Bachelor Thesis

Institute for Biomedical Engineering, PSI and ETH Zürich,
Switzerland

PAUL SCHERRER INSTITUT

A Measurement Control Package for Grating-Interferometry X-Ray Imaging Laboratories

Lars Lenherr

2021

Supervision: Prof. Dr. Marco Stampanoni
Advisor: Dr. Michal Rawlik

Abstract

Imaging methods are required by medical institutions to get information about the structure of a human body and its diseases. For this reason the development of grating-interferometry-based imaging is important for the optimization of imaging methods.

In this project a framework is presented to support the laboratory developments. The dynamic nature of grating-interferometry-based laboratory developments favours quick, adhoc solutions and fast iteration time. Therefore the framework improvement is concentrated on the interface with the hardware and the measurement control. By analysing the structure, different components of a setup and the storage system, the possibilities and limitations of the framework are presented. Based on a grating interferometry setup the framework is demonstrated and several parts of the including code are explained.

Acknowledgements

My deepest gratitude goes to all people who supported me during my bachelor thesis.

First of all I would like to express my appreciation to my advisor, Dr. Michal Rawlik, for his support and his useful advices during the whole time at PSI. I enjoyed all the time during this project and learned a lot from him about how to approach tasks and how to solve issues.

Further I want to thank my supervisor, Prof. Marco Stampanoni, for the chance to write my bachelor thesis at PSI and to collect my first experience of a scientific work by a real project. It was a really great experience for me and will help me a lot in the future.

I would also like to thank Dr. Maxim Polikarpov, for all inputs and a good time during and besides the work. It's nice to spend time with you.

Further I thank Simon Spindler for all his support during my time at the PSI and his advices of different about several aspects of the project.

At last I would like to thank Lionel, for the useful exchanges, which motivated me a lot during the progress of my bachelor thesis.

Contents

1 Background

This chapter introduces the theoretical topics from X-rays up to the aspects of the Talbot interferometry. In this subject area the framework can be used to control a setup.

1.1 X-rays

X-rays were discovered by Willhelm Conrad Röntgen in Germany. In November 1895, he used Crookes tubes covered with black cardboard and a fluorescent screen to conduct experiments. By observing the screen he realized radiation that passes through the cardboard during the change of distance between the tubes and the screen. Since this discovery the research about new radiation, called X-rays, started to grow.[2]

X-rays follow the rules of electromagnetic radiation and are electromagnetic waves. By traveling through an area they transfer energy by waves and photons. To describe the behaviour of X-rays the wave model or photons can be used. [2]

X-rays have a characteristic wavelength between 0.01nm and 10nm and as electromagnetic waves with their velocity is the speed of light. The corresponding radiation energy is directly related to the wavelength of the photons. When X-rays, as light, propagate through a medium, the certain amount of transmitted photon is medium-dependent.[2]

Every material has a characteristic absorption behaviour which will reduce the amount of radiation energy. The reduction relates to the attenuation coefficient and leads to a contrast in an image. This builds the main principle of X-ray imaging. [2]

The logarithm of the attenuation is proportional to the thickness of the material and its attenuation coefficient. The latter varies between kinds of tissue in the human body, which makes it possible to visualize their distribution with through-going X-rays. [2]

1.2 X-ray Imaging

The properties of particle describe the behavior of X-rays when they interacting with matter, for instance the photoelectric effect or the scattering. In addition the wave model explains attenuation, refraction or interference. The properties allows to describe the internal structure of an object.[3]

By passing through matter several aspects of the emitted X-rays get changed, for instance the photon flux or the photon energy. The transmitted X-rays hit the surface of the detector and results in an image of the passed object.[2]

1.3 Grating Interferometry

In figure 1.1 a setup of a Talbot-Lau grating interferometery can be seen. The Setup includes a X-ray source, a detector and the interferometer. G0 is an absorption grating and is located in front of the source. The grating has periodic structure and is usually made of strong-absorbing materials, gold for instance. This leads to a periodically absorbing and transparent behavior. If the spatial coherence of the beam isn't high enough the G0 have to be used to split the source into micro-sized sources which are lined up periodically. They are individually coherent, but mutually incoherent. Sources which provide enough spatial coherence, for instance synchrotrons or microfocus sources, do not require a G0.[3]

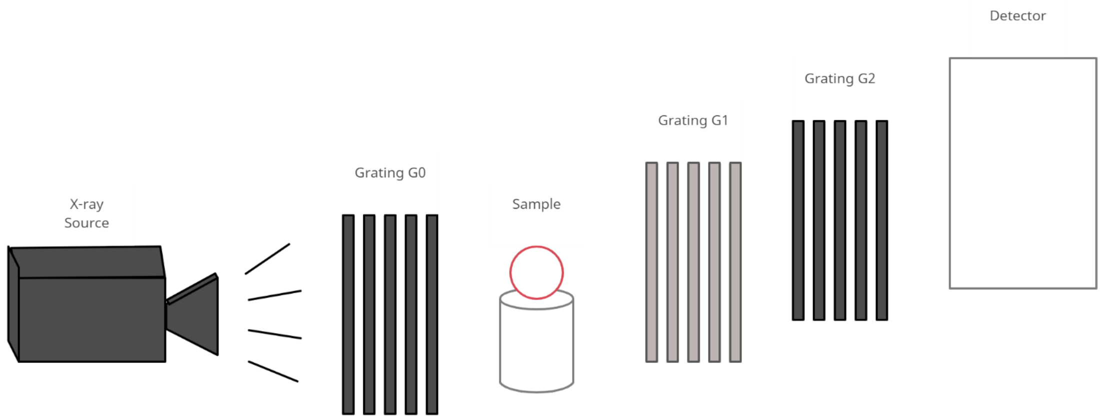

Figure 1.1: A Talbot-Lau Interferometry Setup

To split the beam another required grating is G1, placed directly after the sample. G1 is also periodically structured but is different to G0. The phase grating, G1 is often created of gold, too ,but a much thinner layer than G0 and shifts the phase of the X-ray wave for a certain value, which is usually π. This results in a periodic phase modulation with high intensive fringes. This effect is base on the Talbot effect, for this reason the name is given by Talbot interferometer.[3] The optical Talbot effect can be seen in figure 1.2. [4]

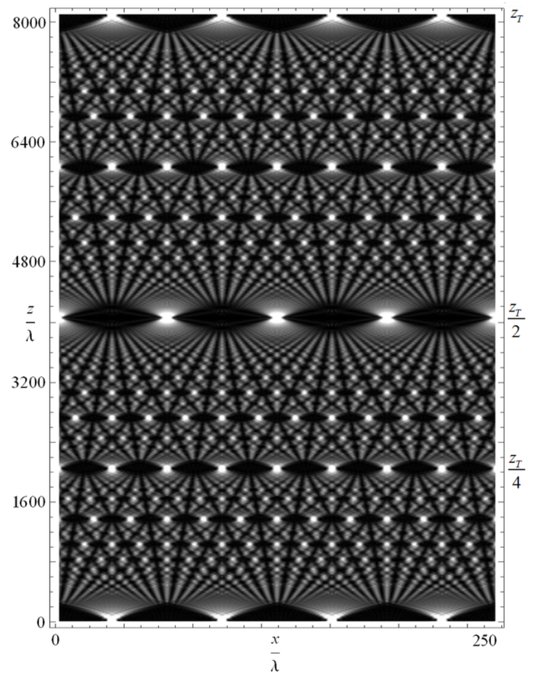

Figure 1.2: The Talbot Carpet. z_T: The Talbot Distance, x: The transverse Direction, z: The Downstream Direction, λ: The Wave Length (adapted from [4])

The foundation of the grating interferometry is the diffraction of a periodic object. The downstream of gratings results in an interference pattern, which is referred to as the Talbot effect. If coherent radiation illuminates a periodic structure, e.g. a grating with transmitting slits, the wave field becomes the same as the complex transmission function of an object at a certain distance downstream. The complex transmission function is determined by the different properties of the object. The distance is referred to as the Talbot distance.[3]

As the object interacts with the X-rays, the attenuation, refraction and scattering of the beam change the interference of the pattern.[3] The attenuation can be determined by the photoelectric effect, the coherent and incoherent scattering, which are depending on the photon energy and the material. The refraction leads to a direction change of photon transmission at the air-object interface related to the different velocities of X-ray in materials.[1]

These interactions results in a reduced fringe amplitude by scattering and a fringe shift determined by refraction and a reduction of the intensity because the attenuation. [3]

A detector can efficiently measure the changes of the interface pattern, which is based on absorption, phase and scattering signal. The pixels of the detector can't measure directly the fringe-shift, because the pixels are usually larger than the spatial period of the fringes. For that reason a third grating is located between G1 and the detector. The grating is an absorption grating so-called G2 has the same characteristics as G0 but has the same period as the intensity fringes. It distributes the intensity in a way that a pixel has high respectively low intensity. So fringes are attenuated or transmitted. The distance between the gratings can be determined by the fractional Talbot order. This means that when a maximum occurs during a downstream of G1, the distance is so-called fractional Talbot order. [3]

To analyse the interference pattern, phase stepping of the grating G2 can be used. For this reason the grating moves several steps over one period with equidistant lengths in transverse beam direction. The intensity curve becomes a convolution of the interference after the transmission and this can be measured by a detector. [3]

2 Purpose

The aim of this project is to create an adaptable framework for laboratory development of grating-interferometry-based imaging, concentrated on the interface with the hardware and measurement control. The framework should provide a simple integration of devices and a flexible base on which an application for a specific grating interferometry setup can be created. This framework will help to set up a grating interferometry system and to document the measurements.

