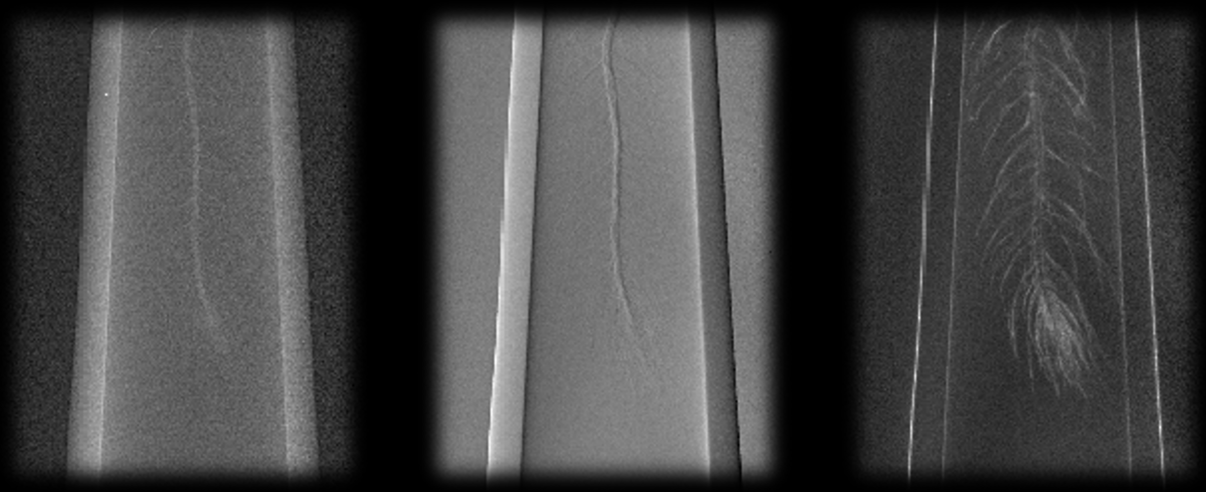
ETH zürich

PSI

Bachelor Thesis

Institute for Biomedical Engineering, ETH Zurich



System Optimization and Software Design in Grating-Interferometry X-Ray Imaging Laboratories

Lionel Peer

Supervision: Prof. Dr. Marco Stampanoni

Advisor: Dr. Maxim Polikarpov

Abstract

The X-Ray Tomography Group of the Biomedical Engineering Institute at ETH Zurich operates an X-ray grating interferometer located at the Paul Scherrer Institut. This setup is extensively used for laboratory X-ray imaging of samples for materials sciences as well as biological and medical ones. As part of ongoing research activities at this setup, I was granted the responsibility to design high-level operation and data processing software in the context of my Bachelor's thesis. Before my arrival the setup had been operated with low-level software and in order to allow for hardware independent operation and reduce acquisition times, a high level software package was created. The written operation software allows for a quick change of hardware components and more can be easily added. In the case of one detector, a significant reduction in overhead time was achieved. The high-level approach ensures the safe operation by group members with very little experience, making it available as a research tool for larger parts of the group. Further, a collection of frequently used data processing functions was created and their capabilities extended, such that they can handle different types of inputs. Tomographic reconstruction had so far only been done using parallel geometries. This functionality was extended such that it now considers the cone shaped geometry of the real system and makes use of the highly parallelizable architecture of GPUs, while operational simplicity for the user was kept at the same level.

Acknowledgments

My time at TOMCAT was an intense and very interesting one. I got to first hand experience day-to-day laboratory operations and was included in every single process. I would like to thank Prof. Dr. Marco Stampanoni, my supervisor and the group leader of TOMCAT, for making my first research experience possible and introducing me to the world of X-ray imaging.

A special thanks goes to Dr. Maxim Polikarpov who was my direct advisor during these 14 weeks. He gave me a lot of liberty with my research and I was always welcome to introduce new ideas. This gave me the chance to get a broad view of not only grating interferometry but X-ray imaging and computed tomography as a whole.

Further I would like to thank Dr. Michał Rawlik, the leader of our neighboring group that focuses on breast CT, and Simon Spindler, who both gave me great advice on simulations and tomographic reconstruction. I was always welcome to ask questions, be it during the lunch breaks or via email.

Another opportunity that I was given over the course of this thesis, was the chance to test the performance of newly produced high aspect ratio absorption gratings. For this I would like to thank Dr. Lucia Romano, Zhitian Shi and Dr. Konstantins Jefimovs with whom I got to spend a great time in the lab.

Finally I would like to thank Lars Lenherr, who wrote his Bachelor's thesis during the same time in Michał's group, for the scientific exchange that we had, both about the setups and the Bachelor's thesis as such.

Contents

1 Introduction

X-ray imaging started with the detection of X-rays by W. Röntgen and the subsequent publishment of the first X-ray image in 1895. Soon, single projections were not enough anymore and driven by the motivation to get an insight into the rib cage, without the ribs interfering, axial tomography was invented in the late 1930s and 40s. Those axial tomographs were purely mechanical and only in the 1960s, with the rise of the digital technology, the idea of a computer-based back projection arose. This led to the creation of CT by A. Cormack and G. Hounsfield, for which they were awarded the Nobel prize in 1979. In parallel, phase contrast imaging was developed for the visible light spectrum and is a well established technology for microscopy [5]. Adaptation using X-ray was slow due to challenges in the microfabrication process of X-ray optical elements. Nevertheless, interferometry for synchrotrons was explored and successfully deployed during the 1990s [15] and early 2000s. Subsequently, in 2006, the development of a Talbot-Lau interferometer setup with a conventional X-ray tube was first published [11]. This opened the doors for future medical applications since the better contrast in soft tissues has great potential, for example in the detection of breast cancer.

Such a system, has been installed at the X-ray tomography group of Prof. Dr. M. Stampanoni at the Paul Scherrer Institut. It offers high spatial resolution (10 µm) in a field of view of 5-15 cm. So far, typical acquisition times for tomographic scans ranged from 15 hours to 3 days, depending on the setup geometry and the number of projections. The motivation behind this thesis was to bring the acquisition times down by streamlining the process, make it universal for different hardware components and provide a software infrastructure for quick reconstruction of the acquired projections. Eventually this setup should be able to acquire and process high resolution phase contrast tomography, for example of histopathological samples, in a time frame of 2-3 h. The high-level approach that the software should follow would also allow this setup to be used as a research tool for other members of the group. After getting comfortable with the operation of the single components, points of optimization can be identified with which this high-level framework should be designed. By optimizing the interaction between the hardware components, acquisition times should be brought down. Tomographic reconstruction should be enhanced by introducing a simplified access to powerful GPU optimized reconstruction algorithms. Those algorithms are not only faster, but they also take the diverging X-ray beam into account, therefore closer resembling the real setup.

The first part of the thesis will cover the theory behind phase contrast X-ray imaging using interferometers and the reconstruction of acquired tomographic data sets. The second part covers the different hardware components and improvements made in this respect. Further, the last two parts, will present the software packages for controlling the setup and for the reconstruction. The emphasis will lie on improvements that simplify and speed up operation as well as the processing of data.

2 Theory

The goal of this chapter is to lay out the mathematical and physical foundation behind X-ray interferometry used for phase contrast imaging and tomographic acquisition and reconstruction. Understanding this is of importance when looking at the operation and reconstruction software packages and for the discussion of experimental results.

2.1. X-Rays and Matter

X-rays are photons with an Energy of $E = \frac{hc}{\lambda}$, where h is Planck's constant, which is experimentally derived, and λ is the wavelength of the corresponding photon. While visible light has wavelengths of 400 nm (violet) to 700 nm (red), the energy range of diagnostic X-rays, which is the primary focus of this thesis, lies in between 10 keV and 150 keV and therefore the wavelengths are significantly smaller (0.12 nm to 0.008 nm). The dominating effects of matter interaction at this energy range are photoelectric absorption, Compton scattering & Rayleigh scattering. Photoelectric absorption is a contributor to the attenuation or decrease in intensity of the incoming X-ray beam, Compton scattering both attenuates and scatters, while Rayleigh only leads to scattering of the photons [19, p. 8]. X-rays follow the wave-particle duality and interferometric imaging makes use of their wave nature with the help of diffraction and refraction.

In order to understand the X-ray and matter interaction it is helpful to start with plane waves propagating in a medium, mathematically described by the following equation:

$$E(r,t) = Re\{\underline{E}(r)e^{-i\omega t}\} \tag{2.1}$$

$\underline{E}$ is the complex field amplitude of the real time dependent field E and $\underline{E}$ is again described with the help of field vector $\underline{E_0}$ and wave vector r, determining polarization and propagation directions.

$$\underline{E}(r) = \underline{E_0}e^{\pm ik\cdot r} \tag{2.2}$$

Without loss of generality, but for reasons of simplicity, it is helpful to constrain calculations to plane waves travelling in z-direction, incident to an object in space. The underscore of the vectorial field quantities will be dropped from now on and $k_z = 2\pi/\lambda$ simplifies to a scalar quantity, the wave number.

$$E_{in}(z) = E_0 e^{ik_z z} \tag{2.3}$$

As described by [19, p. 11], the total interaction of such an X-ray wave travelling through a sample between $z = 0$ and $z = z_0$ can be described by a line integral along the z-axis that sums all interaction. This results in the following relation between incoming and outgoing waves.

$$E_{out} = E_{in} e^{ik \int_0^{z_0} n(z)dz} \tag{2.4}$$

$n(z)$ is the spatial distribution of the complex coefficient of refraction $n = 1 - \delta + i\beta$, where β is related to the attenuation properties of the material and δ to the phase shifting properties. Putting this into Eq. 2.4 yields a term consisting of a propagative part, a phase shifting part and an attenuation part.

$$E_{out} = E_{in} e^{ikz_0} e^{-ik \int_0^{z_0} \delta(z)dz} e^{-k \int_0^{z_0} \beta(z)dz} \tag{2.5}$$

Deriving attenuation A (influence on the amplitude) and phase difference from Eq. 2.5, the resulting equations are the following [19, p. 11]:

$$A = 1 - \left(\frac{\mid E_{out} \mid}{\mid E_{in} e^{ikz_0} \mid} \right)^2 \tag{2.6a}$$

$$\phi = arg(E_{out}) - arg(E_{in} e^{ikz_0}) \tag{2.6b}$$

As can be seen from the equations, the incoming field (E_{in}) that propagated through the sample space without the sample present ($E_{in} e^{ikz_0}$), must be measured as well. From now on, this will be the so called *flat-field* image. X-ray detectors use different technologies to measure the intensity of the beam, therefore determining A from Eq. 2.6a is straightforward. Accessing the information hidden in the phase of the outgoing wave is more complicated, and its process will be explained in detail in the next sections.

2.2. Talbot Effect

In 1836, Henry Fox Talbot made the observation that light, incident to a grating, will reproduce the exact structure of the grating, if a screen is placed at certain distances behind the grating.

In 1881 Lord Rayleigh was the first to quantify these distances as:

$$z = m \cdot \frac{d^2}{\lambda} \tag{2.7}$$

For these distances, d is the period of the grating, λ the wavelength of the incident particle and m is an arbitrary integer value. The actual reproduction happens at double these distances, $z_T = m \cdot 2d^2/\lambda$, the image at d^2/λ is shifted to the side by half the grating period [12, p. 196]. The value of z_T will from now on be called *Talbot distance* and the integer m will be called the *Talbot order*.

In order to calculate this value, one can imagine a wave incident to the grating in the (x, y) plane at $z = 0$. The grating structure can be imagined as vertical, which corresponds to parallel to the y-axis. The incident wave has an angle of incidence of θ between propagation direction and the (y, z) plane. This angle is of interest since the grating acts on the wave component perpendicular to both the grating structure and the propagation direction, which is the x-axis. As presented by [2] the field behind the grating can be calculated by starting with the projection of the wave vector onto the x axis $k_x = ksin(\theta)$, yielding the wave of interest as $\psi = e^{ik_x x}$ and calculating the wave directly behind the grating as follows:

$$\psi(x, +0) = \psi(x, -0)T(x) = \sum_n A_n e^{i\left(k_x + 2\frac{\pi}{d}n\right)x} \tag{2.8}$$

In this equation, $T(x) = \sum_n A_n e^{i2\frac{\pi}{d}nx}$ is the grating transfer function. It can be observed that the grating adds multiples of $2\pi/d$ to the wave vector k_x, which means that everything so far is perpendicular to the optical axis. The other wave vector component, k_z, can be derived from the total wave vector and the above x-direction component:

$$k_z = \sqrt{k^2 - \left(k_x + 2\frac{\pi}{d}n\right)^2} \tag{2.9}$$

Using paraxial- and Taylor approximations, as shown by [20], this can be simplified to:

$$k_z \approx k - \frac{\left(k_x + 2\frac{\pi}{d}n\right)^2}{2k} \tag{2.10}$$

Adding this to the calculation from above, the complete field behind the grating presents itself as the following:

$$\psi(x, z) = \sum_n A_n e^{i\left(ksin(\theta) + 2\frac{\pi}{d}n\right)x + i\left(k - \frac{\left(ksin(\theta) + 2\frac{\pi}{d}n\right)^2}{2k}\right)z} \tag{2.11}$$

For the *Talbot effect* to happen, spatially coherent waves are required to be incident to the grating, therefore $sin(\theta) = 0$. Also putting in the relation that $k = 2\pi/\lambda$ the field simplifies to:

$$\psi(x, z) = \sum_n A_n e^{i2\frac{\pi}{d}nx} e^{i2\frac{\pi}{\lambda}z} e^{-in^2\frac{\pi\lambda}{d^2}z} \tag{2.12}$$

The second term is not of interest, since it is independent of n and introduces a global phase. It is visible that $e^{-in^2\frac{\pi\lambda}{d^2}z}$ becomes $e^{-i2m\pi\cdot n^2}$ for $z = m \cdot 2d^2/\lambda$, which was the before defined Talbot distance. Since this term's exponent includes an integer multiple of 2π, it is always one at those distances. The following field at Talbot distances is formed, corresponding exactly to the transfer function of the grating:

$$\psi(x, z = m \cdot 2d^2/\lambda) = \sum_n A_n e^{i2\frac{\pi}{d}nx} \tag{2.13}$$

2.3. Fractional Talbot Effect & Moiré Patterns

When using phase gratings, a similar effect can be observed, called *Lohmann images* by Suleski in [16], instead of *Talbot self images*. They appear at fractions of the Talbot distance and are therefore part of a class called *Fractional Talbot Effects*. The main motivation behind using phase gratings instead of absorption gratings is, that they can be manufactured from poorly absorbing material, offering very high efficiency compared to standard gratings, where absorption is necessary for their function. Suleski lists 36 different combinations of duty cycles and phase shift values producing such images, this thesis will only look at π and $\pi/2$ shifting gratings. For a specific wave length, from now on called design energy, and a phase grating that causes a phase shift of $\phi = \pi/2$ at this energy and has duty cycle 50%, this distance is 1/4th of the original Talbot distance or:

$$z_{par,\pi/2} = m \cdot \frac{p_1^2}{2\lambda} \tag{2.14}$$

For a phase shift of π, such an image forms at 1/16th of the Talbot distance, but here the period of the Lohmann image is half the grating period, as also stated by [3].

$$z_{par,\pi} = m \cdot \frac{p_1^2}{8\lambda} \tag{2.15}$$

With the help of geometric magnification, this was the grating specification used in the setup during the course of this thesis. With s being the complete setup length and l the distance between the phase grating and a micro focus source, the periods get magnified by $M = \frac{s}{l}$.

For this reason, the distances were deliberately indexed *par*, because they are only true for a parallel shaped geometry. For a cone shaped incident beam, those maxima of intensity are reached at the following distances, as stated by [4], where l is again the source-phase grating distance:

$$z_{cone} = \frac{l}{l - z_{par}} z_{par} \tag{2.16}$$

Since the goal is to detect changes when a sample is placed in the beam, analyzing shifts in this pattern due to a refractive change is necessary. Unfortunately the pixel size of the detector is significantly bigger than the period of the pattern which makes a direct analysis impossible. In order to tackle this problem, an analyzer grating instead of the detector itself, is placed at such a fractional Talbot distance behind the phase grating. The pattern formed by the phase grating should have the same periodicity as the analyzer and overlap with its structure. A refractive change in the beam will then immediately translate into an intensity change on the detector which follows directly behind this analyzer grating.

