration of zoomed area is active, integrates specified area and displays value in the top window.

- CameraApp.integratezoom()
  Called from GUI during live acquisition to display integration of zoom region. Opens new top window for the display of integrated intensity.

- CameraApp.quitzoomint()
  Called when quitting zoom integration window to quit window and restore initial GUI settings.

- 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 control (lasermodule.py)

1.4.1 class 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.update()
  Started during init, runs in a separate thread, continuously queries laser for wavelength and motor settings and displays them in 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()
  direction
  Changes the wavelength according to the user defined increment, negative or positive depending on direction parameter.

- LaserApp.fcuplus()
  number
  Called from GUI button with FCU number parameter, calls the LaserApp.stepfcu() function with positive increment.

- LaserApp.fcuminus()
  number
  Called from GUI button with FCU number parameter, calls the LaserApp.stepfcu() function with negative increment.

- LaserApp.stepfcu()
  number, direction
  Steps FCU motor (defined by number parameter) by user set increment in positive or negative direction.

- LaserApp.shutdown()
  Sends the shutdown command to laser and disables GUI.

1.5 Delay integration (delayintegration.py)

1.5.1 class 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 class 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 an additional delay range with separate increment to the GUI options.
- SeriesGui.removerange()
  instance
  Called by GUI button. Removes a delay range from the GUI options.
- SeriesGui.connectfieldmax()
  Called by GUI button. Establishes connection to FieldMaxII using the pyFieldMaxII python wrapper. Gives an error message if FieldMax is not connected. If connected, laser energy measurement will be taken for each image and saved in separate file.
- 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 class 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.addrange()
  instance
  Adds a wavelength range with separate increment to the Gui (instance = range number).

- WavelengthGui.removerange()
  instance
  Removes a wavelength range (instance = range number) from the Gui.

- WavelengthGui.connectfieldmax()
  Called by GUI button. Establishes connection to FieldMaxII using the pyFieldMaxII python wrapper. Gives an error message if FieldMax is not connected. If connected, laser energy measurement will be taken for each image and saved in separate file.

- 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.wrongentry()
  Called in the case of a detected wrong entry, displays error message.

- 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()
  Started in separate thread. Increments the wavelength and then calls imageloop() to acquire an image. Keeps repeating itself if running parameter is True and end wavelength has not been reached. Otherwise calls savedata().

- WavelengthGui.datahandling()
  Called after every loop iteration, stores image data according to wavelength (queried from laser).

- WavelengthGui.displayloop()
  Displays current data in GUI (runs in tkinter main loop).

- 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.experimentsaver()
  Called by GUI button, saves current experiment settings in user defined file.

- WavelengthGui.experimentloader()
  Called by GUI button, loads experiment settings from file chosen by user.

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

1.8 Motor controller (motormodule.py) - Separate GUI

1.8.1 class MotorApp(tkinter.Tk)

Class for the motor controller GUI, inherits from tkinter.Tk()

- MotorApp.portconnect()
  Scans through the existing COM ports (exception: COM4 and COM5, which are used for other parts of the setup) up to COM9 and looks for the serial numbers of all four motors. Whenever it finds a right serial number, opens connection to that motor and adds it to the connected motor list. If all motors succesfully connected, call MotorApp.loadcurrent(). If not, displays error dialogue and asks for retry.

- MotorApp.steptomm()
  stepvalue
  Translates step value passed to the function into distance in mm.

- MotorApp.mmtostep()
  mminput
  Translates mm input passed to the function into a value in motor steps

- MotorApp.steptodegree()
  stepvalue
  Translates step value passed to the function into an angle in degrees.

- MotorApp.degreetostep()
  degreestring
  Translates an angle in degrees passed as a string to the function into a value in motor steps.

- MotorApp.loadcurrent()
  Queries the connected motors for position. Compares position to its own saved position (in "current.txt" file). If they do not match, displays sync error and asks for user input on which position to choose and which to override. If they match, asks user to confirm position. If right user input is given, calls MotorApp.initiatemotors() with the chosen positions

- MotorApp.initiatemotors()
  xpos,ypos,zpos,rpos
  Calls MotorApp.steptomm or MotorApp.steptodegree to get position in real units. Then opens an instance of class Motor() for every connected motor. Calls the thread functions to open separate threads and calls MotorApp.guiinit() to start the GUI.

- MotorApp.guiinit()
  Initializes the widgets of the Motor GUI. Calls the Motor.guipack() method on all motors to initiate individual motor GUIs. Calls MotorApp.loadfavourites() to display the saved positions.

- MotorApp.loadfavourites()
  Checks the file "favs.txt" for saved positions and displays them.

- MotorApp.secondthread(), MotorApp.thirdthread(), MotorApp.fourththread(), MotorApp.fifththread()
  Start new threads for all four motors to run Motor.update().