Laboratory devices use many different Application Programming Interfaces, as APIs. They enable the connection and the control of a device. An API is provided for only some programming languages, might be a considerable effort to integrate a device.

At the same time the dynamic nature of the laboratory developments favours quick, adhoc solutions and fast iteration time. Building an intermediate layer would homogenise the way to control the devices reducing the overhead of implementing a device in a setup. Furthermore, an established and well-documented interface makes it possible to separate the task of the technical integration of a device from the use in the laboratory context. It is very much like the way it is done in synchrotron beamlines.

A user-friendly python interface is based on a device class. A class holds the device-specific save instructions and provides the functionality via functions. These enable the communication between the computer and the device, which is needed for measurements, through the python interface. So an application can change the settings of a detector by using the functions provided in the corresponding interface.

A device and the corresponding interface build a package, which can be used from other scripts. As a main application a script can be created to import all classes of the device interfaces and therefore a setup is controllable. Based on the interfaces the script includes several functions which are defined by a user and can execute measurements.

A main application can be build in many different ways, in this context the main application is a python script. An other possibility is to use a Jupyter Notebook. With the main application a user can comfortably control all devices which are connected to it.

As an example, a measurement can execute a simple projection with a detector during an X-rays emission from a tube. So the function consists of several commands to prepare and activate the tube and the detector on the right time to get a good image. To know if the measurement started and terminated successfully, somewhere in the measurement-function needs to be a command which takes note of the current process.

For a documentation about measurements, information about the success, the date, the start and end time are essential. Normally the information part always consists of a specific part which depends on the devices of the setup and a non-specific part which includes all general information about the setup. The specific part is located in the application and can provide the model number, the settings and so on. On the other hand the unspecific part can be located outside of the application and normally notes the current measurement number and the information if the measurement terminated successfully.

The separation between the application and the non-specific part results in a better flexibility of the framework. The class can be imported by the main script to provide the functionality. Therefore a user doesn't have to write a new one. Based on the separated information part a specific one can be build. The part based on the devices have to be created by a user for a new main script.

To build a framework for a setup, the user has to connect the hardware to the computer, import the interfaces from each device in the main application and to create his own measurement-functions. If a setup has to integrate a device the measurement function in the main script has to be modified as well.

3 Implementation

In this chapter the functionality and the structure of the developed framework will be discussed. Based on example scripts, the advantages and limitations will be shown and explained. Because of space reasons only small pieces of the program code will be shown in this chapter, but the whole program code is available in the attachment.

3.1 GitLab

GitLab is a framework for hosting git repositories. An other advantage of "GitLab" is to integrate different projects in other project. This way a project can be easily integrated and used in one or more projects without restrictions.

GitLab contains only git-repositories. A repository can be a collection of sub-repositories and scripts, which makes up a project. The following Image shows an example of a git repository, which contains a main script and several sub repositories, called "measurementclasses" and "dexdetector".

Name	Last commit	Last update
dexdetector @ b096d249	Add detector interface	21 minutes ago
measurementclasses @ 4003e554	Add Measurementclass	20 minutes ago
.gitmodules	Add Measurementclass	20 minutes ago
try.py	Update try.py	11 minutes ago

Figure 3.1: A git repository

A repository can easily integrate other projects as sub-repositories, make use of it and can add self-created scripts. So a user can build his own project on different other projects. In figure 3.2 an exemplary structure of four repositories are shown.

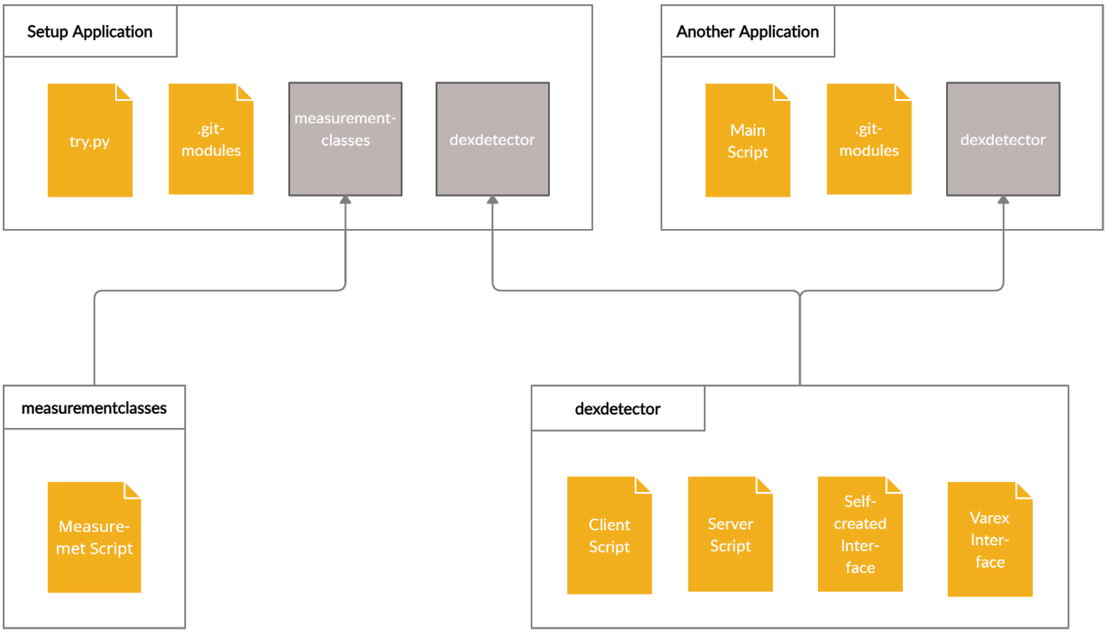

Figure 3.2: Possible Structure of Repositories

3.2 The Integration of a Device

In this section the integration of a device and his corresponding framework will be explained based on an example of a Varex 2315 Dexela detector.

A device needs that interface to enable the connection between the control application and the devices itself. Most companies sell devices with a corresponding documentation API. To make use of the "Dexela" detector an interfaces class is written to simplify the use of the given interface of the company. The structure is shown in figure 3.3.

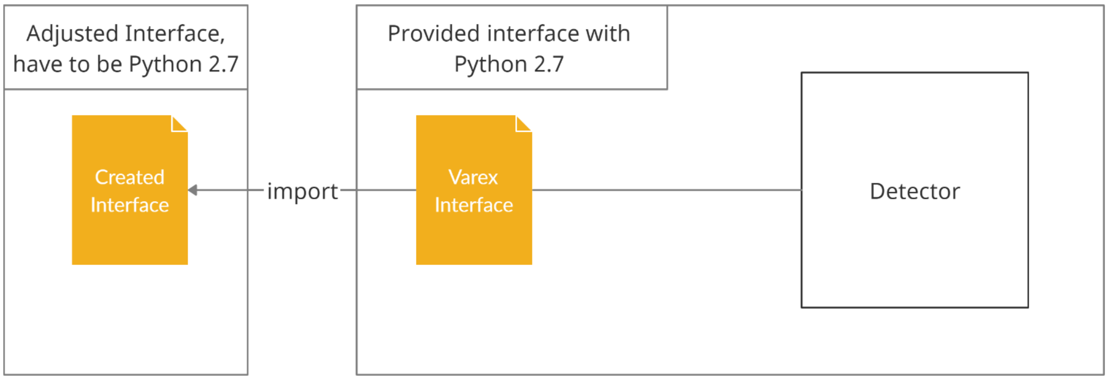

Figure 3.3: Interface-Device Connection

The detector has five settings which are relevant to acquire images, these are: the exposure time, the well mode, the binning mode, the trigger source and the exposure mode. These settings will have impact on the quality of the image.

The class is called "DexDetector" and defines all functions and the initialisation of the detector. An object of this class is initialised with standard settings to acquire a single image. These provide a fast and uncomplicated use of the detector for taking an Image.