2.4. Talbot-Lau Interferometer

In order to reach the needed coherence, a source grating is used that creates individually coherent line sources. Sufficient coherence is a premise for the use of a grating like the the one from section 2.2 or 2.3. From now on this source grating will be called G_0, the phase grating G_1 and the analyzer grating G_2. For these gratings, the periods need to have the following geometry, where l is now the length between G_0 and G_1 and d is the distance between G_1 and G_2 corresponding to the fractional Talbot distance [11]:

$$p_{G_0} = \frac{l}{d} p_{G_2} \tag{2.17}$$

This is called a *Talbot-Lau configuration* and by placing the gratings according to above equation, the incoherence between the single sources works constructively. [22].

The symmetric setup with π-shifting G_1 and the same periods everywhere emerges as one possible configuration for such a Talbot-Lau interferometer. Condition 2.17 is fulfilled and the magnification factor of $M = 2$ increases the period of the Lohmann image at $z_{par,\pi}$ to match the period of G_2. An example of such a symmetric setup with gratings of period $2\,\mu\text{m}$ and π-shifting G_1 is shown in Fig. 2.1.

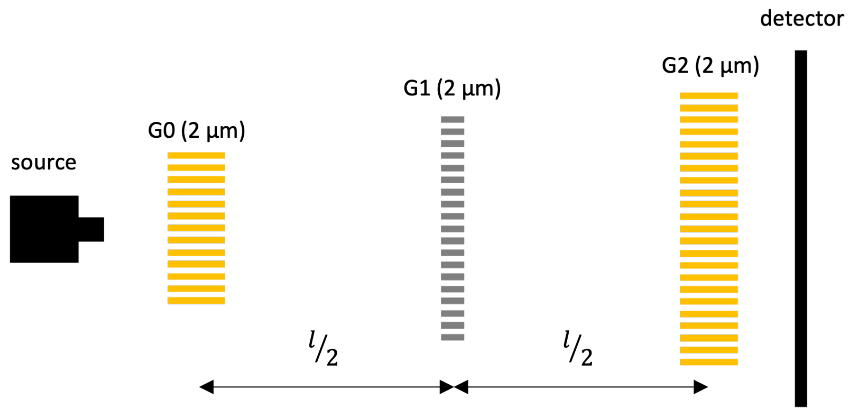


Fig. 2.1.: Symmetric setup with gold filled absorption gratings and π shifting phase grating. Sample placement is between G_0 and G_1

2.5. Image Acquisition, Phase Retrieval & Quality Assessment

Data acquisition is done using a phase stepping technique. The analyzer grating is moved in x-direction, the direction of the grating modulation, over the course of one grating period and an image is taken at every step.

The phase stepping leads to a phase stepping curve in each pixel, from which absorption contrast (amp), differential phase contrast (dpc) and scattering or dark-field contrast (dci) signals can be derived, using Fourier analysis. The values $a_{0,1}$ and ϕ_1 in the formulas below correspond to norm and phase of the first and second Fourier coefficients for either the flat-field phase stepping curve (ref) or the phase stepping curve with the sample (sam). The formula for absorption contrast corresponds to the formula mentioned in Eq. 2.6a. For phase information, the second Fourier coefficient is used (unlike Eq. 2.6b), therefore it is called the *differential* phase contrast image [19, p. 21].

$$amp = 1 - \frac{a_{0,sam}}{a_{0,ref}} \tag{2.18a}$$

$$dpc = \phi_{1,sam} - \phi_{1,ref} \tag{2.18b}$$

$$dci = \frac{a_{1,sam}}{a_{0,sam}} \cdot \frac{a_{0,ref}}{a_{1,ref}} \tag{2.18c}$$

2.5.1. Visibility

Since such an interferometer works with polychromatic sources and not everything gets absorbed in the gold inlays of the absorption gratings, the visibility of the fringes is an important metric to determine its performance.

As demonstrated by [17], the phase stepping curve $I_p(x)$ is a convolution of the optimal interference pattern for coherent sources $I_c(x)$, the source intensity profile at the detector $S'(x)$, and the transmission of the analyzer grating G_2 ($G(x)$). Because of the transmission function, $I_p(x)$ is periodic in the G_2 grating period and the visibility can be calculated using Fourier analysis:

$$V = \frac{I_{p,max}(x) - I_{p,min}(x)}{I_{p,max}(x) + I_{p,min}(x)} = 2\frac{a_1}{a_0} \tag{2.19}$$

Here, $a_{0,1}$ are again the respective Fourier coefficients derived from the phase stepping curve. It is worth noting that assuming G0 and G2 as perfectly absorbing gratings with duty cycle 50%, the visibility would reach 51.6% [17].

2.5.2. Angular Sensitivity

A different metric for measuring the performance of a grating interferometry is the smallest detectable refraction angle. It is dependent on the inter grating distance of G_1 and G_2 (d), the standard deviation (σ_ϕ) of the differential phase contrast image $dpc = \phi_{1,sam} - \phi_{1,ref}$ and the G_2 period, as presented by [18].

$$\alpha_{min} = \frac{p_2}{2\pi d}\frac{l}{l_s}\sigma_\phi \tag{2.20}$$

The distances l and l_s are the distances G_0-G_1 and G_0-sample respectively. Since l_s is in any case the smaller distance than l, the sample should be placed as close as possible to G_1 in order to minimize α_{min}. Longer propagation distances (d) and smaller grating periods are also beneficial in this regard.

2.6. Computerized Tomography & Tomographic Reconstruction

Computerized tomography has revolutionized diagnostic radiology and the 1979 Nobel prize in physiology and medicine was awarded for its development. Many others in the field have followed, for example the 2003 Nobel prize for advancements in magnetic resonance imaging, a related method that also uses the reconstruction from projections [6].

The idea behind such imaging methods is that, given a sufficient amount of information from different angular projections through the object, the cross section of the object can be reconstructed.

2.6.1. Mathematical Basis

Even though reconstruction can be done for any kind of projection image mentioned in 2.5, this section will show it using absorption projections. The formulas hold true for the other images as well.

Assuming an infinitely small slice of the object between $z = 0$ and $z = D$, the attenuation along a line passing through this slice can be described as:

$$m = \int_0^D \mu(x, y) dz \tag{2.21}$$

The value μ is the attenuation coefficient, closely related to β from the complex coefficient of refraction mentioned in 2.1 by $\mu = 2k\beta$, with k being the wave number. The goal is to derive the spatial distribution of $\mu(x, y)$ in the whole slice from this information.

The mathematical basis for this reconstruction had already been provided by Johann Radon in 1917 [6, p. 35].

$$\mu_e(x, y) = -\frac{1}{2\pi^2} \lim_{\epsilon \to 0} \int_\epsilon^\infty \frac{1}{q} \int_0^{2\pi} m_1(x \cos\theta + y \sin\theta + q, \theta)\, d\theta\, dq \tag{2.22}$$

In this formula, m_1 is the partial derivative of the the line integral $m(l = xcos\theta + ysin\theta + q, \theta)$ with respect to l. Looking from above onto the slice like in Fig. 2.2, l is the deviation to the side from the center of rotation and θ is the angle at which the line integral is taken. The analogy in a real setup is the following: a slice represents a pixel row on the detector, l one of these pixels and all l with a certain θ form one projection of the slice. Even though the formula is complicated, it is visible that, given infinitely small l and θ, the attenuation coefficient distribution in the slice could be completely and uniquely reconstructed.

2.6.2. Reconstruction Algorithms

Since in reality the data set is never complete to the point where infinitesimal θ and l are reached, reconstruction is dependent on algorithms that provide reconstruction quality for a finite amount of pixels and acquisition angles. Another important factor besides quality is the computing time to reconstruct such a slice. This is dependent on the algorithm but also on its implementation and the available hardware. The algorithms presented here are back projection-based whereas iterative methods with forward projection will not be explained.

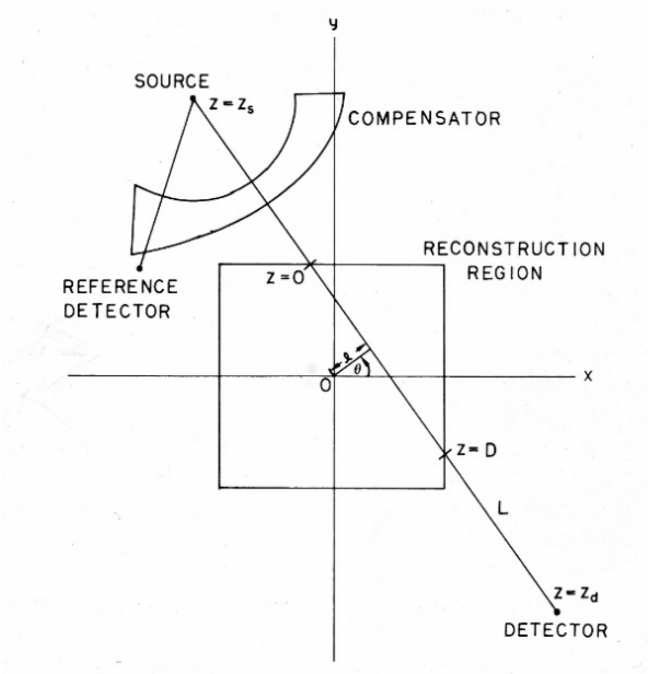


Fig. 2.2.: Computerized Tomography acquisition protocol [7, p. 30]. θ is the angle at which the acquisition happens, l the deviation to the side. The detector measures the intensity of the attenuated beam and together with a flat-field measurement, the attenuation A can be derived.

Fourier Slice Theorem

The two algorithms presented here are based on the Fourier slice theorem which states that projecting a slice of a sample and do a Fourier transform is the same as a 2D Fourier transform of the sample space. Fig. 2.3 shows the 1D transform of the projections in the 2D Fourier space. As can be seen, the sampling gets lower with increasing distance from the origin. This corresponds to the high spatial frequencies, responsible for small details, like sharp edges. Underrepresentation of these frequencies leads to blurry images [13].

Backprojection

Clearly the simplest reconstruction algorithm, it estimates the density (value of the attenuation coefficient μ) at a certain point by assigning every value in the a line the value of the line integral. This is done for all projection angles and the values are summed up [6, p. 125]. The problem stated in the section before is strongly visible with this reconstruction method: The image is very blurry due to the underrepresentation of high spatial frequencies. An example of this is shown in Fig. 2.4. Though unusable in practice, it lays the foundation for the next algorithm.

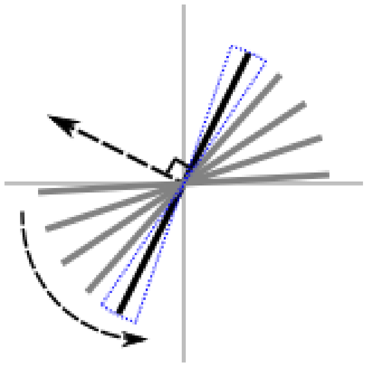
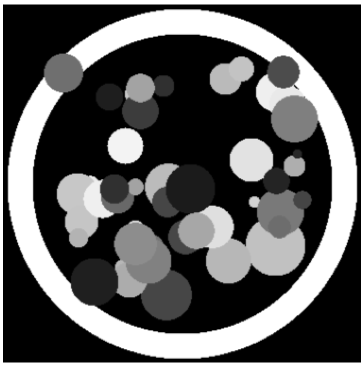
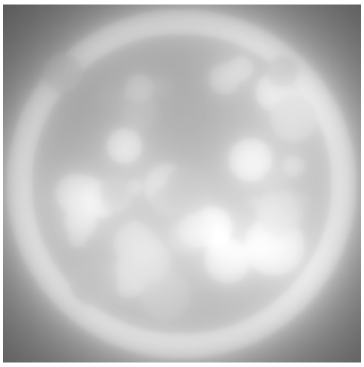


Fig. 2.3.: Fourier slice theorem. Shown is the 1D Fourier transform of the projections (gray and black lines) in the 2D Fourier space. A 2D back transformation would yield the sample space, but the undersampling in the outer regions is well visible (blue) [13].



(a) Original slice through the sample.



(b) Reconstruction of the same slice using BP.

Fig. 2.4.: An example of a backprojection using a phantom. The phantom depicts a capillary with spheres of different radii and density. The image is blurry and no details are visible with this method, making it unusable in practice.

Filtered Back Projection

The simplest approach to solving the problem mentioned in the sections before, is to apply a filter to the 1D Fourier transforms of the projections (gray and black line in Fig. 2.3) before back projecting them [13]. This is to attenuate the low spatial frequencies such that they are equally represented after the back projection. The most commonly used filter is called *Ram-Lak* and is shown in Fig. 2.5. This filter's attenuation is linear with decreasing frequency and many other filters exist. An example of a reconstructed slice using FDK (Feldkamp, Davis & Kress), a 3D extension of the filtered back projection (FBP), will be shown in the Data Processing chapter in Fig. 5.6.

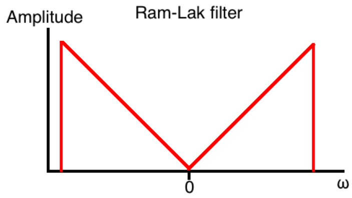


Fig. 2.5.: Ram-Lak filter that attenuates low spatial frequencies (around $\omega = 0$) and lets high spatial frequencies pass [1]. This filter counteracts the underrepresentation of high spatial frequencies when using reconstruction algorithms that are based on the Fourier slice theorem.

3 LAB SETUP

This chapter should give an overview of the different hardware parts of the set up. All of them are mounted on a rail on an optical table such that the distances can be easily adjusted. In order to find points of optimization, the first 2 to 3 weeks of this thesis consisted of learning the following tasks:

1. Calculating and adjusting the setup geometry.
2. Learning how to align gratings by hand and laser as well as the fine-alignment with the motors.
3. Operating the detectors.
4. Combining the hardware in order to do phase stepping scans and determine the performance metrics of the system.

All of this was beneficial to the numerous grating tests that were performed during this time and it also led to the identification of the required capabilities for the operation software, explained in the next chapter.

3.1. Detector

Different detectors were used with this setup, but most frequently the Eiger R 1M from Dectris Ltd., because it provided very good image quality without any further processing. This detector has a photon detection range of 3.5 to 30 keV, a detection threshold can be set, its pixel size is 75 µm and the image size is 1065×1030 pixels, resulting in an active surface of 79.9×77.1 mm [8]. The almost square sensor area proved to be useful when handling samples in pipettes or capillaries that are mostly elongated along the vertical axis.

When creating setups for higher design energies (above 30 keV), a CdTe-based detector from Dectris was used, because its detection threshold could be adjusted to such high energy values.

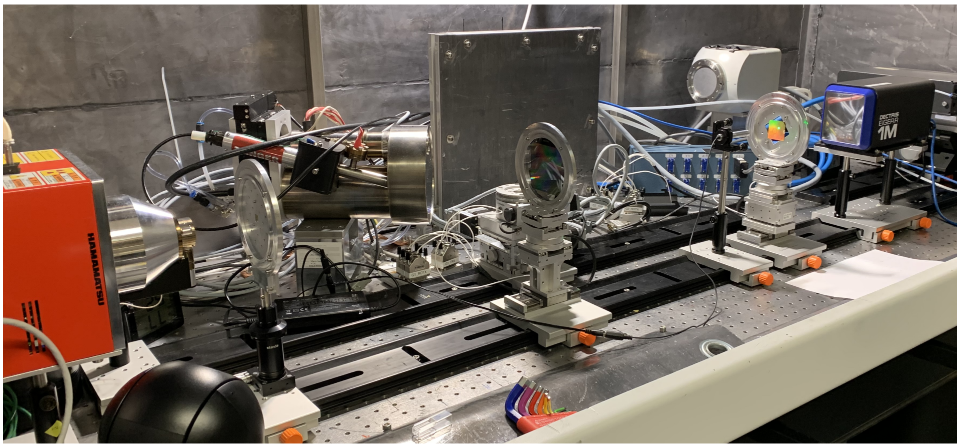


Fig. 3.1.: The setup as it was used during the course of this thesis. On the right the Dectris Eiger R 1M detector, the Hamamatsu L10101 microfocus source on the left and the three gratings on their respective motor towers in between on the rail.

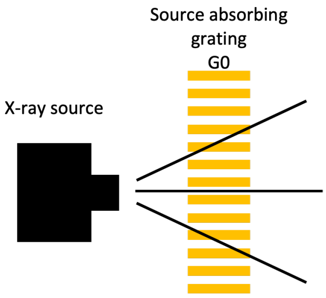
Its pixel size was similar to that of Eiger, but it has a different field of view (256×3094 pixels or 19.2×232 mm) with the longer side along the horizontal axis.