- MotorApp.emergencystop()
  Calls Motor.emergencystop() on all motors, updates error log and displays emergency stop message.

- MotorApp.restart()
  Called by restart button after emergency stop. Re-enables the individual motor GUI buttons to run the motors.

- MotorApp.setfavourite()
  Called by GUI set button. Gets the currently selected favourite position from the list and calls Motor.moveto() on all motors with the saved position.

- MotorApp.savefavourite()
  If motors are not moving, displays popup for user to choose position name and save current position.

- MotorApp.saveundername()
  Called from position saver popup. Saves current position under chosen name. Updates GUI with new favourite.

- MotorApp.closegui()
  Queries motor movement. If motors are not moving, saves current position in the "current.txt" file and adds error log to "errorlog.txt". Closes COM connection to motors and closes GUI.

1.8.2 class Motor()

Class for an individual motor. Each motor object has a serial port connection and positional variable.

- Motor.guiinit()
  Initiates the GUI widgets for the individual motor.

- Motor.guipack()
  Packs the motor frame with its widgets into the main GUI.

- Motor.update()
  Continuously runs in separate thread from mainloop. Queries motor for position and updates GUI with current position.

- Motor.changeposition()
  Checks step and real unit entries for new position, if necessary calls MotorApp.mmtostep() or MotorApp.degreetostep() for conversion. Calls Motor.moveto() with new position

- Motor.moveto()
  newposition
  Checks for absolute limit compliance and calls Motor.checkrellimits() to check relative limits compliance. If no limit violation, calls Motor.runmotor() to execute move to new position

- Motor.checkrellimits()
  Checks the relative limits depending on the positions of the other motors.

- Motor.runmotor()
  newposition
  Sends move command to motor with the position passed to the function while pausing Motor.update()

- Motor.stopmotor()
  Sends stop command to motor while pausing Motor.update()

- Motor.clearerror()
  Sends command to clear error while pausing Motor.update()
- Motor.emergencystop()
  Sends emergency stop command to motor. Disables the start/stop function of motor GUI
- Motor.querymovement()
  Called by other functions to check whether the motor is moving. Sends query about movement status to motor.

2 Hardware and software requirements

2.1 Hardware

Camera:
Should be usable with Spinnaker software/PySpin (written for FLIR BFLY-U3); Connected via USB3

Delay generator:
Connected via COM / USB virtual COM port. Needs to be able to receive basic SCPI commands. Written for BNC model 577 (some commands are specific); Serial to USB connection (virtual COM)

Laser:
Needs to be able to receive commands over serial. Written for Radiant dyes NarrowScan laser (commands are specific); RS232 or USB virtual COM

Motors:
Need to be able to receive commands over serial port. Written for Arun Microelectronics Ltd SMD3 motors (commands are specific); Serial to USB, min. 4 ports

2.2 Software

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.

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.
pyFieldMaxII (FieldMax II python wrapper): https://github.com/jscman/pyFieldMaxII (download from git and add to python library)
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

3 Installation and implementing hardware changes

3.1 Camera

Install drivers (see above) and connect camera via USB3
Camera change: 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.")
```

3.2 BNC delay generator

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 D (channel number 4) is used as the main scanning channel for delay integration and kinetic series experiments. This should therefore be the channel triggering the molecular beam (negative delays relative to laser q-switching). The other channels can be chosen arbitrarily. The reference channel is set manually on the delay generator and should be set once for every channel for the entire setup.

3.3 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 COM7, 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.

Power meter: The software is written to use the FieldMaxII power meter from coherent with a python wrapper published on git (https://github.com/jscman/pyFieldMaxII). This affects the wavelength and kinetic series modules. If another power meter is used, the code regarding the backend powermeasurement (anything communicating with "self.FMII") needs to be rewritten. Use without any powermeter is however possible without issue if the pyFieldMaxII module is installed in python.

3.4 Motors

Install and check motor firmware. If the connection works in the supplied software, it should work for the motor module as well. When scanning the COM ports, COM4 and COM5 are always skipped as these are assigned to other devices on the setup. If this changes, the following code blocks have to be deleted or the numbers changed:

```
class MotorApp(tk.TK):

    def portconnect(self):

        if self.portnum == 5:
            self.portnum += 1
            self.portconnect()
            return

        if self.portnum == 4:
            self.portnum += 2
            self.portconnect()
            return
```

If a motor is replaced, the serial number has to be changed in the code as the program uses the serial number to recognize and assign the motors. This can be done in the following part of the code:

```
class MotorApp(tk.TK):

    def __init__(self):
        self.serialnumbers = {"X": "20052-011", "Y":
            "20052-012", "Z": "20052-013", "R": "20052-014"}
```

If another type of motor is used, ensure that it can receive string commands and that the commands match the ones used for the SMD3 motors. Otherwise, the commands have to be edited.

4 Software repository

The software is in a public github repository which can be found here: https://github.com/LeaHohmann/NAPVMI_software

When making changes to the software, the git repository can be used to track changes and go back to a previous version. The procedure to use the git repository will be quickly described in the following. In order to get access to the git repository on your device, you need to install git: https://git-scm.com/book/en/v2/Getting-Started-Installing-Git

4.1 Making changes to the software

When editing the software in a local copy of the repository, for example on the laboratory computer, use the following procedure:

- Ensure you have the latest files of the software by opening a console in the software file folder (where the NAPVMI_main.py file is located) and use the command

```
git pull
```

- Make the necessary edits
- Update the online repository through the following series of commands in the console (open in the software folder):

```
git add [FILENAMES YOU HAVE EDITED]
git commit -m "DESCRIPTION OF EDITS"
git push
```

4.2 Creating a local copy

Create a local copy of the software repository on your device in order to edit any parts of the software. Do not manually upload to the git repository or copy files to the lab computer! This ensures that you can use the common git commands to track your changes and enable you to go back to previous versions if necessary. It also allows for the easiest syncing of the software files. To create a local copy, open a console or git bash and use the following command:

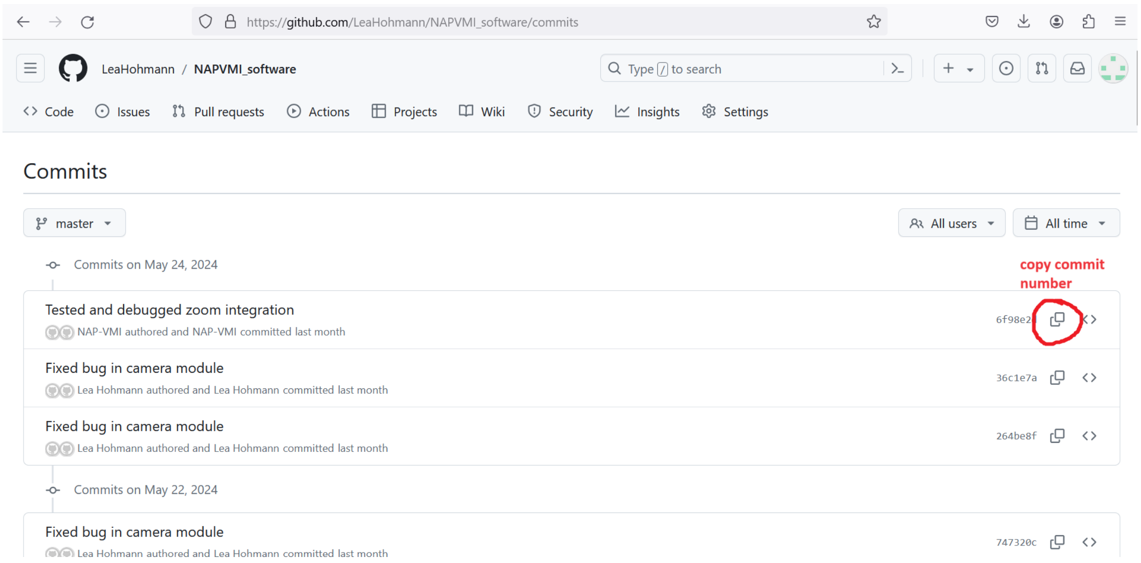
```
git clone https://github.com/LeaHohmann/NAPVMI_software.git
```

After creating a local copy, you can use the git commands to pull the latest version, and commit changes to the online repository.

4.3 Restoring a previous version of the software

If you have made changes and the software stops working, it can be easily reset to a previous version using the git repository. There are several different scenarios:

A) You have ”pushed” the changes to the online repository. In order to reverse this, you need to look up the commit number (SHA) in the repository history, which can be easily copied, see the example image:



Commits

Commits on May 24, 2024

Tested and debugged zoom integration
NAP-VMI authored and NAP-VMI committed last month
6f98e2

Fixed bug in camera module
Lea Hohmann authored and Lea Hohmann committed last month
36c1e7a

Fixed bug in camera module
Lea Hohmann authored and Lea Hohmann committed last month
264be8f

Commits on May 22, 2024

Fixed bug in camera module
Lea Hohmann authored and Lea Hohmann committed last month
747320c

Then you can create a reverse of this commit and commit it to the online repository with the following command (open console or git console in a repository folder):

```
git revert [SHA]
git push
```

B) You have made local changes but not committed anything. This can be very simply reverted by the following command, which re-downloads the version before changes:

```
git checkout –– [FILENAMES]
```

C) You have made local changes and committed them, but not pushed them to the online repository. In this case, you can reset the local repository using a previous commit where everything was still fine (usually the latest one if you have only made local changes), of which you have to find the SHA number as described above. Then the following command is used:

```
git reset ––hard [DESIRED SHA]
```