If other settings are needed there are set-functions which can change the settings to the preferred values. To verify that the right values are adjusted or just to get the information about the actual settings, several get-functions are defined which read the current values out of the detector. Every function prints the information in the terminal by calling it.

Next to all set- and get-functions there are four functions to acquire one or more images in different ways. The functions can be found in the attachment, but won't be explained in detail. The detector class provide four different functions to get images from this detector.

Normally the functions and processes which allows the creation of application that access the features of a device aren't available for all programming languages. Sometimes even for very specific versions, as seen in figure 3.3. Therefore the homogenisation happens by providing a class-based interface in python 3.

Since the interface of the company is python 2.7, a binary version only compatible with a version older than the one of used setup framework. Therefore, two scripts are built to communicate with each other using ZMQ sockets. Sockets make it possible to combine different python versions together.

Figure 3.4: A Socket Connection between two Scripts

The first script is connected to the interface of the detector and the program itself has the behavior of a server. This allows other programs to connect to it via a port which is defined in the server script. Programs which connect to the server are called clients.

Through the server-client connection data can be send from the client to the server and back to it. The server will look at the received message and if it's a valid command, it will call the right detector-function. This way a message from a client can reach the detector. After a call the server will sent the information about the function back to the client. So client is updated if the server has received a message and if the called function has executed correctly.

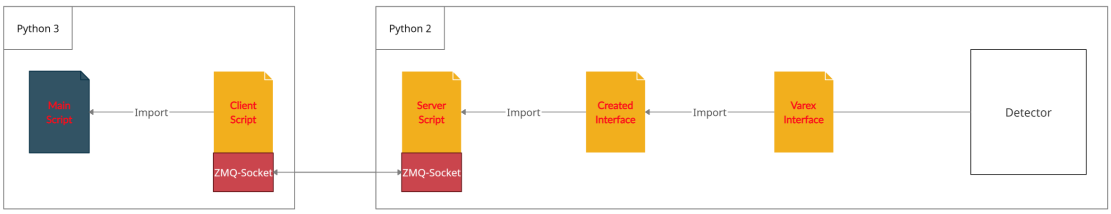

Figure 3.5: The Distribution of the Python Versions for a Main Script written in Python 3

The client script consists of a class which provides functions to send strings to the server. For each defined funcion of the "DexDetector" class, there is one function defined in the client script for reaching the detector via the server. The class in the client script is called "TranslateDet".

The main script of the setup imports the "TranslateDet" class to get a connection via the server to the detector. This is a useful way to create a compatibility between different versions of programming languages.

For instance if a user wants to know how long the exposure time is set in the settings of the detector, he will create an object of the "TranslateDet" class and use the get-function of the exposure time, which is showed in figure 3.6. Since all functions of the client script afford the server script to send an answer back, the receipt part is defined out of the functions where it can be used independent from the functions.

```
def get_exposure_time(self):
    send('get_exposure_time')
    return
```

Figure 3.6: The get-Function of the Exposure Time from the Client Script.

This get-function will send the string "get_exposure_time" to the server. As the string reaches the server, the string is verified if this is a valid command. Since all function of the "TranslateDet" class only send valid strings, the server will find the corresponding function call.

In figure 3.7 the code can be seen, which the server needs to check if the message is valid. Since the message is valid the "det.get_exposure_time"-function get called to define the string "s" and the bool "reply" get set to true.

```
if(message[0] == 'get_exposure_time'):
    s = ('exposure time: %s' %(det.get_exposure_time()))
    reply = True
```

Figure 3.7: The Check from the Server if the received Message is valid

The bool "reply" means that information from the server has to be sent back to client. Only get-functions contains this bool because set-functions doesn't have to return information about the values of the detector, since the values which will be set are known. This is worthwhile keeping the work of the computer as small as possible.

The command "det.get_exposure_time" calls the function in figure 3.8. This function is in the self-created interface of the "Dexela" detector. The function has a connection to the detector through the interfaces from the company and return the information.

There are other aspects which are important to know about the server-client connection. The server can only make one connection at the same time, if a second client connects to the server, the first one will lose the connection. This won't affect the use of the detector as one setup will normally be controlled by one user. This is a solution to control the detector from another script with an other python version.

```
def get_exposure_time(self):
    exposureTime = self.detector.GetExposureTime()
    print("exposure time: %s" %(exposureTime))
    return exposureTime
```

Figure 3.8: The "get_exposure_time"-function from the self-made detector class

3.3 The Structure of a Framework

A setup can have several devices like an X-ray tube, a detector and different motors. To include the devices into the framework of a setup the corresponding interfaces have to be integrated. The interfaces are required to enable the connection and to access the control of a device. They can consist of translation scripts and a device class. The translations scripts are only needed if the versions or the languages of the interface and the framework are different.

In the device classes are the definitions of all functions which provide the application. Python classes are simple to import in other scripts. As a device interface is independent from other scripts except the corresponding translation scripts, the interface and the translation scripts are located in a git-repository.

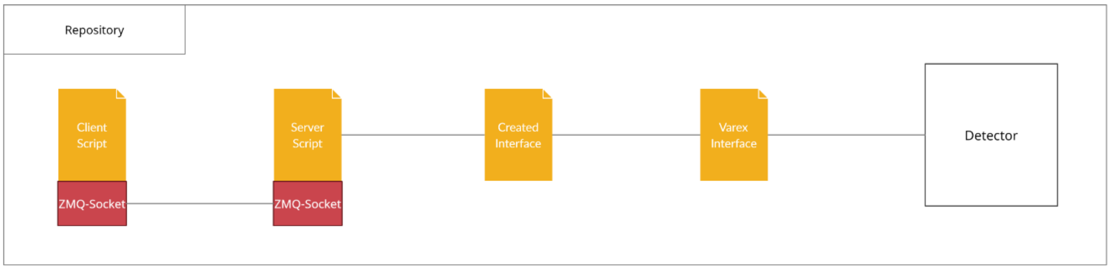

Figure 3.9: Example of the 2315 Dexela Detector Repository

The repository in figure 3.9 can be imported from other repositories. So the creation of a framework is supported by the simplified interface integration of the repositories. This leads to quick ad-hoc solutions for a setup.

To create a framework for a setup a git-repository can be built which adds device interfaces as sub-repositories. Next to the sub-repositories a main script has to be created. Since the interfaces are defined in classes, they can be easily imported in the main script. So the class and the functions of the interfaces can be integrated and used in the main script. As seen in figure 3.10 repositories can be shared between setups, for instance they can be integrated in multiple main scripts.

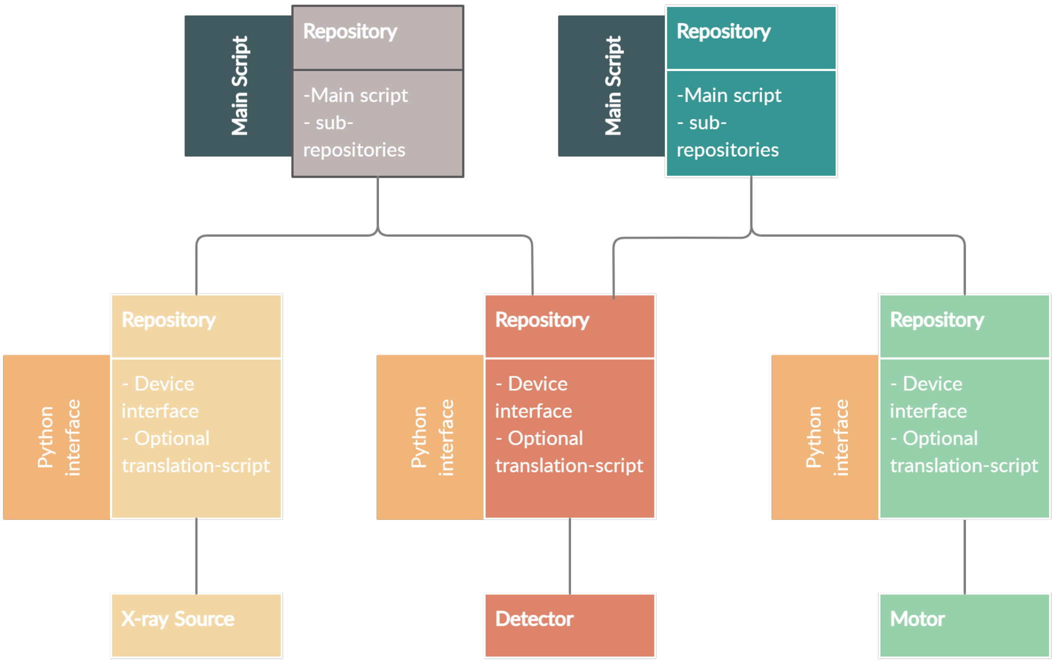

Figure 3.10: A Framework Structure

The main script acts as the application of the framework and therefore a setup can be controlled. Several parts are defined in the main script to provide the functionality of the framework. There are the layout, the management of the devices, several information functions and measurement functions defined. The main script will help the user to control the setup in preferred ways.