Lastly there was the X-ray sCMOS 16MP from *Photonic Science*. This detector's advantage is that it offers a much smaller pixel size which makes it possible to get even better spatial resolution of the sample.

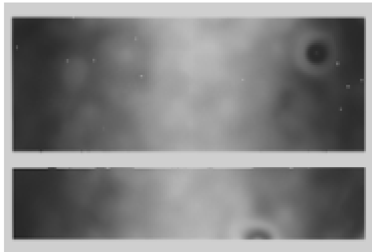
3.2. Gratings

One of the centerpieces of the lab setup are the gratings. They are manufactured from silicon and the absorption gratings are then filled with gold. As explained in 2.4, a Talbot-Lau configuration is used. Important numbers for characterization of these gratings are duty cycle, aspect ratio and phase shifting properties. Duty cycle is defined as the ratio of grating ridge and grating period, aspect ratio as the ratio of trench depth and grating period and the phase shifting properties depend on the used energy and the trench depth. The gratings used during the process of this thesis had aspect ratios of up to 30 and introduced a phase shift of π or $\pi/2$.

As mentioned in 2.5.2, a smaller grating period is beneficial for reaching a smaller minimal detectable angle of refraction, so they should be preferred over gratings of larger period. Apart from the fact that they are more difficult to produce they add a difficulty to the geometry of the setup: As can be seen in Fig. 3.2a, the diverging beam passes easily through the central regions of the gratings but in the outer regions it passes through gold layers and therefore gets partially absorbed. The effect is a limited field of view, as shown in Fig. 3.2b. The easy solution to this problem is to move the source grating further away, where the divergence has less of an influence. This was done during the grating test of the high aspect ratio gratings that is presented in appendix A. The more complex solution would be to bend the gratings with the right radius.



(a) The diverging beam of the X-ray source causes absorption in the outer parts of the grating. This limits the field of view.



(b) The transmission profile of a 2 µm grating. Apart from some inhomogeneities in the right part (circles), it can be seen that transmission is higher in the center part.

Fig. 3.2.: The diverging beam limits the field of view. An easy solution to this is to move the setup further away from the source where the beam shape is less conical.

3.3. Motorized Towers

Basic alignment of the gratings can be done by visual inspection as well as with the help of a laser. Pointing it onto the side where the grating is etched into the silicon wafer, the diffraction pattern can be assessed. In most cases one would want this pattern to be parallel to the earth, which in turn means that the grating structure is vertical to the earth.

The fine-tuning of the grating alignment happens with the help of motorized towers on which the gratings are mounted. These towers are customizable with linear motors and goniometers to reach many degrees of freedom at a very high resolution of motion. Since the sample has to be rotated and moved out of the detector area during the tomographic process, the sample itself is also placed on top of such a motor tower. The motor stages for the gratings were from SmarAct Inc., the one for the sample from HUBER Diffraktionstechnik GmbH & Co. KG.

3.4. X-Ray Source

Two microfocus sources were used with the setup, the L10101 from Hamamatsu Photonics and a prototype from Sigray Inc. The Hamamatsu source has a maximal tube voltage of 100 kV and a maximal current output of 200 µA. The source from Sigray has a significantly higher power output, reaching up to 1200 µA at up to 50 kV. The increased power is important since exposure time is an important factor for increasing the signal-to-noise ratio (SNR). With the introduction of the Sigray source during the time of this thesis and the boost in power that it provides, it should be possible to cut the exposure time sixfold in the future.

Another important feature of the X-ray source are the source sizes, since they influence the coherence. While the Hamamatsu tube has a source size of 10 µm - 15 µm, the Sigray prototype also includes a structured anode that creates an array of line sources, just like the source grating G_0 would. This means that when making use of this structured anode, G_0 is redundant and can be left out.

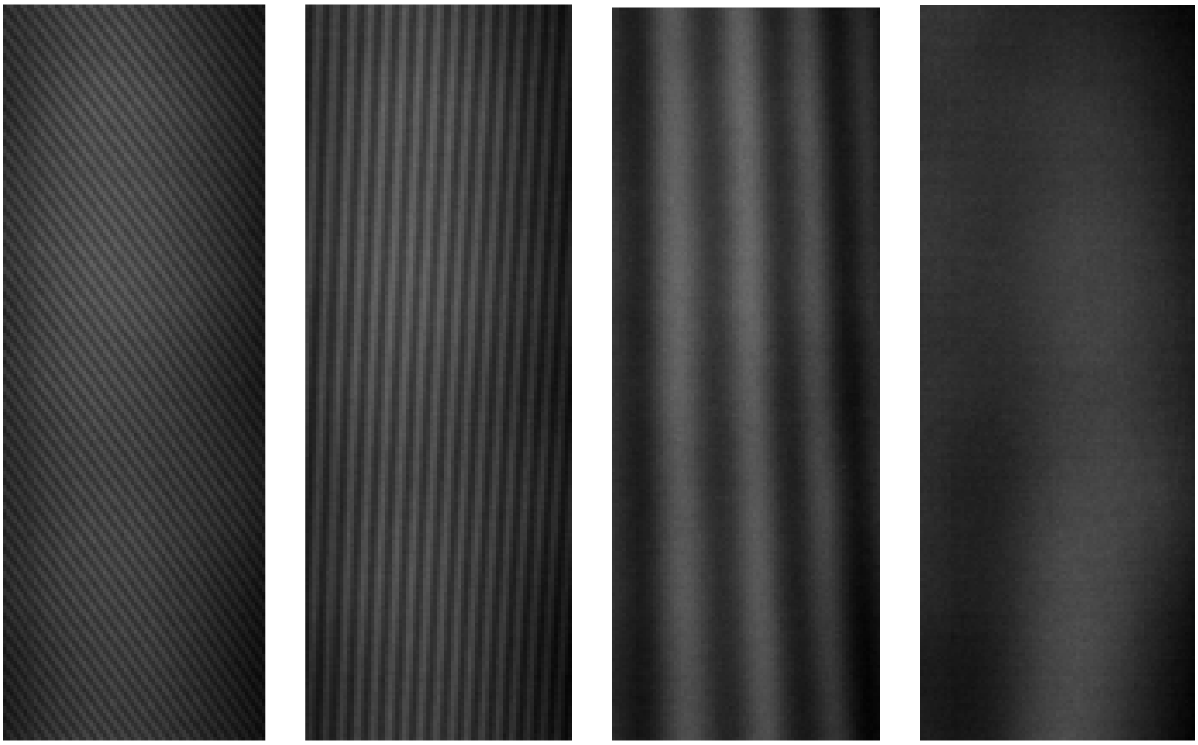
3.5. Placing and Alignment of Components

A geometry in accordance with 2.3 and 2.4 has to be chosen for the desired design energy. Shorter total lengths of the setup have an advantage when it comes to exposure time, longer distances between G_1 and G_2 (d) increase the propagation length of the refracted beam and increase angular sensitivity, shown in 2.5.2.

Basic alignment of the 3 gratings is done using rulers and a laser pointer and G_1 is subsequently aligned with the help of Moiré effects and the motor stage it sits on. Two types of fringes, produced by the overlapping of the reproduction of G_1 with the analyzer G_2, can be observed:

1. Vertical fringes coming from a small deviation around the Talbot distance which is along the z-direction, also corresponding to the optical axis.
2. Horizontal fringes, caused by a rotation of one of the gratings around the z-axis.

The vertical fringe is visible from the beginning if alignment by hand and laser was done carefully. The goal is to make them completely vertical without diverging towards the top or the bottom, and then move G_1 along the optical axis such that they get bigger and eventually disappear. This is the placement where the intensity pattern of G_1 and the grating structure of G_2 are congruent. Such a process is shown in Fig. 3.3. A set of printed gratings on overhead projector sheets proved to be a helpful tool for the interpretation of the observed fringes.



(a) Visible fringes directly after alignment by hand.

(b) Almost vertical fringes by rotating G1 around the z-axis.

(c) Big vertical fringes, indicating that alignment is far advanced.

(d) Fringe pattern has disappeared, gratings are aligned.

Fig. 3.3.: Grating alignment flow. The slight divergence towards the bottom seen in Fig. 3.3c can be corrected by rotating G1 around the x-axis.

4 Operation Software

So far this setup had been operated with the help of a few scripts for operation of the detector as well as the motors, but most operations happened inside of Jupyter Notebooks following a low-level approach. The goals of the software development part were the following:

1. Implement a package that allows for high-level operation of the system.
2. Leave enough flexibility for future hardware components to be integrated in this package.
3. Reach a considerable reduction in acquisition times for the whole tomographic process.

This chapter tries to highlight where optimizations were made in order to reach these goals, what their impact was and why certain parts were left untouched. The full code can be found in the appendix.

4.1. Detector Operation

The detector is one of the key elements of such a setup. As mentioned in the previous chapter, the main detector in use was the Eiger R 1M by Dectris. After gaining experience in operating it with a class that had existed before, the functionality that should be expected from every detector operated with this setup was identified. This led to the creation of the interface `Detector` as seen in the class chart (Fig. 4.1). Separate classes for each detector were created and each class had to implement at least the functionality defined by the interface. This happens with the help of the attributes `client`, which are instances of the low level classes that were used before. This ensures that experienced users still get access to all low-level operations if necessary, with a call of `Eiger.client` in the case of the Eiger detector.

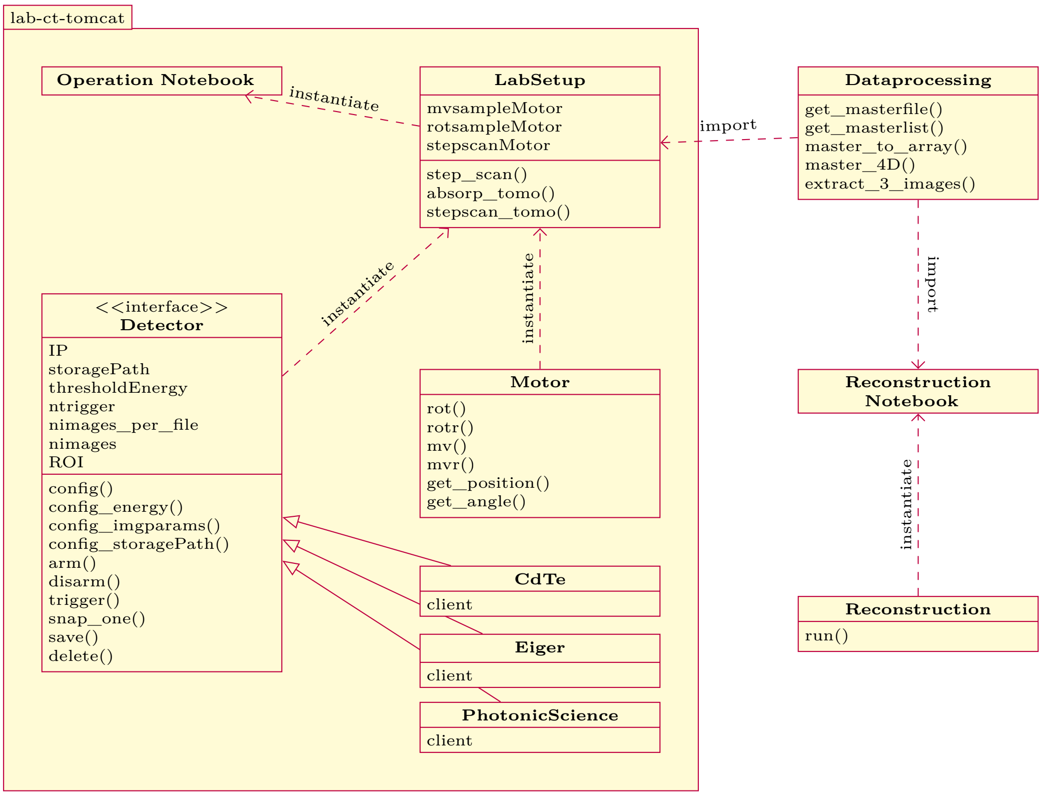


Fig. 4.1.: Class Chart. The interface `Detector` defines the functions and attributes that each detector should implement. `LabSetup` gets assigned 3 motors, a detector of a certain type and implements different functions that make use of all setup hardware. `Reconstruction` handles geometries and algorithms of the ASTRA toolbox. High-level operation happens in Jupyter Notebooks.

4.1.1. Acquiring Images with Eiger

This detector saves data in *HDF5* files. Every acquisition sequence consists of arming, sending a number of triggers for every image to be recorded, and disarming. For every such sequence a master file is created that contains a lot of metadata about the detector operation and links to data files, which contain one data set with one or more images each, as can be seen in Fig. 4.2.

The exact structure of these files is largely influenced by two detector parameters, *ntriggers* and *nimages_per_file*. The first determines how many images the detector expects to take in the next imaging sequence, the latter how large the data set in each data file should be [9]. As will be presented in the next section, this is of importance when operating the detector to avoid data loss and ensure a quick image acquisition with little overhead time.

Fig. 4.2.: File structure of *Eiger's HDF5* files.

4.1.2. Speeding up the Tomographic Process

So far every step scan series had been handled as one imaging series. This means that for every projection angle a master file linking to a data file with 5 images was transferred between the detector and the local storage. Master files are around 22.6 MB, while such a data file is around 14.2 MB. Since the main interest lies on the image, not the metadata, this discrepancy was undesired. Optimally the detector could acquire a whole tomographic data set as one imaging series, but the limiting factor is the buffer space on the detector side. Therefore a break is needed in between, where data saving happens and a new series is started [9].

Experimentally it was derived that overhead time for saving and clearing the buffer can be significantly decreased by combining 20 projection angles of 5 phase steps each and the safe detector operation at those parameters was demonstrated by acquiring several tomographic data sets.

Overhead time between the two methods was tested and the results can be found in Table 4.1. The code in List. 4.1 was the one that generated the faster results. It can be seen that overhead is very similar for the different exposure times but largely different from the process of saving

List. 4.1: Testing acquisition times with the Eiger detector. Shown here is the fast configuration where images are only saved after 20 projections or 100 images.

```
expTime = 5
n_im = 100

eiger.config_imgParams(nimages_per_file=5, ntrigger=n_im)

time1 = time.time()
eiger.arm(expTime)
for i in range(n_im):
    eiger.trigger()
eiger.disarm()
eiger.save()
eiger.delete()
time2 = time.time()

timetot = time2 - time1
print(timetot)
```

Table 4.1.: Acquisition times for 100 images with different exposure times, once with saving every phase stepping scan of 5 images, once with saving only at the end.

	20 proj. per masterfile	overhead	1 proj. per masterfile	overhead
15 s exposure	1523 s	23 s	1590 s	90 s
5 s exposure	520 s	20 s	587 s	87 s

only 1 projection angle per master file. This experiment was only conducted over 20 projection angles and a time difference of 70 s was reached. A typical number of projections for this setup would be 4 per degree, yielding a total of 1440 projections and a time saving of 5040 s or 1.4 h. This is a significant decrease compared to the prevalent method from before.

4.2. Motor Operation

The setup uses two different types of motor stages. One, the sample stage, is operated via the *epics* package for python, the others are operated through a custom written package. Though not implementing the exact same commands, both packages ensure that the python interpreter waits for the motor to reach its position. This is important to check because acquiring a projection or a flat-field while the sample is in movement, or not yet completely out of the detector area, should be avoided.

Instances of the motor classes all implement functions for relative movement, absolute movement and position feedback. They were left untouched since they already allowed high-level operation and worked very reliably.

4.3. The Lab CT as one Object

Acquisition of a whole tomographic data set may take several hours, therefore the risk of human error that leads to bad or incomplete data should be mitigated. The purpose of the `LabSetup` class is to implement the basic functionalities that are regularly used with this setup in such a way that they minimize the potential for human error and maximize time efficiency. This is done by implementing things learned during operation of the setup, like the imaging series optimization from section 4.1.2.

LabSetup gets assigned three motors, one for moving the sample out of the detector area to take flat-field projections, one to rotate the sample and one that moves the grating in order to perform the phase stepping scan. These procedures will generally be performed by the same motors therefore it makes sense to allocate those as defaults and avoid misassignment of such.