The layout of a framework depends on the preferred type of the application. For example, the framework can be used in a command line. All definitions of required functions are written in the upper part of the main script and at the end of the script a command will executed which opens a python application in the command line.

To use the function of the interfaces in the python application, the objects of the required devices classes have to be created. The main script needs a management of the devices for this reason.

Normally there are more devices connected to the framework as needed for the measurements, because several devices are used interchangeably, swapping an X-ray source for instance. The management limits the devices to the required ones, therefore the work of the computer will be reduced and the overview about the devices becomes better.

A possible way to build a management is to use the "argparse" standard module of python. The module provides the possibility to define several arguments in the main scripts. These arguments can be added by executing the main script to create and initialise the object of the required interface classes. To create an overview about the objects the "logging" standard module can be used to print a message if one is built. So the management provide the initialisation of the devices which will be used for the measurements.

To execute a measurement a function can be defined in the main script. So the function includes all command to prepares the devices and to carryout a measurement. A function has to be structured as the measurement itself.

An example of a possible measurement can be an acquisition of an image with a detector during X-ray emission. A function has to be structured as the measurement itself. This can be called for acquiring a single image, the commands for the preparation of the devices are executed as first. So the devices get the preferred settings, then the detector acquires the image and save it on a chosen location. A call of a measurement function execute a chosen process.

To document measurements during the process, information function have to be defined. Normally a documentation contains the information about the success, the date, the start time, end time and used devices. So the documentation is independent of the device except the information about the devices itself. A separation of the independent part leads to use it in frameworks for any other setups and the documentation part becomes more flexible.

The part about the device can be based on the separated one and has to be written in the main script by a user. Using these structure will result in a certain flexibility and a well-documented output.

The code to provide the storage organisation has different functions which create a folder structure in the location which can be defined. Therefore the folder on the same stage as the script has the name of the year and the subfolder in in which the output of the measurement are stored has the name of the actual month. As output the acquired image, logging and meta files will be saved in this location. The two files provide the device independent measurement information.

In this way the functionality of the documentation part is unchanged, but the framework becomes more flexible and the integration of a device simpler. As a second advantage it's simpler to create a new application for a new setup, because the independent documentation part can easily get imported.

So the structure of the framework is created to provide a comfortable integration of devices and a good flexibility which supports users in building a new setup.

4 Example

This section is about building an example script to use a simple setup including just a detector and a X-ray source. The detector is controlled with the program of the main script which acts as the framework. For simplicity the X-ray source is separated from the framework and is controlled by other programs. Therefore one device has to be integrated in the system and the corresponding interface in the main script.

There are several python standard modules in the main script which are used and won't be explained in this project. The link to documentation can be found in the attachment. The main script build a git repository which import the repository from the detector as a sub-repository. Therefore the interface of the detector is imported and the functionality of it can be used in the main script.

The main script has four different parts. The first part is about the layout of the framework, the second part is created for the management and initialisation of the devices, then the part to save the information of the measurements and the last part includes measurement functions. The main script has to be created from a user to control a specific setup.

The layout part refers to how a user wants to use the framework, how and where the information about the measurements are saved. This file is build to use it as framework in a command prompt. Therefore the script can be called in a terminal, then the computer go through the definitions of the program and at least it opens a python terminal in which all defined functions of the main script and all imported functions can be used.

```
$ winpty python try.py
In [1]:
```

Figure 4.1: The Layout of the Framework

To build the management of the devices, the standard python "argparse" module is used. In the main script several arguments for devices can be defined. A user can give the arguments to the executive command of the main script to include the required devices. Figure 4.2 represents an example the definition for an "–dexdet" argument of a detector.

Then if the argument of the detector is given to executive command, there are various parts defined to initialize the devices and start the script to translate the different python versions to each other. When the computer start running the script the terminal looks like figure 4.3 and waits for additional commands. So the devices can easily get activated at the start of use.

```
parser.add_argument("--dexdet", action= "store_true",
    help="TranslateDet")

if args.dexdet:
    stored_image_path = 'C:/Users/gac-gi_bct/Desktop/example/picture'
    subprocess.Popen(["C:/Users/gac-gi_bct/Anaconda3/envs/py2/python.exe"
    ,"C:/Users/gac-gi_bct/Desktop/Setup/Detector/DexDetector/server.py" ])
    dexdet = TranslateDet(stored_image_path)
    logging.info("Connected to the Dexela Detector 2315")
    logging.info("CREATED OBJECTS: dexdet.")
```

Figure 4.2: The Definition of the Detector-Argument (upper part) and the Commands to initialise the Detector if the Argument is given (lower part).

From this point a user can call every function of a device interface. As seen in figure 4.3 the detector prints an overview of the initialised values in the terminal. To transport the information from the detector to the framework a connection is required. The python versions are different, for this reason a server and a client script are created. A server and a client script are written to translate the different python versions to each other. So the framework and the detector can communicate together. The terminal includes a reply of the server as well, which can be seen in figure 4.3. This reply includes the information if the initialisation is completed and which port is used for the server-client connection.

```
$ winpty python try.py --dexdet
Connecting to detector via server...
Scanning to see how many devices are present...
1
ready to acquire some images from detector with serial number: 55086
Initializing detector settings...
expMode: Expose_and_read
exposure time: 250.0
binning: x11
wellMode: High
trigger source: Internal_Software
Received reply  1 message:  b'init completed, connected port : 5556'
2021-05-19 14:15:04,070 try              INFO     Connected to the Dexela Detector 2315
2021-05-19 14:15:04,072 try              INFO     CREATED OBJECTS: dexdet.

In [1]: |
```

Figure 4.3: The Termination of the Main Script with a given Argument

To save various information about a measurement, a class is created which can be used for different setup. The class is based on the "logging" standard module and is used to save the date, the system number of the measurement, the start and end time of the measurement. The information is written into a log file.

The "yaml" module is used to create a meta file, which stores the type of the measurement and if the measurement completed successfully. this leads to a simple overview about the done measurements.

```
2021-05-19 15:07:19,196 Measurement     INFO     Starting 'simple_snap' measurement: 20210519_150719195066.
2021-05-19 15:07:20,227 Measurement     INFO     Finished measurement 'simple_snap' measurement: 20210519_150719195066.
```

Figure 4.4: A log-file crated by a measurement with a detector

```
Measurement:
  success: true
  type: simple_snap
```

Figure 4.5: A generated yaml-file including the type of a measurement and the success

The measurement class is separated from the main script. The main script imports the class to modify it based on current setup. For example there is the information about the detector saved as dictionary. The script can be seen in the attachment.

The last part of the framework consists of measurement functions. To execute a single image acquisition, a user can define a function to prepare the devices and to carry out the process. Figure 4.6 shows the definition of measurement function from the example script. Therefore a "B45Measurement"-object is created with the type as "simple_snap" and the storage path. The commands "m.start()" and "m.stop()" save the information about start and end time of the measurement. The part between those commands executes the acquisition of one image if the argument of the detector was given.The outputs of one measurement are an image, a log-file and a meta file.

```
def Simple_snap():
    m.start()
    if args.dexdet:
        dexdet.change_path(m.get_path().get_measurement_directory())
        dexdet.AcquireSingleImage()
    m.stop()
    return m.name, m.success
```

Figure 4.6: An example of a measurement-function

After calling this function the terminal ends reaches the stage as shown in figure 4.7. There all information about the measurement and the detector settings can be seen.

```
$ winpty python main.py --dexdet
Connecting to detector via server...
('Received request: ', ['-1', 'C:/Users/gac-gi_bct/Desktop/Setup/Detector/Images'])
Scanning to see how many devices are present...
1
Acquiring single image from detector with serial number: 55086
Initializing detector settings...
exposureTime: 250, Binning: x11,wellMode: High, trigger: Internal_Software,
expMode: Expose_and_read
expMode: Expose_and_read
exposure time: 250.0
binning: x11
wellMode: High
trigger source: Internal_Software
Received reply  1 message:  b"init completed, connected port : 5556, info: ['exposuremode : Expose_and_read', 'exposuretime : 250.0',
'binning : x11', 'wellmode : High', 'triggersource : Internal_Software']"
2021-05-25 14:31:57,541 main            INFO     Connected to the Dexela Detector 2315
2021-05-25 14:31:57,541 main            INFO     CREATED OBJECTS: dexdet.

In [1]: Simple_snap()
2021-05-25 14:32:39,024 Measurement     INFO     Starting 'simple_snap' measurement: 20210525_143239024777.
Connecting to detector via server...
('Received request: ', ['AcquireSingleImage'])
Grabbed Image!
Image successfully saved!
Received reply  1 message:  b'task completed, connected port : 5556, Image successfully saved! filename: Image1_3072x1944.smv '
2021-05-25 14:32:40,291 Measurement     INFO     Finished measurement 'simple_snap' measurement: 20210525_143239024777.
```

Figure 4.7: The Terminal after calling a Measurement Function

A user can repeat this process as long as preferred. After the function call the output consists of a meta file, a logging file and the acquired image. to save the files the measurement class creates folders in a certain path to save the log and meta file. In addition the storage path of the detector can be changed to the same directory as seen in figure 4.8. So the three output files are stored in the same location.The image in of figure 4.8 is shown in figure 4.9 where a chocolate bar with rice crisps is used as a sample. If the measurements are finished and the framework isn't used anymore, the command "exit" can be executed. This will closed the script.