Further, LabSetup instantiates an object of a class that implements the detector interface. When instantiating an instance of LabSetup, the user can choose the detector that he would like to use. LabSetup's methods rely on the fact that no matter which type of detector gets instantiated they can all be operated the same way, calling the same functions. This is the reason why the approach with the Detector interface was chosen. All detectors that are so far implemented support at least the functionality that is needed for the 2D operation of the setup. This means that all of them work reliably together with the `step_scan()` function that is defined in LabSetup. Functionality and ease of use of this LabSetup class was proven during several grating tests, including one where Zhitian Shi, a member of the microfabrication team, was able to operate the setup on his own in a very short period of time. The results of this grating test can be seen in appendix A.

For the case of the Eiger detector, where an enhancement of the operation was reached, functions were added to the LabSetup class that make use of those enhancements, either for a phase contrast tomography or for a pure absorption tomography. The tomographic scans by default require the user to visually inspect the movement of all involved motors, that the source is switched on and that the sample is placed on the sample motor stage. This again acts to minimize the risk of something going wrong during the scan and allows operation of the setup by a person who does not have a lot of experience. The system check can also be skipped in case several consecutive tomographies want to be acquired. Further, a log file will be created, containing all important information needed for further processing of the data set, which will be of importance when looking at tomographic reconstruction in chapter 5. This includes the detector threshold, exposure time, information about the scanned angles and info about the geometry. An example of how the LabSetup class can be used to acquire a tomography and a single phase stepping scan can be seen in List: 4.2 and the output of the log file directly follows.

List. 4.2: Usage of the the LabSetup class to acquire a phase stepping tomography and two phase stepping scan with a sample. The distances of the setup can be passed with the constructor like it is seen here or can be added later. These are important properties that should be included in the log file. `sample_in` and `sample_out` are the motor positions where the sample is either inside or outside the frame.

```
base = '/sls/X02DA/Data20/e15889/Maxim_LCT/data/2021/May/tomotryouts/'
lab = LabSetup(storagePath=base, detector='Eiger', source_sample_d=345,
    sample_dect_d=160, vertcenter=410)

sample_in = -2000
sample_out = 6000
angles = np.linspace(0, 185, 185*4, endpoint=False)
lab.stepscan_tomo(storagePath=base+'tomo1', threshold=13000, sample_in=sample_in,
    sample_out=sample_out, angles_degrees=angles, expTime=15, stepsize=0.4)

# phase stepping scan with the sample
lab.mvsampleMotor.mv(sample_in) # move sample in
lab.step_scan(5, 0.4, 15, storagePath=base+'sample') # do a step scan for grating of
     pitch 2 um

# phase stepping scan without the sample
lab.mvsampleMotor.mv(sample_out)
lab.step_scan(5, 0.4, 15, storagePath=base+'sample')
```

List. 4.3: logfile.txt from the tomographic data set acquired by the functions used in List.4.2

```
Threshold: 13000
Exposure Time: 15
Start Angle: 0.0
End Angle: 185.0
Number of Projections: 740
Source to Sample (rotaxis): 345mm
Sample (rotaxis) to Detector: 160mm
Acquisition Time: 56705.87584042549
```

4.4. Conclusions & Perspective

As a result of the implemented software framework it is now possible to operate the setup with a high-level approach, either from Jupyter Notebooks or directly from scripts. Not only does this help to mitigate the risk of something going wrong, it also opens new possibilities for other team members who are now, with very little instruction, able to use this setup as a tool for their own research. While facilitating this, it was also paid attention to the fact that more experienced users might want to make use of low-level operation: The already existing classes were integrated into the framework and can still be called and used. In order to preserve the acquired data for the future, it was made sure that acquisition information gets stored together with it.

In the case of the Eiger detector, acquisition times for a whole tomographic data set were significantly cut by enhancing the file saving procedure. The gained knowledge was integrated into functions for tomographic scans and their successful operation was demonstrated with the acquisition of several such tomographies.

Possible next steps, in order to further enhance the operation, might include:

1. Cutting overhead times of the motors, since they were not looked at during the course of this thesis.

2. Cutting overhead times of the two other detectors and write enhanced tomography functions for them. As demonstrated with Eiger, this can make a significant difference.

5 Data Processing

This chapter will present how the quality of the interferometer can be assessed, as well as the handling and processing of large tomographic data sets. Specifically the retrieval of differential phase contrast and dark-field images will be discussed followed by their tomographic reconstruction. The two main goals were the following:

1. Create a collection of often used data processing functions and enhance their performance if necessary.
2. Implement an easy to use reconstruction class, capable of tomographic reconstruction of fan beam and cone beam shaped geometries.

5.1. Reading Detector Output

As mentioned in 4.1.2, the way the Dectris Eiger saves images is very specific. For further handling of the data it makes sense to import the projection data, or certain slices (detector rows) thereof, into an array of the dimensions ($\#projections \times \#phasesteps \times x \times y$). The values x and y are the pixel rows and columns of the detector and together form one projection image.

The challenge lies in finding a quick way to create said 4D array and handing it to the next step which is the retrieval of the 3 images, discussed in 5.3. In order to do this, one has to iterate over the files and merge the data sets. This is a lengthy process, even if only certain slices of the projection set should be imported. Therefore, 2 different approaches were chosen and their computing times were compared.

Data handling makes use of arrays from the *numpy* package. These data structures use fixed sizes such that the values can be stored closely together in storage and are quickly accessible. It was expected that one fast approach would be to initialize an array of this size with zero values and write the data sets one by one into it. This has the advantage that storage gets only allocated once and the values are directly written into the final structure, but the disadvantage that its shape does not reveal if all values have been added. A second approach was to create a Python list of data sets and write them all into an array at the end. This has the disadvantage that before writing the data into the final structure, the list creation happens as an intermediate step. The advantage is that with this approach, the shape of the final array clearly states the actual amount of imported values. The results of this test can be seen in Table 5.1. Since this is mainly an I/O operation and no calculations are necessary, parallelization does not have a big influence. It was nevertheless tried to split the import of one master file among several processes, but the time saving was minimal. Therefore the parallelized approach was dropped in favor of list appending, which works reliably and its success is verifiable. In any case a repeated import should be avoided. This can be achieved by saving numpy binaries of imported slices on fast internal storage of the machine on which data processing happens. Since those binaries are often only a small fraction of the size of the whole data set they can be reloaded in a matter of seconds.

Table 5.1.: Computing times of two serial (zero initialization of the final array & list appending) and a parallel approach (distribute master files to different processes) to creating a 4D array from Eiger data. The data set contained 1440 projection angles and 2 slices were loaded. Parallelization did not save a lot of time and was therefore dropped in favor of the simplest approach which is list appending.

	Computing Time [s]	
	1st attempt	2nd attempt
zero initialization	251	233
list appending	234	233
splitting by core number	205	204

Apart from the creation of the 4D array, the data processing collection includes functions that can extract single phase stepping scans from a master file, as well as create a 3D array from a pure absorption tomography. These work in similar ways, but data sets are significantly smaller therefore a speed-up was not tried.

5.2. Measuring Visibility & Angular Sensitivity

This setup is often used for testing of gratings produced at PSI and other institutions. Anyone operating the setup should be able to quickly analyze the most important performance metrics of *visibility* and *angular sensitivity*. For this, several functions were added to the collection, able of handling different inputs, making them very versatile and easy to use. Measurement of the visibility needs data from a phase stepping scan and functions were added that directly print a histogram that shows the visibility distribution with its peak as well as an image of the visibility, called the *visibility map*. Measurement of the angular sensitivity is done in a region of the differential phase contrast image. This region should not include include the sample but be constrained to the area of good visibility. A code example of a visibility measurement and angular sensitivity measurement is shown in List. 5.1, an example of an output can be found in Fig. A.2 of the appendix where a test of high aspect ratio bottom-up gold filled gratings is presented.

List. 5.1: Example of system operation and subsequent measurement of visibility and angular sensitivity. `lab` is an instance of LabSetup and the data processing functions are imported as `dp`.

```
steps = 5
grating_period = 2
stepsize = grating_period / steps
expTime = 120
distanceG1_G2 = 0.40
distanceSource_G1 = 0.60
distanceSource_sam = 0.48

lab.step_scan(steps, stepsize, expTime)
ff = dp.master_to_array(dp.get_masterfile(folder, seqID=7)) # step scan file had sequence ID 7
ref = ff[:, 200:500, 460:640] # choosing ROI

vis = dp.visibility(ref) # calculate visibility from a phase stepping scan
dp.plot_vishistogram(vis) # plot the visibility distribution
dp.plot_vismap(vis) # plot the visibility as an image

# step scan with sample
lab.step_scan(steps, stepsize, expTime)
sam = dp.master_to_array(dp.get_masterfile(folder, seqID=8)) # step scan file had sequence ID 8
sam = sam[:, 200:500, 460:640] # choosing ROI

amp, dpc, dci = dp.extract_3_images(sam, ref) # get dpc image for measuring angular sensitivity

# calculation of angular sensitivity takes all distances in m, small region of dpc image is passed
print(dp.angular_sensitivity(grating_period*10**(-6), distanceG1_G2, dpc[50:100, 50:100], distanceSource_G1, distanceSource_sam))
```

5.3. Extracting the 3 Images

Every phase stepping scan in the 4D array from 5.1 must be analyzed and a set of projections (absorption contrast, differential phase contrast or dark field) must be generated. The implementation of FFT (fast Fourier transform) which numpy provides is very efficient as was shown with a comparison of extraction times, the results of which are shown in Table 5.2. The measurement was restricted to a maximum of 100 detector rows (*slices*), which means that from this data, 100 sinograms of each image type were created. This restriction was necessary since it is difficult to import many more rows from the raw data without the Python interpreter crashing. This is a task that should be easily parallelizable since it performs numerical calculations on already loaded arrays, but the results do not demand an increase in speed as it is already reasonably fast.

List. 5.2: Example of the extraction of absorption contrast image and scattering image from a tomographic data set. The data processing functions are imported as `dp` and the extraction happens for the slices 400 to 420.

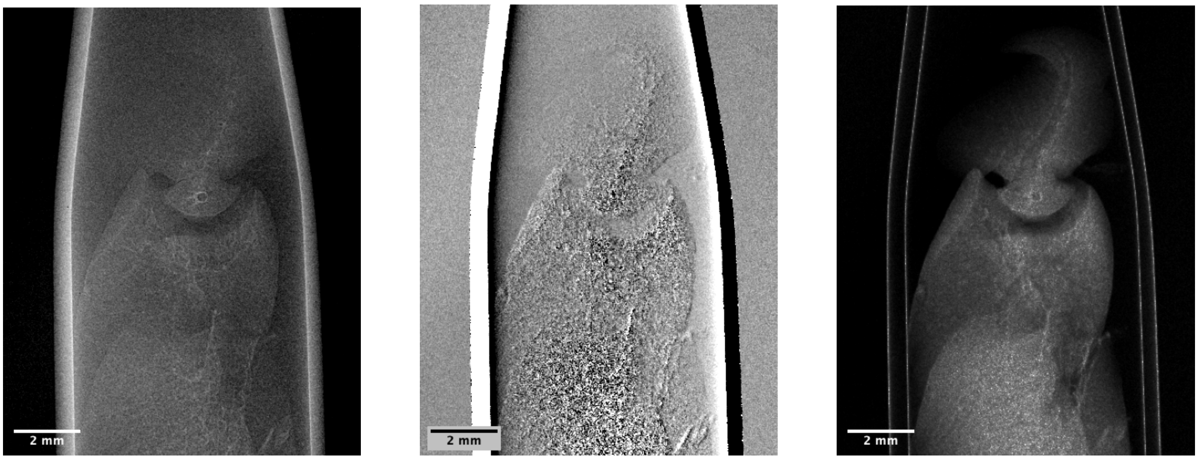
```
folder = '/mnt/test/Data20/Maxim_LCT/data/2021/April/tomography/murine_lung/25
    sec_4perdeg/'
masters = dp.get_masterlist(folder) # create a list with all masterfiles

myslice = np.s_[400:420] # choose which slices from detector data you want to import
dataset = dp.master_4D(masters=masters[1:-1], no_of_projections=740, sino_slice=
    myslice) # first and last masterfiles are flatfields
ff = dp.master_4D(masters[0], 1, myslices)

amp, dci = dp.extract_3_images(dataset, ff, amp=True, dpc=False, dci=True) # extract
     the images
```

Table 5.2.: Performance of the extraction was tested. The data set contained 740 projection angles and all 3 images were extracted on a machine with an Intel Xeon Gold 5222 (4 cores of 2 threads each) and 96 GB of RAM. Extraction is reasonably fast: For handling a limited number of slices from a data set, there is no need for a speed up.

	Computation Time [s]	
	1st attempt	2nd attempt
10 slices	1.27	1.24
50 slices	5.93	5.97
100 slices	12.18	12.24

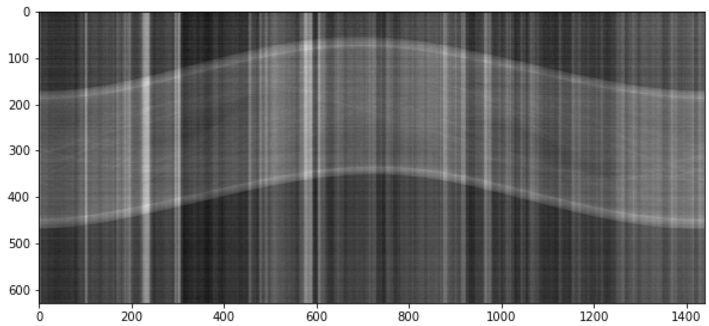
The results from such an extraction process can be seen in Fig. 5.1. These pictures depict a murine lung sample, placed in a capillary. The increased contrast in the dark field signal compared to the absorption contrast image is remarkable in this sample. More samples can be seen in Fig. A.3 and Fig. A.4 of the grating test in the appendix.

During the processing of a real sinogram from a tomography with the murine lung sample it was realized that the photon current provided by the Sigray source was strongly drifting over time. This leads to stripe artifacts after flat field correction as can be seen in Fig. 5.2. A straightforward approach to this would be to increase the frequency at which flat-fields are taken. This has the downside that tomographic scans would again become longer and depending on how fast the current changes it might not solve the problem. Nevertheless it should be mentioned that the LabSetup class from 4.3 was extended in order to take a flat field phase stepping scan after every 20th projection angle. This thesis ended before it was possible to determine the influence on the results. A different solution would be to explore dynamic flat-field correction with a database of flat-field images. The similarity of the projection and the flat-fields could be assessed and the best fitting flat-field would be used for correction.

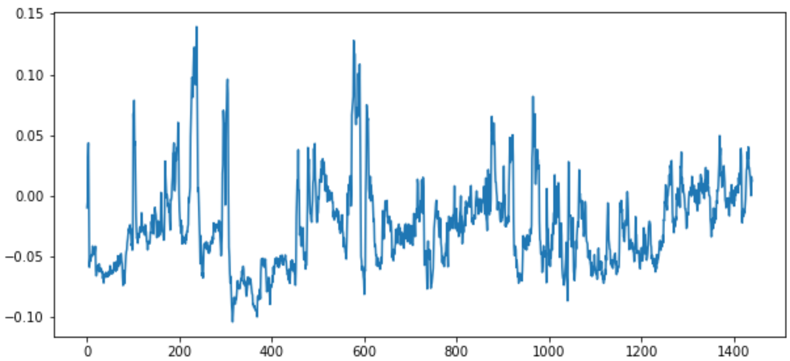


(a) Absorption Contrast **(b)** Differential Phase Contrast **(c)** Dark Field

Fig. 5.1.: The results from processing the phase stepping data. The sample is a murine lung placed in a capillary. The contrast difference for the dark field signal compared to absorption contrast is remarkable in this sample. The magnification of $M = 3$ was considered and the scale bars correspond to 2 mm in the sample plane.



(a)



(b)

Fig. 5.2.: The figure in (a) shows a sinogram from a tomography with the Sigray source and murine lung sample. A total of 1440 projections was acquired (horizontal axis) over a time frame of 11 hours and after performing flat field correction with the same flat field for all projections the drift of the source was realized. The mean values of the lowest 100 pixels are shown in (b), indicating that the drift is very strong in the first 6 hours and gets better over time.

5.4. Tomographic Reconstruction

So far, only the *tomopy* package had been used for tomographic reconstruction from this setup. Tomopy uses CPU parallelization to reconstruct from projections taken with parallel geometries and is very well documented and user-friendly. It also includes a lot of useful tools for preprocessing the acquired data. Since tomopy was not able to handle reconstruction from cone beam shaped geometries, access to such algorithms was sought. The ASTRA toolbox is a package that implements a collection of sophisticated reconstruction algorithms, among them FBP & FDK (a 3D implementation of FBP) for cone and fan beam geometries as well as iterative methods, like SIRT. In addition to offering a wide variety of algorithms, many of them make use of the highly parallelizable Nvidia CUDA GPU architecture. The downside of ASTRA is, that it is poorly documented and its application is not user-friendly. For this reason, a reconstruction class was created that should make reconstruction with ASTRA as easy as it is with tomopy. Since the parallel geometry algorithms of the ASTRA toolbox were already integrated into tomopy in 2016, such that its users get access to GPU accelerated reconstruction within the same package [10], the focus was on reconstruction from fan-shaped and cone-shaped geometries.

5.4.1. The Reconstruction Class

In order for ASTRA to work properly it needs four basic objects:

Volume geometry An object describing the space in which the sample was placed.

Projection geometry Virtual representation of the real world setup dimensions.

Volume data An object in which the reconstructed data will be stored. It receives its dimensions from the volume geometry.

Projection data An object in which the data to be reconstructed – the sinograms – are stored.

Geometric magnification of a cone beam shaped setup is an issue that can be very confusing when trying to define a projection geometry in ASTRA. The standard origin is defined as the center of the sample and the reconstruction voxel size is $1 \times 1 \times 1$. For the geometry implementation in the Reconstruction class, a little trick presented by [14] is used to shift the origin into the detector plane, which makes the projection pixel the same size as the one from the reconstruction. All of the objects from the list above are used to create an algorithm object and such an object is also created when instantiating an object from the Reconstruction class using a cone beam shaped geometry. For the fan beam geometry it is a list of algorithm objects that is created. This is due to the fact that the original fan beam geometry of ASTRA is only capable of handling one slice at a time. The fan beam handling of several slices is a significant improvement over using ASTRA directly and having to work with loops in the script or the Jupyter Notebook. A Reconstruction object has only one method, `run()`, that takes an optional argument `iterations` for iterative algorithms like SIRT.

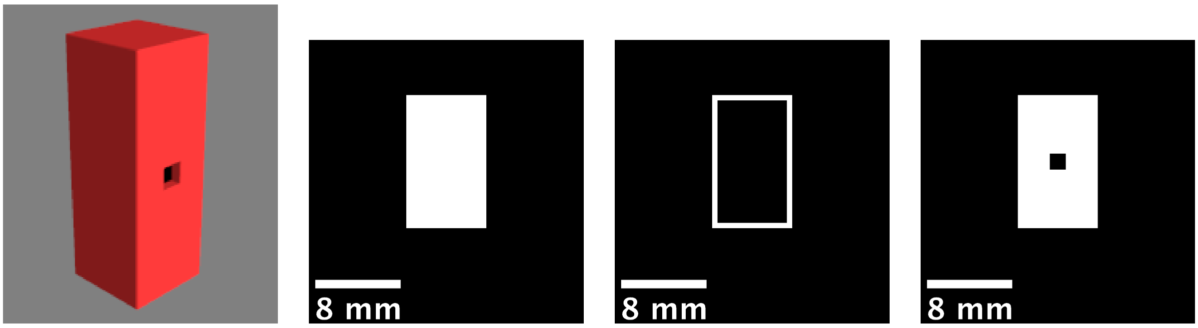
List. 5.3: Example of how the Reconstruction class can be used to reconstruct slices from a number of sinograms. The goal was to make it as easy as it is in the tomopy package, therefore the direct comparison is given in the source code.

```
data = dp.make_sino(dci) # the data to be reconstructed are dark field sinograms

distance_source_origin = 235
distance_origin_detector = 685-235
detector_pixel_size = 0.075
angle_stop = 185
rotcenter = 565.7
vertcenter = 410

num_of_projections = 740
angles = np.linspace(0, angle_stop, num=num_of_projections, endpoint=False)
angles = angles * np.pi / 180 # ASTRA works with radians instead of degrees

# reconstruction using tomopy with the gridrec algorithms
recon = tomopy.recon(data,
                    angles,
                    rotcenter,
                    algorithm='gridrec')

# reconstruction using a cone beam shaped geometry and SIRT with 200 iterations
recon = Reconstruction('cone',
                    'SIRT3D_CUDA',
                    rotcenter,
                    angles,
                    distance_source_origin,
                    distance_origin_detector,
                    detector_pixel_size,
                    data,
                    vertcenter=vertcenter)

data_recon = recon.run(iterations=200)
```

ASTRA by default assumes the rotation center to be in the center of the sinogram and in tomopy it is defined by a float passed to the reconstruction function, specifying the pixel where the rotation center is. The Reconstruction class is implemented such that the rotation center can now be passed the same way as in tomopy. This makes it easy to use both packages inside the same script. Further, the vertical difference between the source and the central slice of the detector can be passed as an optional argument. This is an important property when reconstructing data from strongly cone shaped geometries. Important to note is that the vertical difference between the sample and the X-ray source can not be adjusted. This would only be possible with a definition of the geometry using vectors. This is not only complicated but the fast FDK algorithm is so far not supported for these geometries. This means that in order to be able to create the same geometry in ASTRA as in the real setup, the X-ray source and the sample have to be vertically aligned, while the detector may be slightly shifted up or down. The projection of a vertically aligned tip of a needle can be used to determine the central detector row. An example of how the Reconstruction class can be used is given in List. 5.3.

5.4.2. Phantom Creation & Performance Testing

In order to test the performance of the package and determine how strongly the real setup is influenced by the diverging beam, several phantoms were created. The first one was a hollow cuboid with a square hole. It was derived from [14] and scaled to resemble the size of a real sample placed in the setup. A cuboid was chosen because the sharp edges show the influence of only partially attenuated beams very clearly. The phantom itself, together with three of its slices, are shown in Fig. 5.3. A simulated projection of the phantom was done using the exact geometry that was used in the real setup with the Sigray source. A look at the edges, as seen in Fig. 5.4, revealed that the blur caused by beams that only partially pass through the walls is significant and that reconstruction should definitely be done using a cone beam algorithm.



(a) (b) (c) (d)

Fig. 5.3.: In order to determine how strong the influence of the diverging beam in the real setup is, a hollow box phantom was created. This phantom is inspired by [13] where picture (a) is from. The cuboid shape has the advantage that beams which only partially pass through the walls will be clearly visible as a blur on the otherwise sharp edge. Slices through the back wall (b), the middle (c) and the front wall (d) show the sharp edge and the scale bar shows that the size of the sample corresponds to the sample size of the murine lung from Fig. 5.1.

For testing how well the algorithms perform at reconstructing small details, a phantom was created that resembled the capillary from the murine lung sample from Fig. 5.1 and it was populated with 1000 spheres of randomized placement, radius and absorption coefficient. This phantom and one of its projections is shown in Fig. 5.5. Reconstruction of several slices using both FDK and SIRT were performed and the reconstruction results from this noise-free data are very convincing. Both reconstructions are shown in Fig. 5.6.

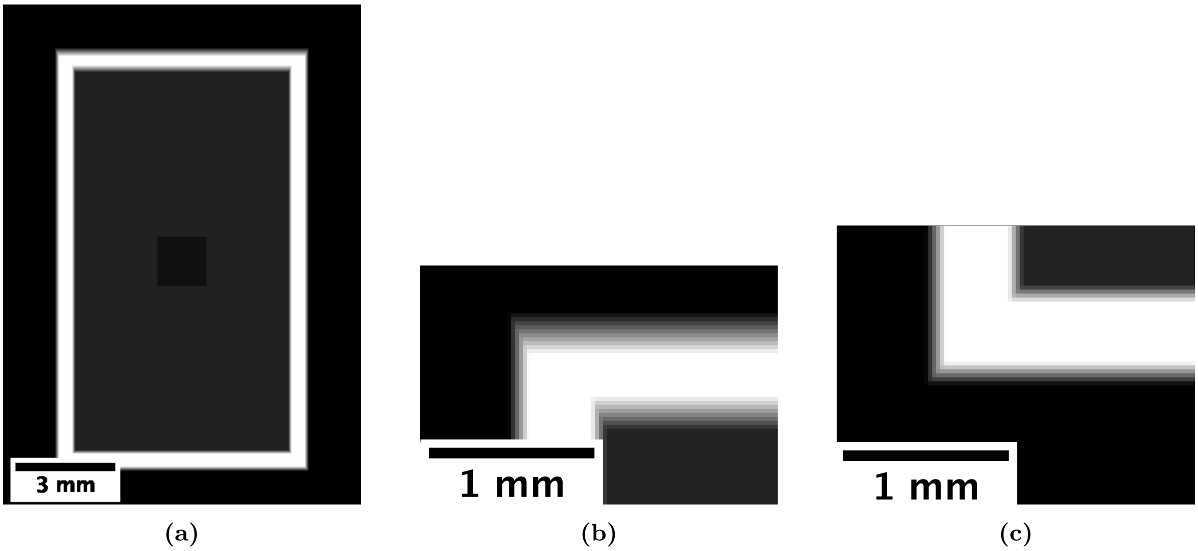


Fig. 5.4.: A simulated set of projections from the phantom in Fig. 5.3 was created with ASTRA and a cone beam geometry that corresponds to the real setup. Such a projection is shown in (a), clearly visible is the lower attenuation in the middle where the beam only passes through back and front wall (only the back wall in the case of the hole). A slight blur at the edges is also visible, especially when zooming into the edges (b) and (c). The diverging beam has a higher influence on the upper edge since the sample was placed above the vertical center (visible in Fig. 5.3).

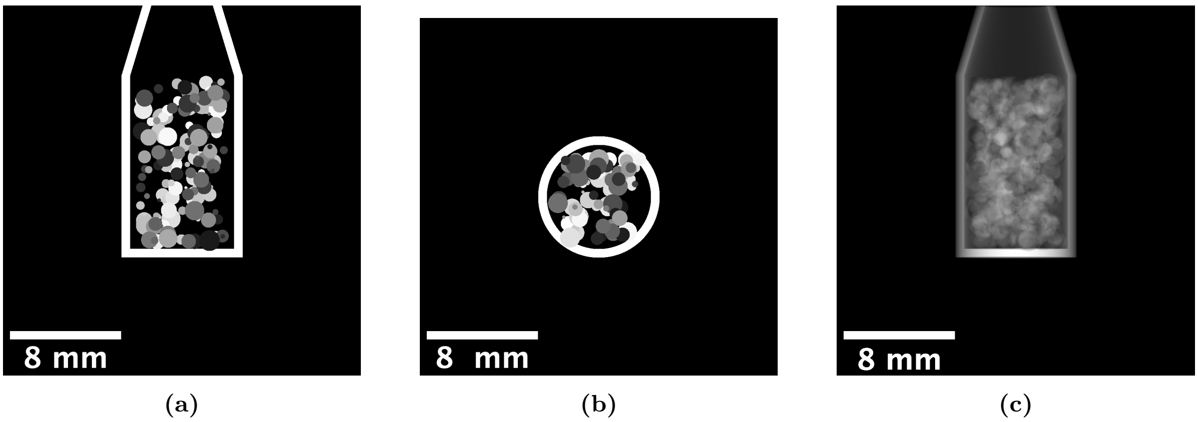
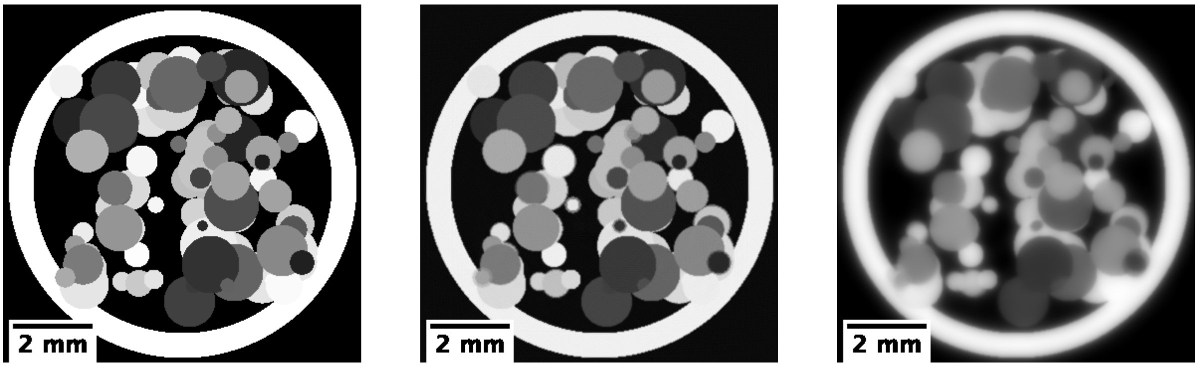


Fig. 5.5.: A phantom, resembling the murine lung sample from Fig. 5.1 in shape and size, was created in order to test the performance of the Reconstruction class. It consists of a cylindric capillary, populated with 1000 spheres of randomized placement, density and radius. Slices through the side and the top of the phantom are shown in (a) and (b) respectively. The phantom was placed in a cone beam geometry of ASTRA and a set of simulated projections was created. One such simulated projection is shown in (c). Assuming it to be the size of the murine lung sample, the scale bar would correspond to 8 mm.



(a) (b) (c)

Fig. 5.6.: A total of 1440 projections (4 per degree) from Fig. 5.5c were used for reconstruction using a cone beam geometry created with the Reconstruction class. The original phantom slice is shown in (a), a reconstruction using FDK in (b) and a reconstruction using SIRT with 100 iterations in (c). Computing times for 100 slices were 3.4 s with FDK and 207 s with SIRT. With such a high number of projections and completely noise-free data, FDK is not only faster but also delivers superior reconstruction quality as is clearly visible from the images.

5.5. Conclusions & Perspective

Functions were created and collected to quickly load projections from raw acquisition data and retrieve absorption, differential phase and dark field images from them. Their performance was analyzed and improvements were tried where deemed necessary.

Tomographic reconstruction is now possible with 3D enhanced algorithms while the complexity for the user was kept to a level that is comparable to the software used before. With the help of phantoms it was shown that the chosen geometry corresponds to the real setup and that the quality of the reconstruction with high density and noise-free data is very good.

Further exploration of image retrieval from the phase stepping scans could include dynamic flat-field correction with a flat-field database. For tomographic reconstruction it could include the minimal number of projections needed for a desired spatial resolution. For this purpose the phantom with the spheres could be improved with smaller structures, a smaller range of densities and noise could be artificially added.

6 Conclusions & Perspective

During the course of this thesis I learned how X-ray grating interferometry works and how to operate the micro CT setup at the Paul Scherrer Institute. This led to the identification of points of improvement in the setup and an operation software package was created which makes it possible to operate the hardware components from within a single class. The function of the package was proven with the acquisition of several tomographic data sets and a grating test, the results of which can be found in appendix A. It was also demonstrated that by optimizing detector overhead time, a significant reduction in acquisition time is possible. The operation package allows an easy integration of further hardware components and its sustainability is supported by its placement on the PSI internal Git platform and a strong emphasis on complete Python docstrings. This not only helps the process of further development but the end-user also gets access to package information by a call of the `help()` function within the script or notebook. Further improvement of this package should include the cutting of overhead times in the motors as well as in the other detectors.

The created package for tomographic reconstruction and data processing follows the same principles of being well documented and easily accessible through PSI Git. It is now possible to access powerful tomographic reconstruction algorithms of the ASTRA toolbox with an easy-to-use class and also a function to create projections from a phantom was added. The functionality of the tomographic reconstruction class was demonstrated with the help of several phantoms, all of which are accessible via the corresponding Jupyter Notebooks. The collection for data processing contains all needed functions to assess the most important quality metrics of the interferometer and to prepare tomographic data for its reconstruction. This collection only contains the most basic functions and is expected to grow in the future. This could include dynamic flat-field correction for the image retrieval, which might help solve drift issues of the X-ray source. When it comes to tomographic reconstruction, the phantoms could be developed further by adding different structural features and make use of artificial noise in order to better assess the merits of each algorithm.

A Testing of Bottom-Up Gold Filled High Aspect Ratio Gratings for X-Ray Interferometry

The following section includes a grating test that was performed for high aspect ratio 1D absorption gratings. The gratings were produced using a bottom-up filling Au electrodeposition technique that yields in void-free gold fillings of the trenches. The grating tests included measurements of visibility and angular sensitivity, as well as the imaging quality assessment with several samples.

A.1. Grating Tests

The performance of 1D-gratings was tested with an X-ray grating interferometer setup implemented at TOMCAT, Paul Scherrer Institute. The setup consisted of a microfocus X-ray source (Hamamatsu Photonics, model L10101) and a Dectris Eiger R 1M photon detector with a pixel size of 75 µm. The X-ray source was operated with a tube voltage of 42 kV and a current of 200 µA, providing a source size of 10 µm.