The framework can include every device with a corresponding interface. If the setup changes the measurement-functions, the argument part and the information part has to be updated. In addition, the measurement class provides a file-naming scheme and a nested storage framework.

```
Folder PATH listing for volume System
Volume serial number is 5E05-6936
C:.
|   output.txt
|
\---Detector
    |   main.py
    |
    +---2021
        +---04
        |   +---20210429_160510983831
        |   |       20210429_160510983831.log
        |   |       20210429_160510983831.yaml
        |   |
        |   +---20210429_160535878372
        |   |       20210429_160535878372.log
        |   |       20210429_160535878372.yaml
        |   |
        |   +---20210429_161220578771
        |   |       20210429_161220578771.log
        |   |       20210429_161220578771.yaml
        |   |
        |   \---20210430_134953690058
        |           20210430_134953690058.log
        |           20210430_134953690058.yaml
        |
        \---05
            \---20210525_161056478868
                    20210525_161056478868.log
                    20210525_161056478868.yaml
                    Image1_3072x1944.smv
```

Figure 4.8: The Folder Path Tree of the Information Files and the Image

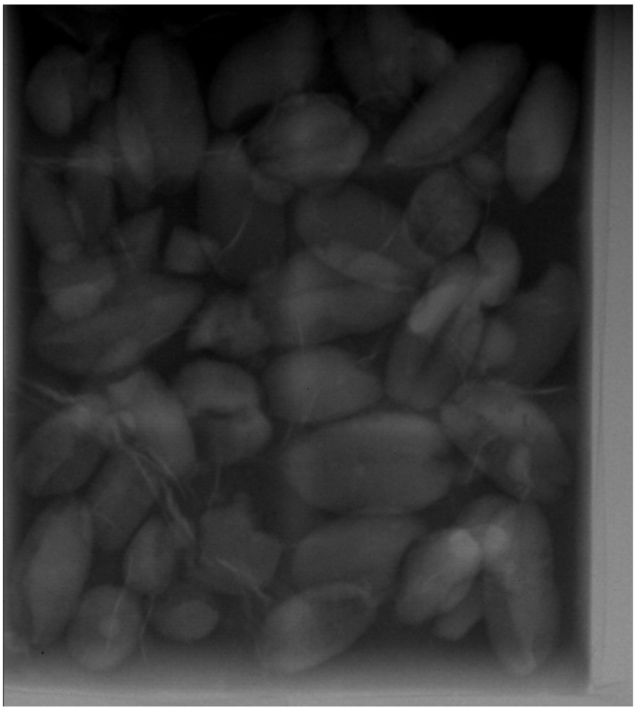

Figure 4.9: An acquired Image of a Chocolate Bar including Rice Crisps

5 Conclusion

A framework can be used to control a grating interferometry setup. For laboratory developments quick ad-hoc solutions, small iteration times and organised storage system are desirable. To support the development the framework has to simplify the integration of devices, hence it sometimes results in a huge effort.

This means that the effort to integrate devices has to be as simple as possible without limiting the functionality of the framework. Therefore the separations of setup-independent part results in certain flexibility of the framework, which provides a simplicity by a device integration. The separations can be made without any impact on the functionality, since all device dependent parts are saved in a main script. The main script provides the application of the framework.

The main script provides the functionality of the framework which will be designed by a user to create an application of the setup. By defining measurement functions the execution of measurements can be simplified and adapted as preferred. To proof the measurements executed successfully and to have an helpful overview about the accomplished measurement, a documentation about it is created. Since a documentation is based on setup independent parts, a separation leads to an uncomplicated creation of a new setup-application.

To control a setup with the framework, a user has to create a application of it and to integrate the devices. Therefore the framework requires a certain flexibility which is created by minimizing the effort of device integration and the documentation of the measurements without limiting the the functionality. The constructed framework will support the laboratory developments of grating-interferometry-based imaging.

6 Attachment

6.1 An Example of an Application Script

```
import os
import numpy as np
import time
import datetime
import yaml
import h5py
import threading
import coloredlogs
import logging
from DexDetector import TranslateDet
from time import sleep
import subprocess
import IPython
from MeasurementClasses.Measurement import Setup, Measurement
import argparse

path = "C:/Users/gac-gi_bct/Desktop/example/"
log_format = "%(asctime)s %(module)-15s %(levelname)-8s %(message)s"

### specified setup class
class B45Setup():

    def __init__(self, path ):
        self.setup = Setup(path)
        self.path = path

    def get_setup(self):
        return self.setup

    def get_measurement_directory(self,name):
        return self.setup.get_measurement_directory(name)

    def get_new_prefix(self):
        return self.setup.get_new_prefix()

    def create_measurement_directory(self, prefix):
        return self.setup.create_measurement_directory(prefix)

    def savemeta(self, name, initdict = {}):
        tubeinfodict = {
```

```
            "model": "NA" }

        if args.dexdet:
            detectorinfodict = {
                "model" : "Dexela 2315 (CL)",
                "exposure mode" : dexdet.get_exposure_mode,
                "exposure time" : dexdet.get_exposure_time,
                "binning" : dexdet.get_binning,
                "buffer dim" : dexdet.get_bufferdim,
                "trigger source" : dexdet.get_trigger_source,
                "well mode" : dexdet.get_well_mode,
            }
        else:
            detectorinfodict = {
                "model": "NA" }

        infodict = {
            "Tube": tubeinfodict,
            "Detector": detectorinfodict,
            **initdict }

        self.setup.savemeta(name, initdict)

    def snap(self, prefix="", exposure_delay=None):
        name = create_measurement_directory(prefix=prefix)

        #Acquire a single image with the detector
        if args.dexdet:
            dexdet.AcquireSingleImage()


        infodict = {
        "Measurement": {"type": "single_exposure"} }

        self.savemeta(name, infodict)

        logging.info(f"Saved single exposure image {name}.")


        return self.setup.snap(prefix,exposure_delay)


### specified measurement class for bunker 5
class B45Measurement():

    def __init__(self, type, path, prefix="", autostart=True):
        self.setup = B45Setup(path)
        self.measurement = Measurement(type, self.setup, path = path,
    prefix = prefix)
        self.success = False
        self.name = self.measurement.get_name()

    def start(self):
```

```
        self.success = True
        return self.measurement.start(self.setup.get_setup())

    def stop(self, info={}):
        self.measurement.stop(self.setup, info)

def Simple_snap():
    m = B45Measurement("simple_snap", path)
    m.start()
    if args.dexdet:
        dexdet.AcquireSingleImage()
    m.stop()
    return m.name, m.success

def setup_logging(loglevel):
    logfile_folder = "C:/Users/gac-gi_bct/Desktop/Setup/Detector/Log"
    logfile_date = datetime.datetime.now().strftime("%Y%m%d_%H%M%S")
    logfile_filename = f"bunker5_{logfile_date}.log"
    logfile_full = os.path.join(logfile_folder, logfile_filename)
    logging.basicConfig(
        filename=logfile_full,
        level=logging.DEBUG,
        format=log_format)
    coloredlogs.DEFAULT_LOG_FORMAT = log_format
    coloredlogs.DEFAULT_FIELD_STYLES["module"] = {"color": "magenta"}
    coloredlogs.DEFAULT_FIELD_STYLES["asctime"] = {"color": "blue", "faint": "true"}
    coloredlogs.DEFAULT_FIELD_STYLES["levelname"] = {"color": "cyan"}
    coloredlogs.DEFAULT_LEVEL_STYLES["debug"] = {"color": "black", "bright": "true"}
    coloredlogs.install(milliseconds=True)
    coloredlogs.set_level(loglevel)

if __name__=="__main__":
    parser = argparse.ArgumentParser(
        description="Command-Line interface for the Project5 setup.")

    parser.add_argument("--dexdet", action= "store_true",
    help="TranslateDet")

    parser.add_argument(
        '-d', '--debug',
        help="Print lots of debugging statements",
        action="store_const", dest="loglevel", const="DEBUG",
        default="INFO")
    args = parser.parse_args()

    setup_logging(args.loglevel)

    if args.dexdet:

        stored_image_path = 'C:/Users/gac-gi_bct/Desktop/example/picture'
        subprocess.Popen(["C:/Users/gac-gi_bct/Anaconda3/envs/py2/python.
```

```
    exe","C:/Users/gac-gi_bct/Desktop/Setup/Detector/DexDetector/server.py
    "])
        dexdet = TranslateDet(stored_image_path)
        logging.info("Connected to the Dexela Detector 2315")
        logging.info("CREATED OBJECTS: dexdet.")

    # start an interactive IPython session
    IPython.embed(banner1="", colors="Neutral", autocall=2)
```