The interferometer was set up in Talbot-Lau configuration, with gratings G_0, G_1 and G_2 being placed between the X-ray source and the detector, as shown in Fig. A.1. The purpose of the experiment was to measure the visibility in an X-ray interferometer [17] - a performance metric for grating interferometers, which depends on absorption properties of G_0 and G_2. Better uniformity of the gold fillings translates into better absorption properties which allows to evaluate the manufacturing quality.

A design energy of 20 keV and the Talbot order 9 were chosen for the visibility tests. The phase shifting grating G_1 was manufactured from silicon with a trench height of 25 µm, providing a phase shift of π at the given energy. Absorption gratings G_0 and G_2 were Au-filled with a height of 60 µm All gratings – G_0, G_1, G_2 – had a pitch of 2 µm and a duty cycle of 0.5 (trench

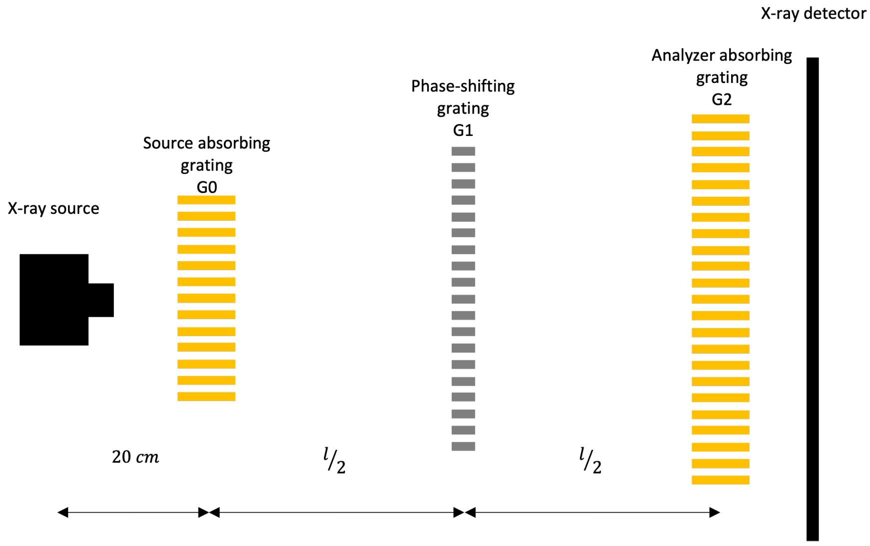
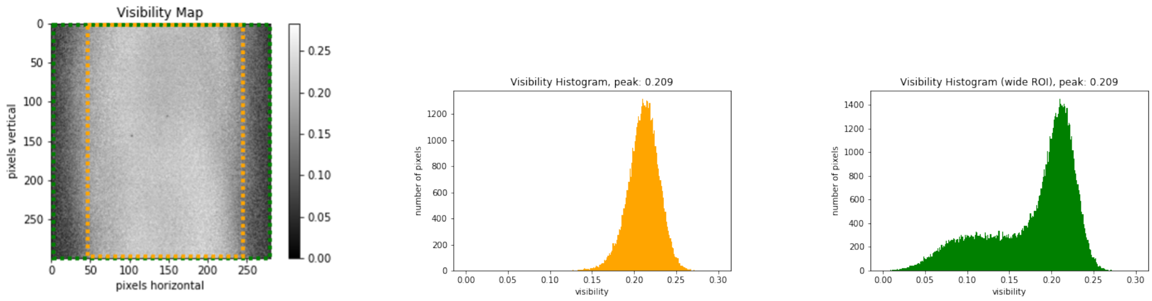


Fig. A.1.: A schematic of the symmetric X-ray grating interferometry setup, used for the quality assessment of the 1D gratings. Source-G_0 distance was 20 cm, and the detector was placed right behind G_2. For the case of the Talbot order 9, which was used for visibility measurements, the overall length l was 29 cm. For the case of the Talbot order 23, the overall length was 73.6 cm.

size of 1 µm. The visibility was measured using a phase stepping technique [23] [21], where G_2 was moved in the grating plane perpendicular to the Au line structures. G_2 was moved by a total of 5 steps over one grating period (0.4 µmper step) and one image was taken for each step. The acquired visibility map can be seen in the Fig. A.2a, and the peak visibility of 21% was achieved, which is considered to be an indication of a good grating quality [17].

In order to visually assess the image quality of the tested gratings, two samples were alternately placed at a distance of 9.5 cm, upstream to G_1. The samples were chosen to fit in the area of a uniform visibility (Fig. A.2a, orange selection). For each sample, the stepping scan was performed [23], allowing to reconstruct three images with absorption, differential phase and dark-field contrast, respectively. The total magnification at the Talbot Order 9 was about 2, resulting in the effective pixel size of 38 µm.

Speaking of the imaging applications, angular sensitivity is another important performance metrics of the grating interferometer, which corresponds to the minimal refraction angle (α_{min}) that can be still detected by the interferometer. The refraction occurs after the interaction of X-rays with the features inside the sample. It was demonstrated in [21], that an angular sensitivity of less than 100 nrad is desirable to achieve good contrast. As for the case of our setup at the Talbot order 9, the angular sensitivity of 134 nrad was calculated, using the formula in [21] and calculating the standard deviation in the empty space of the differential



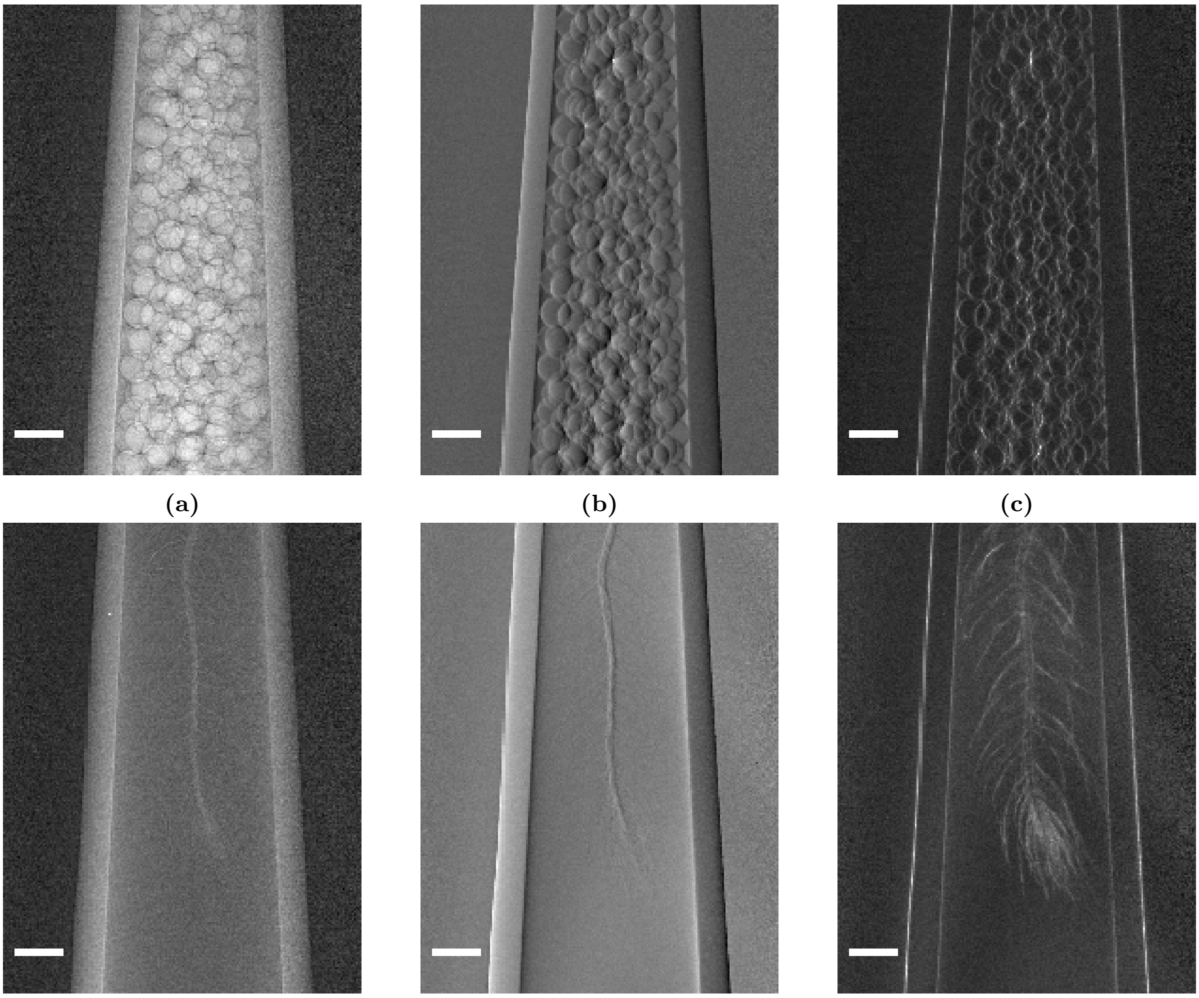
(a) (b) (c)

Fig. A.2.: The visibility map (a) indicates a uniform visibility distribution in the central region of interest (shown in orange), corresponding to the pronounced peak of 21% visibility at the histogram of individual pixels within this area (b). When enlarging the analyzed area in the horizontal direction (a, shown in green), the visibility drops rapidly towards the edges as seen from the histogram (c). This is due to the higher absorption at the edges of the G_2 grating. Because of the high aspect ratio (60) and the high divergence of an X-ray beam generated by the laboratory X-ray source, the strong misalignment among X-rays and grating lines progressively increases the grating absorption at the edges. The usual solution for this issue is bending the grating to match the geometry of the X-ray wave front and improve their transmission.

phase contrast image. As angular sensitivity improves with using the higher distances, we decided to increase the G_0-G_2 distance up to 73.6 cm, to match the Talbot order 23. In addition, we modified the sample: the polysterene balls were put in the ethanol to reduce the absorption contrast. The sample was placed upstream the G_1 at the distance of 9 cm and the magnification of 1.6 was achieved. The imaging results shown in Fig. A.4a-c indicated an angular sensitivity of 90 nrad. This allowed to clearly highlight sample interfaces in the differential phase contrast, while showing no contrast in absorption mode.

A.2. Conclusion

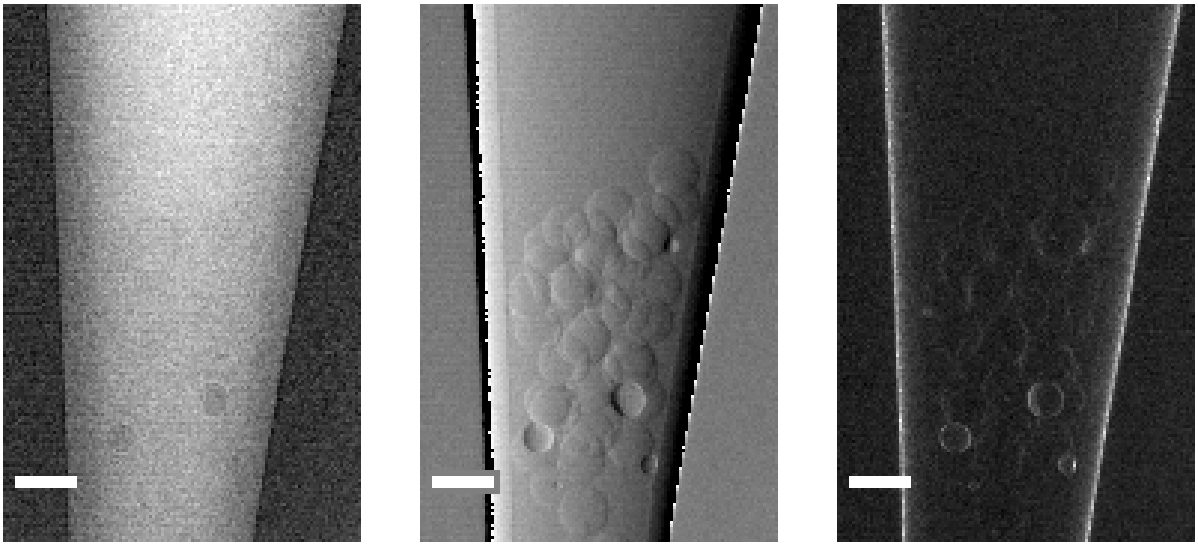
The performance of 1D gratings was tested by measuring the peak visibility of 21% in a Talbot-Lau X-ray interferometry setup. The recorded number indicates high manufacturing quality of the produced gratings, which was additionally confirmed by the good image quality of imaged samples, leading to high potential for such gratings in material science and biomedical applications.



(a) **(b)** **(c)**

(d) **(e)** **(f)**

Fig. A.3.: Absorption (a, d), differential phase (b, e) and dark field (c, f) contrast images of two samples - polysterene balls of 700 µm and stem of Poaceae, respectively. The plant (Poaceae) sample is almost transparent in the absorption and phase contrast regimes, while the dark-field image clearly reveals more details of its inner structure. Scale bars at all images corresponds to 1 mm at the sample position as the magnification factor of 2 is taken into account.



(a) **(b)** **(c)**

Fig. A.4.: Absorption (a), differential phase (b) and dark field (c) contrast images of the polystyrene balls with the diameter of 700 µm placed in the ethanol. The sample provides no contrast in the absorption mode while clearly revealing its structure in the differential phase image and, partially, - in the dark field image. Scale bars at all images corresponds to 1 mm at the sample position as the magnification factor of 1.6 is taken into account.

B Operation & Reconstruction Software

last update of this README file: 4. June 2021

This is a package to operate and process the data of the LAB CT setup at TOMCAT. A working clone of this package is currently located in 'Data20/Maxim_LCT/Lionel_dev/lab-ct_tomcat/' and can for example be operated from cons-10.

please direct questions to lionel.peer@gmail.com

B.1. Requirements for operating the setup

- make sure the low level class DEigerClient is added to this folder (or make sure Detector.py gets access to this file)
- make sure the package smaract_client_py3 from git.psi.ch is cloned in the same folder (or just make sure that LabSetup.py gets access to this package)
- add or change smaract_client_py3.channel_definition.py where you instantiate channels for the smaract motors (gratings towers) and for the epics motors (sample tower)
- make sure controls.remote_detector for calling the CdTe detector is reachable by Detector.py
- make sure client.PS_GSENSE_Control for calling the PhotonicScience detector is reachable by Detector.py

B.2. Requirements for operating the reconstruction

- conda environment with astra toolbox and tomopy, a currently working command that creates such an environment with the name 'tomoastra' is given below:

conda create -n tomoastra -c conda-forge -c astra-toolbox/label/dev python=3.6 astra-toolbox tomopy

- your workstation needs a Nvidia GPU in order to use of the Reconstruction class!