6.2 The generalized Information Classes

```

import logging
import os
import threading
import h5py
import datetime
import yaml




class Setup():
    def __init__(self, path ):
        self.path = path

    def get_measurement_directory(self, name):
        year = name[:4]
        month = name[4:6]
        return f"{self.path}/{year}/{month}/{name}/"

    def get_new_prefix(self):
        return datetime.datetime.now().strftime("%Y%m%d_%H%M%S%f")

    def create_measurement_directory(self, prefix=""):
        name = self.get_new_prefix()

        os.makedirs(self.get_measurement_directory(prefix + name))
        return prefix + name

    def savemeta(self, name, initdict={}):

        fname = os.path.basename(name)
        fullpath = os.path.join(self.get_measurement_directory(name), f"{
    fname}.yaml")
        with open(fullpath, "w") as f:
            yaml.dump(initdict, f, default_flow_style=False)


    def snap(self, prefix="", exposure_delay=None):
        name = self.create_measurement_directory(prefix=prefix)

```

```
        infodict = { "Measurement": {"type": "single_exposure"} }

        self.savemeta(name, infodict)

        logging.info(f"Saved single exposure image {name}.")

        return name


class Measurement():



    def __init__(self, type, setup, path, prefix="", autostart=True):

        self.type = type
        self.prefix = prefix
        self.name = setup.create_measurement_directory( prefix)
        self.success = False



    def get_name(self):
        return self.name

    def start(self, setup):
        self.success = True

        log_format = "%(asctime)s %(module)-15s %(levelname)-8s %(message)s"
        # log to a file
        self.logger = logging.getLogger()
        logfile_path = os.path.join(
                setup.get_measurement_directory(self.name),
                f"{os.path.basename(self.name)}.log")
        self.logging_handler = logging.FileHandler(logfile_path)
        self.logging_handler.setLevel(logging.DEBUG)
        self.logging_handler.setFormatter(logging.Formatter(log_format))
        self.logger.addHandler(self.logging_handler)

        logging.info(f"Starting '{self.type}' measurement: {self.name}.")

        return self.name


    def stop(self, setup, info={}):
        """
        info : dict
            A dictionary with additional information about the measurement.
        """

```

```

        measurement_dict = {
            "type": self.type,
            "success": self.success
        }
        infodict = { "Measurement": {**measurement_dict, **info}}
        setup.savemeta(self.name, infodict)

        logging.info(f"Finished measurement '{self.type}' measurement: {
    self.name}.")

        # remove the logger
        self.logger.removeHandler(self.logging_handler)
```

6.3 The self-created Interface of the 2315 Dexela Detector from the Varex Company

```
import DexelaPy
import msvcrt
import time
import weakref
import threading
import os

class CBData():
    def __init__(self, callbackString, callbackFloat, callbackInt):
        self.callbackString = callbackString
        self.callbackFloat = callbackFloat
        self.callbackInt = callbackInt

class DexDetector():
    """
        A class for Dexela 2315 Detector
    """



    """init"""
    def __init__(self, mypath):

        print("Scanning to see how many devices are present...")
        scanner = DexelaPy.BusScannerPy()

        count = scanner.EnumerateDevices()
        print(count)

        for i in range(0,count):
            info = scanner.GetDevice(i)
            self.detector = DexelaPy.DexelaDetectorPy(info)
            print("Acquiring single image from detector with serial
    number: %d" % info.serialNum)
```

```
        expMode = DexelaPy.ExposureModes.Expose_and_read
        binFmt = DexelaPy.bins.x11
        wellMode = DexelaPy.FullWellModes.High
        trigger = DexelaPy.ExposureTriggerSource.Internal_Software
        exposureTime = 250
        img = DexelaPy.DexImagePy()
        self.detector.OpenBoard()
        w = self.detector.GetBufferXdim()
        h = self.detector.GetBufferYdim()
        print("Initializing detector settings...")
        self.detector.SetFullWellMode(wellMode)
        self.detector.SetExposureTime(exposureTime)
        self.detector.SetBinningMode(binFmt)
        self.detector.SetTriggerSource(trigger)
        self.detector.SetExposureMode(expMode)
        self.path = mypath

        model = self.detector.GetModelNumber()
        print("exposureTime: %d, Binning: %s,wellMode: %s, trigger: %s," %(exposureTime, binFmt, wellMode, trigger))
        print("expMode: %s" %(expMode))




    """set functions"""

    """set exposure modes"""
    def set_expose_and_read(self):
        expMode = DexelaPy.ExposureModes.Expose_and_read
        self.detector.SetExposureMode(expMode)
        print("expMode: %s" %(expMode))
        return

    def set_sequence_exposure(self):
        expMode = DexelaPy.ExposureModes.Sequence_Exposure
        self.detector.SetExposureMode(expMode)
        print("expMode: %s" %(expMode))
        return

    def set_frame_rate_mode(self):
        expMode = DexelaPy.ExposureModes.Frame_Rate_exposure
        self.detector.SetExposureMode(expMode)
        print("expMode: %s" %(expMode))
        return


    """set Binning"""
```

```
    def set_binFmt_x11(self):

        binFmt = DexelaPy.bins.x11
        self.detector.SetBinningMode(binFmt)
        print("Binning: %s" %(binFmt))
        return

    def set_binFmt_x22(self):
        binFmt = DexelaPy.bins.x22
        self.detector.SetBinningMode(binFmt)
        print("Binning: %s" %(binFmt))
        return

    def set_binFmt_x44(self):
        binFmt = DexelaPy.bins.x44
        self.detector.SetBinningMode(binFmt)
        print("Binning: %s" %(binFmt))
        return

    """set wellMode"""

    def set_well_mode_high(self):
        wellMode = DexelaPy.FullWellModes.High
        self.detector.SetFullWellMode(wellMode)
        print("wellMode: %s" %(wellMode))
        return



    def set_well_mode_low(self):
        wellMode = DexelaPy.FullWellModes.Low
        self.detector.SetFullWellMode(wellMode)
        print("wellMode: %s" %(wellMode))
        return

    """set trigger"""

    def set_trigger_internal_software(self):
        trigger = DexelaPy.ExposureTriggerSource.Internal_Software
        self.detector.SetTriggerSource(trigger)
        print("trigger: %s" %(trigger))
        return

    def set_ext_neg_edge_trig(self):
        trigger = DexelaPy.ExposureTriggerSource.Ext_neg_edge_trig
        self.detector.SetTriggerSource(trigger)
        print("trigger: %s" %(trigger))
        return

    def set_ext_duration_trig(self):
        trigger = DexelaPy.ExposureTriggerSource.Ext_Duration_Trig
        self.detector.SetTriggerSource(trigger)
        print("trigger: %s" %(trigger))
```

```
        return

    """set exposure time"""

    def set_exposure_time(self,time):
        exposureTime = time
        self.detector.SetExposureTime(exposureTime)
        if(time > 37.77):
            print("exposure time: %s" %(exposureTime))
        else:
            print('exposure time too small, min : 37.77')
            print('exposure time is set to %d' %(self.detector.
GetExposureTime()))
        return

    """set CL interface power"""

    def set_power_CL_interface_On(self):
        self.detector.PowerCLInterface(True)
        print("Power is on")
        return

    def set_power_CL_interface_Off(self):
        self.detector.PowerCLInterface(False)
        print("Power is off")
        return

    """ set Generator toggling"""


    def start_generator_toggling(self):
        self.detector.ToggleGenerator(True)
        print("Generator started toggling")
        return

    def stopp_generator_toggling(self):
        self.detector.ToggleGenerator(False)
        print("Generator stopped toggling")
        return

    """set readout mode"""

    def set_readout_mode_cont(self):
        mode = DexelaPy.ReadoutModes.ContinuousReadout
        self.detector.SetReadoutMode(mode)
        print("readout mode: %s" %(mode))
        return

    def set_readout_mode_idle(self):
        mode = DexelaPy.ReadoutModes.IdleMode
        self.detector.SetReadoutMode(mode)
        print("readout mode: %s" %(mode))
        return
```

```


    """get functions"""

    """get binning"""

    def get_binning(self):
        binning = self.detector.GetBinningMode()
        print("binning: %s"%(binning))
        return binning

    """get Bufferdimension"""

    def get_bufferdim(self):
        w = self.detector.GetBufferXdim()
        h = self.detector.GetBufferYdim()
        print("h (y-axis): %d, w(x-axis): %d" %(h,w))
        return '%d %d' %(h,w)

    """get exposure time"""

    def get_exposure_time(self):
        exposureTime = self.detector.GetExposureTime()
        print("exposure time: %s" %(exposureTime))
        return exposureTime

    def get_trigger_source(self):
        triggersource = self.detector.GetTriggerSource()
        print("trigger source: %s"%(triggersource))
        return triggersource

    """get readout mode"""

    def get_readout_mode(self):
        mode = self.detector.GetReadoutMode()
        print("readout mode: %s" %(mode))
        return mode

    """get exposure mode"""

    def get_exposure_mode(self):
        expMode = self.detector.GetExposureMode()
        print("expMode: %s" %(expMode))
        return expMode

    """get well mode"""
    def get_well_mode(self):
        wellMode = self.detector.GetFullWellMode()

        print("wellMode: %s" %(wellMode))
        return wellMode


```

```


    """Aquire functions"""

    def AcquireSingleImage(self):
        "start acquisition for a single image"
        exposureTime = int(self.detector.GetExposureTime())
        os.chdir(self.path)
        img = DexelaPy.DexImagePy()
        try:
            self.detector.Snap(1, exposureTime+1000)

            print("Grabbed Image!")
            self.detector.ReadBuffer(1,img);


            img.UnscrambleImage()


            filename = 'Image1_%dx%d.smv' % (img.GetImageXdim(),img.GetImageYdim())
            img.WriteImage(filename)

            print("Image successfully saved!")
            #self.detector.CloseBoard()
        except DexelaPy.DexelaExceptionPy as ex  :
                print("Exception Occurred!")
                print("Description: %s" % ex)
                DexException = ex.DexelaException
                print("Function: %s" % DexException.GetFunctionName())
                return 'failed to grab an Image! No solution for this problem... maybe next time'
        return  ("Image successfully saved! filename: %s" %(filename))

    def AcquirePulseGeneratorSequence(self):
        img = DexelaPy.DexImagePy()
        exposureTime = int(self.detector.GetExposureTime())
        os.chdir(self.path)


        self.detector.EnablePulseGenerator()
        self.detector.GoLiveSeq()

        self.detector.ToggleGenerator(True)
        counter = 0

        while counter < 10:
            try:
                self.detector.WaitImage(exposureTime + 1000)
                counter += 1
                buf = self.detector.GetCapturedBuffer()

                self.detector.ReadBuffer(buf,img,img.GetImageDepth())
```

```
            except DexelaPy.DexelaExceptionPy as ex  :
                print("Exception Occurred!")
                print("Description: %s" % ex)
                DexException = ex.DexelaException
                print("Function: %s" % DexException.GetFunctionName())

                print('failed to grab an Image! No solution for this problem... maybe next time')
                return 'failed to grab an Image! No solution for this problem... maybe next time'
            print("Grabbed Image %d!" % (counter))
            time.sleep(1)

        self.detector.ToggleGenerator(False)
        self.detector.DisablePulseGenerator()

        img.UnscrambleImage()

        print("Images Successfully Grabbed!")

        filename = 'PulseGenImages_%dx%dx%d.smv' % (img.GetImageXdim(), img.GetImageYdim(),img.GetImageDepth())
        img.WriteImage(filename)

        print("Images successfully saved!")

        if self.detector.IsLive() :
            self.detector.GoUnLive()



        return ("Image successfully saved! filename: %s" %(filename))

    def AcquireSequenceMode(self, gapTime = 0, exposures = 5):
        os.chdir(self.path)
        self.detector.CloseBoard()
        self.detector.OpenBoard(exposures)
        self.detector.SetNumOfExposures(exposures)

        self.detector.SetGapTime(gapTime)

        exposureTime = self.detector.GetExposureTime()

        if gapTime == 0:
            expMode = DexelaPy.ExposureModes.Sequence_Exposure
        else:
            expMode = DexelaPy.ExposureModes.Frame_Rate_exposure

        img = DexelaPy.DexImagePy()
        self.detector.GoLiveSeq(0,exposures-1,exposures)
        startCount = self.detector.GetFieldCount()
        count = startCount

```

```
        self.detector.SoftwareTrigger()

        start_time = time.time()
        print((time.time() - start_time))
        while count < startCount+exposures+1 and (exposureTime * exposures) > (time.time() - start_time)* 1000:
            count = self.detector.GetFieldCount()
            print("field count: %d" % count)
            time.sleep(0.1)

        if(count < startCount+exposures+1):
            return "couldn't grab an Image an error occured! tyr again maybe it works now"

        for i in range (0,exposures):
            self.detector.ReadBuffer(i,img,i)

        img.UnscrambleImage()

        print("Sequence Successfully Grabbed!")

        filename = 'ImageSequence_%dx%dx%d.smv' %(img.GetImageXdim(),img.GetImageYdim(),exposures)
        img.WriteImage(filename)

        print("Sequence successfully saved!")

        if self.detector.IsLive() :
            self.detector.GoUnLive()



        return ("Image successfully saved! filename: %s" %(filename))

    def AquireImageCallback(self, imCnt = 0):
        os.chdir(self.path)
        #go_further = False
        start_time = time.time()
        #done = False
        #times = 0
        def myCallback(fc, buf, detRef):
            #times = time + 1
            det = detRef()
            #print(det)
            cbData = det.GetCallbackData()
            #print(cbData)
            print("Callback message: %s. float: %f. int: %d." % (cbData.callbackString,cbData.callbackFloat,cbData.callbackInt))
            print("Image: %d grabbed! Image is in buffer: %d. Model Number: %d" % (fc,buf,det.GetModelNumber()))



```

```

        img = DexelaPy.DexImagePy()
        exposureTime = int(self.detector.GetExposureTime())

        self.detector.EnablePulseGenerator()

        cbData = CBData("My callback message!",3.14,2015)
        self.detector.SetCallbackData(cbData)
        self.detector.SetCallback(myCallback, weakref.ref(self.detector))

        self.detector.GoLiveSeq()

        print("Press any key to terminate acquisition")
        self.detector.ToggleGenerator(True)

        while time.time() - start_time < 8.90799999237:
            self.detector.CheckForCallbackError()
            self.detector.CheckForLiveError()


        self.detector.ToggleGenerator(False)
        self.detector.DisablePulseGenerator()

        if self.detector.IsLive() :
            self.detector.GoUnLive()

        time.sleep((exposureTime/1000)+0.1)

        self.detector.StopCallback()

        self.detector.CloseBoard()

        return ("Callback finished, 30 Images grabbed and stored in
    buffer.")
```

6.4 The Client Script

```
import zmq
import sys

""" send function to communicate with detector """

def send(string):




    port = "5556"

    if len(sys.argv) > 2:
        port1 =  sys.argv[2]
        int(port1)
```

```

    context = zmq.Context()
    print ("Connecting to detector via server...")
    socket = context.socket(zmq.REQ)
    socket.connect ("tcp://localhost:%s" % port)
    if len(sys.argv) > 2:
        socket.connect ("tcp://localhost:%s" % port1)


    request = 1
    socket.send_string (string)
    message = socket.recv()
    print ("Received reply ", request, "message: ", message, )
    request +=1

    return message



class TranslateDet():

    def __init__(self, path):

        send('-1 %s' %(path))

        return

    def new_method(self, path):
        self.path = path

    """ set functions"""

    def set_exposure_time(self,time):
        msg = send('set_exposure_time %s' %(time))
        if(time > 37.77):
                print("set exposure time to %s" %(time))
        else:
            print('exposure time too small, min : 37.77')
            print('exposure time is set to 37.7700004578')
        return

    def set_expose_and_read(self):
        send('expose_and_read')
        print('set exposure mode to Expose_and_read')
        return

    def set_sequence_exposure(self):
        send('set_sequence_exposure')
        print('set exposure mode to Sequence_Exposure')
        return

    def set_frame_rate_mode(self):
        send('set_frame_rate_mode')
```

```
        print('set exposure mode to Frame_Rate_exposure')
        return

    def set_binFmt_x11(self):
        send('set_binFmt_x11')
        print('set binning to x11')
        return

    def set_binFmt_x22(self):
        send('set_binFmt_x22')
        print('set binning to x22')
        return

    def set_binFmt_x44(self):
        send('set_binFmt_x44')
        print('set binning to x44')
        return

    def set_well_mode_high(self):
        send('set_well_mode_high')
        print('set wellmode to High')
        return

    def set_well_mode_low(self):
        send('set_well_mode_low')
        print('set wellmode to Low')
        return

    def set_trigger_internal_software(self):
        send('set_trigger_internal_software')
        print('set triggersource to Internal_Software')
        return

    def set_ext_neg_edge_trig(self):
        send('set_ext_neg_edge_trig')
        print('set triggersource to Ext_neg_edge_trig')
        return

    def set_ext_duration_trig(self):
        send('set_ext_duration_trig')
        print('set triggersource to Ext_Duration_Trig')
        return

    def set_power_CL_interface_On(self):
        send('set_power_CL_interface_On')
        print('turned the CL interface power on')
        return

    def set_power_CL_interface_Off(self):
        send('set_power_CL_interface_Off')
        print('turned the CL interface power off')
        return