B.3. Operation Software

List. B.1: `Detector.py`

```python
import os
import glob
import h5py
import hdf5plugin
import matplotlib.pyplot as plt
import math
import numpy as np
import re
import time


# these must be provided
import DEigerClient as eigclient
import controls.remote_detector
import client.PS_GSENSE_Control as PS


class Detector:
    """Detector Class defines methods to be implemented by different subclasses of Detector.
    It is kind of for illustration of how you should create functions and is responsible for handling the variables that are common among all detectors.
    """

    def __init__(self, IP, storagePath, photonEnergy, thresholdEnergy, ROI=None, ntrigger=5, nimages=1, nimages_per_file=5, expTime=5):
        """Instantiate and initialize a Detector.

        Parameters:
        IP (str): IP address of the detector server
        storagePath (str): where to store the images
        photonEnergy (int): targeted photon energies
        thresholdEnergy (int): detector threshold photon energy
        ROI (numpy.s_): 2D numpy slice object specifying region of interest
        ntrigger (int): expected number of sent triggers
        nimages (int): images to be taken for each trigger
        nimages_per_file (int): images to be stored in one array
        """
        self.IP = IP
        storagePath = os.path.join(storagePath, '') # add slash at end of path if not already there
        self.storagePath = storagePath
```

```
        if not os.path.exists(self.storagePath):
            os.makedirs(self.storagePath)
        self.photonEnergy = photonEnergy
        self.thresholdEnergy = thresholdEnergy
        self.ntrigger = ntrigger
        self.nimages = nimages
        self.nimages_per_file = nimages_per_file
        self.ROI = ROI
        self.expTime = expTime

    def config_energy(self, photonEnergy, thresholdEnergy):
        """Reconfigure targeted photon energy and detector threshold energy.

        Parameters:
        photonEnergy (int): targeted photon energies
        thresholdEnergy (int): detector threshold photon energy
        """
        self.photonEnergy = photonEnergy
        self.thresholdEnergy = thresholdEnergy

    def config_storage_path(self, storagePath):
        """Reconfigure where to store the images.

        Parameters:
        storagePath (str): where to store the images
        """
        storagePath = os.path.join(storagePath, '')
        self.storagePath = storagePath
        if not os.path.exists(self.storagePath):
            os.makedirs(self.storagePath)
        print("Saving file to " + self.storagePath)

    def config_imgParams(self, ntrigger, nimages, nimages_per_file):
        """Reconfigure expected number of sent triggers and number of images taken per trigger.

        Parameters:
        ntrigger (int): expected number of sent triggers
        nimages (int): images to be taken for each trigger
        nimages_per_file (int): how many pictures to store in one array
        """
        self.ntrigger = ntrigger
        self.nimages = nimages
        self.nimages_per_file = nimages_per_file

    def config_ROI(self, ROI):
        """Reconfigure region of interest of the detector.
        Useless at the moment, but may help in the future

        Parameters:
        ROI (numpy.s_): 2D numpy slice object specifying region of interest
        """
        self.ROI = ROI

```

```
    def snap_one(self, expTime, saving=True):
        """Take one image with specified exposure time and save it to the detectors storage path.

        Parameters:
        expTime (int): seconds of exposure time
        """
        self.expTime = expTime

    def arm(self, expTime):
        """Make detector ready for acquisition sequence.

        Parameters:
        expTime (int): seconds of exposure time
        """
        self.expTime = expTime

    def trigger(self):
        """Append an image to the acquisition sequence."""
        print("Not implemented for parent class Detector.")

    def disarm(self):
        """End the acquisition sequence"""
        print("Not implemented for parent class Detector.")

    def delete(self):
        """Delete all images on the server side."""
        print("Not implemented for parent class Detector.")

    def save(self):
        """Save all images from the server to local storage."""
        print("Not implemented for parent class Detector.")

    def show(self, image, ROI=np.s_[:,:], vmin=None, vmax=None, figsize=None):
        """Show an image.

        Parameters:
        image (numpy.array): 2D or 3D numpy array containing the image to be shown. in case of 3D, the first image will be shown. try show_parallel().
        ROI (numpy.s_): 2D numpy slice for zooming into specific region
        vmin (float): min data range
        vmax (float): max data range
        figsize (float, float): float tuple, size in inches (fuck imperial!)
        """
        plt.subplots(figsize=figsize)
        try:
            if image.ndim is 3:
                plt.imshow(image[0][ROI], cmap='gray', vmin=vmin, vmax=vmax)
            elif image.ndim is 2:
                plt.imshow(image[ROI], cmap='gray', vmin=vmin, vmax=vmax)
        except:
            print("This image does not have the right shape. Enter a 2D or 3D numpy array")

```

```
    def show_parallel(self, images, ROI=np.s_[:,:], vmin=None, vmax=None, figsize=None):
        """Show several images next to each other.

        Parameters:
        image (numpy.array or list): 3D numpy array containing images to be shown or list of 2D arrays
        ROI (numpy.s_): 2D numpy slice for zooming into specific region
        vmin (float): min data range
        vmax (float): max data range
        figsize (float, float): float tuple, size in inches (fuck imperial!)
        """
        if type(images) is list:
            no_plots = len(images)
        elif type(images) is np.ndarray:
            no_plots = images.shape[0]

        fig, axs = plt.subplots(1, no_plots, figsize=figsize)
        for i in range(no_plots):
            axs[i].imshow(images[i][ROI], cmap='gray', vmin=vmin, vmax=vmax)
        plt.show()

class Eiger(Detector):
    """Create an instance of Eiger if you want to use this detector without doing any phase stepping or even tomography."""

    def __init__(self, storagePath, IP="129.129.99.92", photonEnergy=20000, thresholdEnergy=10000, ntrigger=100, nimages=1, nimages_per_file=5, init=False):
        """Instantiate and initialize an Eiger Detector.

        Parameters:
        IP (str): IP address of the detector server
        storagePath (str): where to store the images
        photonEnergy (int): targeted photon energies
        thresholdEnergy (int): detector threshold photon energy
        ntrigger (int): expected number of sent triggers
        nimages (int): images to be taken for each trigger
        nimages_per_file (int): how many images to store in one array
        """
        super(Eiger, self).__init__(storagePath=storagePath, IP=IP, photonEnergy=photonEnergy, thresholdEnergy=thresholdEnergy, ntrigger=ntrigger, nimages=nimages, nimages_per_file=nimages_per_file)
        self.client = eigclient.DEigerClient(host=self.IP)
        if init == True:
            print("Reinitializing the Eiger Detector, setting seq ID to 1...")
            self.client.sendDetectorCommand("initialize")
        else:
            print("Configuration of Eiger Detector...")
        self.client.setDetectorConfig("ntrigger", self.ntrigger)
        self.client.setDetectorConfig("nimages", self.nimages)
        self.client.setDetectorConfig("photon_energy", self.photonEnergy)
        self.client.setDetectorConfig("threshold_energy", self.thresholdEnergy)
        self.client.setFileWriterConfig("nimages_per_file", self.nimages_per_file)
```

```python
        print("Photon energy set to " + str(self.client.detectorConfig("photon_energy")["value"]) + " eV")
        print("Threshold energy set to " + str(self.client.detectorConfig("threshold_energy")["value"]) + " eV")
        print("ntrigger set to " + str(self.client.detectorConfig("ntrigger")["value"]))
        print("nimages set to " + str(self.client.detectorConfig("nimages")["value"]))
        print("nimages_per_file set to " + str(self.client.fileWriterConfig("nimages_per_file")["value"]))
        print("Saving files to " + self.storagePath)

    def reinitialize(self):
        """Reset the sequence ID to 1 and keep detector parameters."""

        print("Reinitializing the Eiger Detector, setting seq ID to 1...")
        self.client.sendDetectorCommand("initialize")
        self.__init__(self.storagePath, self.IP, self.photonEnergy, self.thresholdEnergy, self.ntrigger, self.nimages, self.nimages_per_file)

    def config_energy(self, thresholdEnergy, photonEnergy=None):
        """Reconfigure targeted photon energy and detector threshold energy.

        Parameters:
        thresholdEnergy (int): detector threshold photon energy
        photonEnergy (int): targeted photon energies
        """
        super(Eiger, self).config_energy(photonEnergy=photonEnergy, thresholdEnergy=thresholdEnergy)
        print("Reconfiguration of Eiger Detector...")
        if photonEnergy is not None:
            self.client.setDetectorConfig("photon_energy", self.photonEnergy)
        self.client.setDetectorConfig("threshold_energy", self.thresholdEnergy)
        print("Photon energy set to " + str(self.client.detectorConfig("photon_energy")["value"]) + " eV.")
        print("Threshold energy set to " + str(self.client.detectorConfig("threshold_energy")["value"]) + " eV.")

    def config_imgParams(self, ntrigger, nimages, nimages_per_file):
        """Reconfigure expected number of sent triggers and number of images taken per trigger

        Parameters:
        ntrigger (int): expected number of sent triggers
        nimages (int): images to be taken for each trigger
        nimages_per_file (int): how many images to store in one array
        """
        if ntrigger > 120:
            print('Recommended to go to lower ntrigger value. This can lead in the detector hanging up and data loss.')
        super(Eiger, self).config_imgParams(ntrigger=ntrigger, nimages=nimages, nimages_per_file=nimages_per_file)
        print("Reconfiguration of Eiger Detector...")
        self.client.setDetectorConfig("ntrigger", self.ntrigger)
```

```python
        self.client.setDetectorConfig("nimages", self.nimages)
        self.client.setFileWriterConfig("nimages_per_file", self.nimages_per_file)
        print("ntrigger set to " + str(self.client.detectorConfig("ntrigger")["value"]))
        print("nimages set to " + str(self.client.detectorConfig("nimages")["value"]))
        print("nimages_per_file set to " + str(self.client.fileWriterConfig("nimages_per_file")["value"]))

    def arm(self, expTime):
        """Arm the Eiger for passed exposure time.

        Parameters:
        expTime (float): exposure time in seconds
        """
        super(Eiger, self).arm(expTime)
        self.client.setDetectorConfig("frame_time", self.expTime + 0.000020)
        self.client.setDetectorConfig("count_time", self.expTime)
        print("Arming the Eiger Detector...")
        retVal = self.client.sendDetectorCommand("arm")
        if type(retVal) is not dict:
            print("EIGER control hang and got probably reinitialized")
            sys.exit("EIGER hang")
        self.last_sq_id = retVal['sequence id']
        print("Sequence ID is: " + str(self.last_sq_id))

    def disarm(self):
        """Disarm the Eiger. End acquisition sequence."""
        time.sleep(0.2)
        self.client.sendDetectorCommand("disarm")
        print("Eiger Detector disarmed")

    def trigger(self):
        """Acquire image for acquisition sequence."""
        self.client.sendDetectorCommand("trigger")

    def snap_one(self, expTime, saving=True):
        """Take one image with given exposure time

        Parameters:
        expTime (float): exposure time in seconds
        saving (bool): default True, specifies if detector should save() and delete()

        Return:
        storagePath (str): where image was saved
        last_sq_id (int): the sequence id of the last image
        """
        super(Eiger, self).snap_one(expTime)
        self.arm(self.expTime)
        self.trigger()
        self.disarm()
        if saving:
            self.save()
```

```
                self.delete()
            return self.storagePath, self.last_sq_id

    def save(self, storagePath=None):
        """Save all images from the server to storagePath or to default storage path
         of detector.

        Parameters:
        storagePath (str): default is self.storagePath
        """
        time.sleep(1)
        matching = self.client.fileWriterFiles()
        old_path = self.storagePath
        if storagePath is not None:
            self.config_storage_path(storagePath)
        for fn in matching:
            self.client.fileWriterSave(fn, self.storagePath)
        self.config_storage_path(old_path)

    def save_onlydata(self):
        """Save only the _data files created by Eiger to specified storage Path."""
        time.sleep(1)
        matching = self.client.fileWriterFiles()
        contains_data = lambda x: "_data_" in x
        matching = list(filter(contains_data, matching))
        for fn in matching:
            self.client.fileWriterSave(fn, self.storagePath)

    def delete(self):
        """Delete all images on the server side."""
        time.sleep(1)
        matching = self.client.fileWriterFiles()
        for fn in matching:
            self.client.fileWriterFiles(fn, method='DELETE')

    def master_to_array(self, file):
        """Return 3D numpy array of images stored in master file.

        Parameters:
        file (str): path to h5 master file containing images
        """
        with h5py.File(file, 'r') as f:
            arr = np.array(f['entry']['data']['data_000001'])
            for i in list(f['entry']['data'].keys())[1:]:
                try:
                    arr = np.append(arr, f['entry']['data'][i], axis=0)
                except:
                    print("Some images might not have been added due to corrupted
    file links")
                    break
        return arr

    def get_masterfile(self, sequence_id):
        """Return path to master file with specified sequence ID.
```

```python

        Parameters:
        sequence_id (int): which file to get path from

        Return:
        path (str): path to searched master file
        """
        contains_id = lambda x: ("_" + str(sequence_id) + "_master") in x
        file = list(filter(os.path.isfile and contains_id, glob.glob(self.storagePath + '*')))
        return file[0]

class CdTe(Detector):
    """Create an instance of CdTe if you want to use this detector without doing any phase stepping or even tomography."""

    def __init__(self, storagePath, IP="129.129.99.75", photonEnergy=None, thresholdEnergy=20000, thresholdEnergy2=50000, ntrigger=10):
        """Instantiate CdTe.

        photonEnergy (int): this detector does not have this parameter.
        """
        super(CdTe, self).__init__(IP=IP, storagePath=storagePath, photonEnergy=photonEnergy, thresholdEnergy=thresholdEnergy, ntrigger=ntrigger)
        self.thresholdEnergy2 = thresholdEnergy2
        self.seqID = 1
        print("Initialization of CdTe Detector...")
        self.client = controls.remote_detector.RemoteDetector(IP, storage_path=self.storagePath, photon_energy=[self.thresholdEnergy, self.thresholdEnergy2])
        print("Threshold 1 set to", self.thresholdEnergy)
        print("Threshold 2 set to", self.thresholdEnergy2)

    def snap_one(self, expTime, saving=True):
        """Take one image with exposure time: expTime"""
        super(CdTe, self).snap_one(expTime)
        self.client.setNTrigger(1)
        self.client.arm()
        self.client.trigger(self.expTime)
        print('seqID:', self.seqID)
        if saving:
            self.save()

    def arm(self, expTime):
        """Arm CdTe detector with given exposure time."""
        super(CdTe, self).arm(expTime)
        self.client.arm()
        print('seqID:', self.seqID)

    def trigger(self):
        """Send trigger to CdTe detector."""
        self.client.trigger(self.expTime)

    def disarm(self):
        """Disarm CdTe detector."""
```

```
        self.client.disarm()

    def config_imgParams(self, ntrigger, nimages, nimages_per_file):
        """File saving with CdTe should be further explored. ntrigger is the only parameter at the moment that has an influence here."""
        super(CdTe, self).config_imgParams(ntrigger=ntrigger, nimages=nimages, nimages_per_file=nimages_per_file)
        self.client.setNTrigger(ntrigger)

    def config_energy(self, thresholdEnergy1, thresholdEnergy2):
        """Config energy thresholds for CdTe. This detector has no parameter photonEnergy."""
        super(CdTe, self).config_energy(None, thresholdEnergy1)
        self.thresholdEnergy2 = thresholdEnergy2
        self.client.set_energy_and_thresholds([self.thresholdEnergy, self.thresholdEnergy2])

    def save(self):
        """Save files in the master file convention with the sequence id."""
        self.client.save('master_' + str(self.seqID) + '.h5')
        self.seqID += 1

    def delete(self):
        """No such function necessary here."""
        pass

    def config_storage_path(self, storagePath):
        """Reconfigure where to store the images.

        Parameters:
        storagePath (str): where to store the images
        """
        super(CdTe, self).config_storage_path(storagePath)
        self.client.storage_path = self.storagePath

    def get_masterfile(self, sequence_id):
        """Return path to master file with specified sequence ID

        Parameters:
        sequence_id (int): which file to get path from

        Return:
        path (str): path to searched master file
        """
        contains_id = lambda x: ("master_" + str(sequence_id)) in x
        file = list(filter(os.path.isfile and contains_id, glob.glob(self.storagePath + '*')))
        return file[0]

    def master_to_array(self, file, threshold):
        """Extract dataset from master file."""

        with h5py.File(file, 'r') as f:
            arr = []
```

```python
            if threshold is 1:
                th = 'threshold_0'
            else:
                th = 'threshold_1'

            for i in list(f['entry']['data'][th].keys()):
                try:
                    arr.append(f['entry']['data'][th][i][()])
                except:
                    print("Some images might not have been added due to corrupted file links")
                    break
        return np.asarray(arr)

class PhotonicScience(Detector):
    """Instantiate this one if you want to use the Photonic Science detector."""

    def __init__(self, storagePath, IP='129.129.99.116', port=50000):
        """Initialize PhotonicScience detector."""
        super(PhotonicScience, self).__init__(IP=IP, storagePath=storagePath, photonEnergy=None)
        self.port = port
        self.seqID = 1
        filename = 'master_' + str(self.seqID)
        self.client = PS.PSCamera(host=self.IP, port=self.port, camera="FKGSense_bin1x1")
        self.client.CameraSetup(self.expTime, self.storagePath, filename, self.nimages)

    def config_energy(self, photonEnergy, thresholdEnergy):
        super(PhotonicScience, self).config_energy(photonEnergy, thresholdEnergy)
        # has this even a threshold?

    def config_imgParams(self, ntrigger, nimages, nimages_per_file):
        super(PhotonicScience, self).config_imgParams(ntrigger, nimages, nimages_per_file)

    def arm(self, expTime):
        super(PhotonicScience, self).arm(expTime)

    def disarm(self):
        pass

    def trigger(self):
        pass

    def snap_one(self, expTime):
        super(PhotonicScience, self).snap_one(expTime)

    def save(self):
        pass

    def delete(self):
        pass
```