```

```
    def start_generator_toggling(self):
        send('start_generator_toggling')
        print('generator started toggling')
        return

    def stopp_generator_toggling(self):
        send('stopp_generator_toggling')
        print('generator stopped toggling')
        return

    def set_readout_mode_cont(self):
        send('set_readout_mode_cont')
        print('set readout mode to ContinuousReadout')
        return

    def set_readout_mode_idle(self):
        send('set_readout_mode_idle')
        print('set readout mode to IdleMode')
        return

    """get functions"""

    def get_binning(self):
        send('get_binning')
        print('binning : %s' %(s))
        return

    def get_bufferdim(self):
        send('get_bufferdim')
        return

    def get_exposure_time(self):
        send('get_exposure_time')
        return

    def get_trigger_source(self):
        send('get_trigger_source')
        return

    def get_readout_mode(self):
        send('get_readout_mode')
        return

    def get_exposure_mode(self):
        send('get_exposure_mode')
        return

    def get_well_mode(self):
        send('get_well_mode')
        return

    """ Image Acquisition functions"""

```

```
    def AcquireSingleImage(self):
        send('AcquireSingleImage')
        return

    def AcquirePulseGeneratorSequence (self):
        send('AcquirePulseGeneratorSequence')
        return

    def AcquireSequenceMode(self,gaptime = 0, exposures = 5):
        send('AcquireSequenceMode %s %s' %(gaptime, exposures))
        return

    def AquireImageCallback(self,imCnt = 0):

        send('AquireImageCallback %s' %(imCnt))
        return
```

6.5 The Server Script

```

import zmq
from time import sleep
import sys
from DetClass import DexDetector


port = "5556"
if len(sys.argv) > 1:
    port =  sys.argv[1]
    int(port)

context = zmq.Context()

socket = context.socket(zmq.REP)
socket.bind("tcp://*:%s" % port)





while(True):
    init = False
    msg = ''

    while (msg == ''):
        msg = socket.recv()
        message = msg.split()
    message = msg.split()
    print("Received request: ", message)

    reply = False # to get the values of the get-functions

```

```


    if (message[0] == '-1'):
        path = message[1]
        det = DexDetector(path)
        a = det.get_exposure_mode()
        b = det.get_exposure_time()
        c = det.get_binning()
        d = det.get_well_mode()
        e = det.get_trigger_source()
        task = 'init'
        info = ['exposuremode : %s' %(a),'exposuretime : %s' %(b) ,'
    binning : %s' %(c),'wellmode : %s' %(d),'triggersource : %s' %(e)]
        init = True

    """ set functions"""

    if(message[0] == 'set_exposure_time'):
        a = det.set_exposure_time(int(message[1]))


    if(message[0] == 'set_expose_and_read'):
        det.set_expose_and_read()

    if(message[0] == 'set_sequence_exposure'):
        det.set_sequence_exposure()

    if(message[0] == 'set_frame_rate_mode'):
        det.set_frame_rate_mode()

    if(message[0] == 'set_binFmt_x11'):
        det.set_binFmt_x11()

    if(message[0] == 'set_binFmt_x22'):
        det.set_binFmt_x22()

    if(message[0] == 'set_binFmt_x44'):
        det.set_binFmt_x44()

    if(message[0] == 'set_well_mode_high'):
        det.set_well_mode_high()

    if(message[0] == 'set_well_mode_low'):
        det.set_well_mode_low()

    if(message[0] == 'set_trigger_internal_software'):
        det.set_trigger_internal_software()

    if(message[0] == 'set_ext_neg_edge_trig'):
        det.set_ext_neg_edge_trig()

    if(message[0] == 'set_ext_duration_trig'):
        det.set_ext_duration_trig()
```

```

    if(message[0] == 'set_power_CL_interface_On'):
        det.set_power_CL_interface_On()

    if(message[0] == 'set_power_CL_interface_Off'):
        det.set_power_CL_interface_Off()

    if(message[0] == 'start_generator_toggling'):
        det.start_generator_toggling()

    if(message[0] == 'stopp_generator_toggling'):
        det.stopp_generator_toggling()

    if(message[0] == 'set_readout_mode_cont'):
        det.set_readout_mode_cont()

    if(message[0] == 'set_readout_mode_idle'):
        det.set_readout_mode_idle()

    """get functoions"""

    if(message[0] == 'get_binning'):
        s = ('binning: %s' %(det.get_binning()))
        reply = True

    if(message[0] == 'get_bufferdim'):
        info = det.get_bufferdim()
        prepare = info.split()

        s = ('bufferdim (h,w): (%s,%s)' %(prepare[0], prepare[1]))
        reply = True

    if(message[0] == 'get_exposure_time'):
        s = ('exposure time: %s' %(det.get_exposure_time()))
        reply = True

    if(message[0] == 'get_trigger_source'):
        s = ('trigger source: %s' %(det.get_trigger_source()))
        reply = True

    if(message[0] == 'get_readout_mode'):
        s = ('readout mode: %s' %(det.get_readout_mode()))
        reply = True

    if(message[0] == 'get_exposure_mode'):
        s = ('exposure mode: %s' %(det.get_exposure_mode()))
        reply = True

    if(message[0] == 'get_well_mode'):
        s = ('wellmode: %s' %(det.get_well_mode()))
        reply = True

    """acquire Images functions"""
```

```

if(message[0] == 'AcquireSingleImage'):
    s = (det.AcquireSingleImage())
    reply = True

if(message[0] == 'AcquirePulseGeneratorSequence'):
    s = (det.AcquirePulseGeneratorSequence())
    reply = True

if(message[0] == 'AcquireSequenceMode'):
    s = (det.AcquireSequenceMode(int(message[1]), int(message[2])))
    reply = True

if(message[0] == 'AquireImageCallback'):
    s = (det.AquireImageCallback(message[1]))
    reply = True



sleep (1)
if (init == False):
    if (reply == True):
        socket.send("task completed, connected port : %s, %s " % (port, s))
    else:
        socket.send("task completed, connected port : %s" % (port))



else:
    socket.send("%s completed, connected port : %s, info: %s" % (task, port, info))
```

6.6 Links of Python Standard Modules

Modules:	Links:
os	https://docs.python.org/3/library/os.html
numpy	https://numpy.org/doc/stable/reference/
time	https://docs.python.org/3/library/time.html
datetime	https://docs.python.org/3/library/datetime.html
yaml	https://pyyaml.org/wiki/PyYAMLDocumentation
h5py	https://buildmedia.readthedocs.org/media/pdf/h5py/latest/h5py.pdf
threading	https://docs.python.org/3/library/threading.html
clororedlogs	https://coloredlogs.readthedocs.io/en/latest/api.html
logging	https://docs.python.org/3/library/logging.html
subprocess	https://docs.python.org/3/library/subprocess.html
IPython	https://ipython.org/ipython-doc/3/index.html
argparse	https://docs.python.org/3/library/argparse.html
msvcrt	https://docs.python.org/3/library/msvcrt.html
weakref	https://docs.python.org/3/library/weakref.html
zmq	https://zguide.zeromq.org/
sys	https://docs.python.org/3/library/sys.html

Bibliography

[1] Carl A Carlsson and Gudrun Alm Alm Carlsson. *Basic physics of X-ray imaging*. Linköping University Electronic Press, 1973.

[2] Andreas Maier et al. "Medical imaging systems: An introductory guide". In: (2018).

[3] Thomas Thüring. "Compact X-ray grating interferometry for phase and dark-field computed tomography in the diagnostic energy range". en. PhD thesis. Zürich: ETH Zurich, 2013. DOI: 10.3929/ethz-a-010008147.

[4] Wikipedia. *Talbot effect — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Talbot\%20effect&oldid=1024983109. [Online; accessed 01-June-2021]. 2021.