List. B.2: `LabSetup.py`

```python
import time
import numpy as np
import numpy.fft as npfft
import matplotlib.pyplot as plt

import Detector as dect

import warnings
with warnings.catch_warnings():
    warnings.filterwarnings("ignore", category=RuntimeWarning)
    warnings.filterwarnings("ignore", category=UserWarning)
    from smaract_client_py3.channel_definition import *

class LabSetup:
    """Lab Setup implements methods that use all hardware parts.
    Initialise this class as soon as you need to use motors.
    Otherwise, the Detector.py file may be good enough.
    """

    def __init__(self, storagePath, detector, stepscanMotor=G2_TRX, mvsampleMotor=SAM_TRX, rotsampleMotor=SAM_ROTY, source_sample_d=None, sample_dect_d=None, vertcenter=None):
        self.stepscanMotor = stepscanMotor
        self.rotsampleMotor = rotsampleMotor
        self.mvsampleMotor = mvsampleMotor
        self.source_sample_d = source_sample_d
        self.sample_dect_d = sample_dect_d
        self.vertcenter = vertcenter
        if detector == 'Eiger':
            self.detector = dect.Eiger(storagePath=storagePath, init=True)
        elif detector == 'CdTe':
            self.detector = dect.CdTe(storagePath=storagePath)
        elif detector == 'PhotonicScience':
            self.detector = dect.PhotonicScience(storagePath=storagePath)
        # add different detectors here
        else:
            print("no such detector")

    def sys_check(self, sample_in, sample_out):
        """Perform system check."""
        input("Check the detector parameters and press y (yes) to continue...")

        self.rotsampleMotor.put(90, wait=True)
        input("Check movement of sample rotation motor and press y (yes) to continue...")
        self.rotsampleMotor.put(0, wait=True)

        self.mvsampleMotor.mv(sample_out)
        input("Check if sample out of the beam and press y (yes) to continue...")
        self.mvsampleMotor.mv(sample_in)
```

```python

        self.stepscanMotor.mvr(6000)
        input("Check if stepscan motor moved and press y (yes) to continue...")
        self.stepscanMotor.mvr(-6000)

        y = input("Check if x-rays are on and press y (yes) to continue")
        print(y)

        return y

    def create_logfile(self, threshold, no_proj, angles_start, angles_end, expTime):
        """Create a log file for a tomography."""
        f = open(self.detector.storagePath + "logfile.txt", "a+")
        f.write("Threshold: " + str(threshold) + "\n")
        f.write("Exposure Time: " + str(expTime) + "\n")
        f.write("Start Angle: " + str(angles_start) + "\n")
        f.write("End Angle: " + str(angles_end) + "\n")
        f.write("Number of Projections: " + str(no_proj) + "\n")
        if self.source_sample_d is None:
            source_sample_d = input("Source-sample distance for this setup has not yet been entered. Enter source-to-sample distance in mm:")
            self.source_sample_d = source_sample_d
        f.write("Source to Sample (rotaxis): " + str(self.source_sample_d) + "mm" + "\n")

        if self.sample_dect_d is None:
            sample_dect_d = input("Enter origin-to-detector distance in mm:")
            self.sample_dect_d = sample_dect_d
        f.write("Sample (rotaxis) to Detector: " + str(self.sample_dect_d) + "mm" + "\n")

        f.close()


    def absorp_tomo(self, storagePath, threshold, sample_in, sample_out, angles_degrees, expTime, source_sample_d=None, sample_dect_d=None, waitTime=None, sys_check=True):
        """Perform pure absorption tomography.

        Parameters:
        storagePath (str): where to save tomography
        threshold (str): set detector threshold
        sample_in (int): motor position when sample in the beam
        sample_out (int): motor position when sample out of beam
        angles_degrees (numpy array): angles where projections should be taken
        expTime (int): exposure time for the projections in seconds
        waitTime (int): default None, possible wait time for tube to warm up
        """
        self.detector.reinitialize()
        self.detector.config_imgParams(nimages_per_file=5, ntrigger=100, nimages=1)
        self.detector.config_storage_path(storagePath)
        self.detector.config_energy(self.detector.photonEnergy, threshold)
        self.detector.delete()

```

```
        if sys_check:
            y = self.sys_check(sample_in, sample_out)
            if y is not 'y':
                print("Exiting tomography...")
                return

        self.create_logfile(threshold, angles_degrees.shape[0], angles_degrees[0],
    angles_degrees[-1], expTime)

        if waitTime is not None:
            time.sleep(waitTime)

        time1 = time.time()

        self.rotsampleMotor.put(0, wait=True)

        # flatfield
        self.detector.config_imgParams(nimages_per_file=1, ntrigger=1, nimages=1)
        self.mvsampleMotor.mv(sample_out)
        self.detector.snap_one(expTime)

        # tomography
        self.detector.config_imgParams(nimages_per_file=5, ntrigger=100, nimages=1)
        self.mvsampleMotor.mv(sample_in)

        for i, angle in enumerate(angles_degrees):
            print("Scanning at angle: ", angle)
            self.rotsampleMotor.put(angle, wait=True)

            if i%100 == 0:
                self.detector.arm(expTime)
                print("at image ", i, ", newly armed")

            self.detector.trigger()

            if i%100 == 99 or i == len(angles_degrees)-1:
                self.detector.disarm()
                print("at image ", i, ", newly disarmed")
                self.detector.save()
                self.detector.delete()

        self.rotsampleMotor.put(0, wait=True)

        # flatfield
        self.detector.config_imgParams(nimages_per_file=1, ntrigger=1, nimages=1)
        self.mvsampleMotor.mv(sample_out)
        self.detector.snap_one(expTime)

        self.mvsampleMotor.mv(sample_in)

        time2 = time.time()
        timetot = time2 - time1
        print("Scan took ", timetot/60, " minutes")

```

```
        f = open(self.detector.storagePath + "logfile.txt", "a+")
        f.write("Acquisition Time: " + str(timetot) + "\n")
        f.close()

    def stepscan_tomo(self, storagePath, threshold, sample_in, sample_out,
    angles_degrees, expTime, stepsize, source_sample_d=None, sample_dect_d=None,
    waitTime=None, sys_check=True):
        """Perform stepscan tomography.

        Parameters:
        storagePath (str): where to save tomography
        threshold (str): set detector threshold
        sample_in (int): motor position when sample in the beam
        sample_out (int): motor position when sample out of beam
        angles_degrees (numpy array): angles where projections should be taken
        expTime (int): exposure time for the projections in seconds
        stepsize (float): step size in um
        waitTime (int): default None, possible wait time for tube to warm up
        sys_check (bool): default True; lead through system inspection
        """
        self.detector.reinitialize()
        self.detector.config_imgParams(nimages_per_file=5, ntrigger=100, nimages=1)
        self.detector.config_storage_path(storagePath)
        self.detector.config_energy(self.detector.photonEnergy, threshold)
        self.detector.delete()

        if sys_check:
            y = self.sys_check(sample_in, sample_out)
            if y is not 'y':
                print("Exiting tomography...")
                return

        self.create_logfile(threshold, angles_degrees.shape[0], angles_degrees[0],
    angles_degrees[-1], expTime)

        if waitTime is not None:
            time.sleep(waitTime)

        time1 = time.time()

        self.rotsampleMotor.put(0, wait=True)

        # flatfield
        self.detector.config_imgParams(nimages_per_file=5, ntrigger=5, nimages=1)
        self.mvsampleMotor.mv(sample_out)
        self.step_scan(5, stepsize, expTime)

        # tomography
        self.detector.config_imgParams(nimages_per_file=5, ntrigger=100, nimages=1)
        self.mvsampleMotor.mv(sample_in)

        for i, angle in enumerate(angles_degrees):
            print("Scanning at angle: ", angle)
            self.rotsampleMotor.put(angle, wait=True)
```

```
            if i%20 == 0:
                self.detector.arm(expTime)
                print("at image ", i, ", newly armed")

            self.step_scan(5, stepsize, expTime, saving=False)

            if i%20 == 19 or i == len(angles_degrees)-1:
                self.detector.disarm()
                print("at image ", i, ", newly disarmed")
                self.detector.save()
                self.detector.delete()

        self.rotsampleMotor.put(0, wait=True)

        # flatfield
        self.detector.config_imgParams(nimages_per_file=5, ntrigger=5, nimages=1)
        self.mvsampleMotor.mv(sample_out)
        self.step_scan(5, stepsize, expTime)

        self.mvsampleMotor.mv(sample_in)

        time2 = time.time()
        timetot = time2 - time1
        print("Scan took ", timetot/60, " minutes")

        f = open(self.detector.storagePath + "logfile.txt", "a+")
        f.write("Acquisition Time: " + str(timetot) + "\n")
        f.close()

    def step_scan(self, n_steps, step, expTime, saving=True, storagePath=None):
        """Perform a single step scan.

        Parameters:
        n_steps (int): number of steps over one period
        step (float): the step size in um
        expTime (int): exposure time of single image
        saving (bool): saves step scan by default to the detector storage path
        storagePath (str): default is detector storage path
        """
        self.detector.config_imgParams(nimages_per_file=n_steps, ntrigger=n_steps,
    nimages=1)
        init_pos = self.stepscanMotor.get_position()
        if storagePath is not None:
            old_path = self.detector.storagePath
            self.detector.config_storage_path(storagePath)

        if saving:
            print("Scan will be saved to: " + self.detector.storagePath)
            self.detector.arm(expTime)
        for i in range(n_steps):
            self.detector.trigger()
            self.stepscanMotor.mvr(step)
        if saving:
```

```
            self.detector.disarm()
            self.detector.save()
            self.detector.delete()
        self.stepscanMotor.mv(init_pos)

        # reset defaults
        self.detector.config_imgParams(nimages_per_file=5, ntrigger=100, nimages=1)
        if storagePath is not None:
            self.detector.config_storage_path(old_path)

    # these two functions are so far only experimental, they have not been tested yet
    def take_ff(self, before_position):
        self.rotsampleMotor.put(0, wait=True)
        self.detector.config_imgParams(nimages_per_file=5, ntrigger=5, nimages=1)
        self.mvsampleMotor.mv(sample_out)
        self.step_scan(5, stepsize, expTime)
        self.mvsampleMotor.mv(sample_in)
        self.rotsampleMotor.put(before_position, wait=True)

    def stepscan_tomo_flatfield(self, storagePath, threshold, sample_in, sample_out, angles_degrees, expTime, stepsize, source_sample_d=None, sample_dect_d=None, waitTime=None, sys_check=True):
        self.detector.reinitialize()
        self.detector.config_imgParams(nimages_per_file=5, ntrigger=100, nimages=1)
        self.detector.config_storage_path(storagePath)
        self.detector.config_energy(self.detector.photonEnergy, threshold)
        self.detector.delete()

        if sys_check:
            y = self.sys_check(sample_in, sample_out)
            if y is not 'y':
                print("Exiting tomography...")
                return

        self.create_logfile(threshold, angles_degrees.shape[0], angles_degrees[0], angles_degrees[-1], expTime)

        if waitTime is not None:
            time.sleep(waitTime)

        time1 = time.time()

        self.rotsampleMotor.put(0, wait=True)

        # initial flatfield
        self.take_ff(0)

        # tomography settings
        self.detector.config_imgParams(nimages_per_file=5, ntrigger=100, nimages=1)
        self.mvsampleMotor.mv(sample_in)

        for i, angle in enumerate(angles_degrees):
            print("Scanning at angle: ", angle)
```

```python
            self.rotsampleMotor.put(angle, wait=True)

            if i%20 == 0:
                self.detector.arm(expTime)
                print("at image ", i, ", newly armed")

            self.step_scan(5, stepsize, expTime, saving=False)

            if i%20 == 19 or i == len(angles_degrees)-1:
                self.detector.disarm()
                print("at image ", i, ", newly disarmed")
                self.detector.save()
                self.detector.delete()
                self.take_ff(angle)

        time2 = time.time()
        timetot = time2 - time1
        print("Scan took ", timetot/60, " minutes")

        f = open(self.detector.storagePath + "logfile.txt", "a+")
        f.write("Acquisition Time: " + str(timetot) + "\n")
        f.close()

    # this remains here for the moment in order to keep old notebooks working
    # in principle you don't need it anymore
    def fba(self, data, periods=0):
        """Implementation of fba function."""
        data = np.transpose(data, axes=(1,2,0))
        fdata = npfft.fft(data)
        A = np.abs(fdata[:,:,0])
        B = np.abs(fdata[:,:,periods+1])
        P = np.angle(fdata[:,:,periods+1])
        return (A, B, P)
```

B.4. Data Processing Software

List. B.3: data_processing.py

```python
import time
import numpy as np
import numpy.fft as npfft
import matplotlib.pyplot as plt
import os
import sys
import glob
import h5py
import hdf5plugin
from scipy.signal import argrelextrema
import re

```

```
def plot_vismap(vis):
    """Plot visibility map of the scan.

    Parameters:
    vis (2D/3D nparray): visibility from a scan or the scan itself
    """
    if vis.ndim is 3:
        vis = visibility(vis)
    plt.figure(figsize=(10, 10))
    plt.colorbar(plt.imshow(vis, cmap ='gray'))
    plt.title("Visibility Map")
    plt.show()

def plot_phasemap(phase):
    """Plot phase of the scan.

    Parameters:
    phase (2D/3D nparray): Pref from a scan or the scan itself
    """
    if phase.ndim is 3:
        a0, a1, phase = fba(phase)
    plt.figure(figsize=(10, 10))
    plt.colorbar(plt.imshow(phase, cmap ='gray', vmin = -3.14, vmax = 3.14))
    plt.title("Phase Map")
    plt.show()

def plot_vishistogram(vis, peak=True):
    """Plot the visibility histogram.

    Parameters:
    vis (2D nparray): visibility of a scan
    peak (bool): plot the peak value
    """
    if vis.ndim == 3:
        vis = visibility(vis)

    if peak is True:
        height, visib = np.histogram(vis[vis<1], range=(0,0.3), bins=256)
        a = argrelextrema(height, comparator=np.greater, order=1000)
        peak_val = visib[a[0][0]]

    plt.hist(vis[vis<1], bins = 256, range=(0,0.4))

    if peak is True:
        plt.title("Visibility Histogram, peak: " + str(peak_val)[:5])
    elif peak is False:
        plt.title("Visibility Histogram")

    plt.show()

def visibility(flatfield):
    """Calculate visibility of 3D phase step set.

    Parameters:
```

```
    flatfield (3D nparray): stepscan data

    Return:
    visibility (2D nparray): visibility distribution
    """
    A0, A1, P1 = fba(flatfield)
    return 2*A1/A0

def angular_sensitivity(period, G1_G2, diff_phase, source_G1=1, source_sam=1):
    """Return angular sensitivy from diff phase image.

    For short setups: definitely make use of last two arguments

    Parameters:
    period (float): grating period in m
    G1_G2 (int): propagation distance in m
    diff_phase (2D np.array): area of diff phase image that should be looked at
    source_G1 (int): distance source-G1
    source_sam (int): distance source-sample
    """
    return period*source_G1 / (2*np.pi*G1_G2*source_sam) * np.std(diff_phase)

def fba(data):
    """Return Fourier coefficients.

    Parameters:
    data (3D nparray): stepscan data

    Return:
    (A0, A1, P1): Fourier coeff and phase needed for visibility and imaging
    """
    fdata = npfft.fft(data, axis=0)
    A0 = np.abs(fdata[0])
    A1 = np.abs(fdata[1])
    P1 = np.angle(fdata[1])
    return (A0, A1, P1)

def show(image, ROI=np.s_[:,:], vmin=None, vmax=None, figsize=(10, 10)):
    """Show an image.

    Parameters:
    image (numpy.array): 2D or 3D numpy array containing the image to be shown, 0
    will be shown for 3D
    ROI (numpy.s_): 2D numpy slice for zooming into specific region
    vmin (float): min data range
    vmax (float): max data range
    figsize (float, float): float tuple, size in inches (fuck imperial!)
    """
    if image.ndim is 3:
        image = image[0]
    plt.subplots(figsize=figsize)
    plt.imshow(image[ROI], cmap='gray', vmin=vmin, vmax=vmax)

def show_parallel(images, ROI=np.s_[:,:], vmin=None, vmax=None, figsize=(15,15)):
```

```
    """Show several images next to each other

    Parameters:
    image (numpy.array or list): 3D numpy array or list of 2D numpy arrays
    containing images to be shown
    ROI (numpy.s_): 2D numpy slice for zooming into specific region
    vmin (float): min data range
    vmax (float): max data range
    figsize (float, float): float tuple, size in inches (fuck imperial!)
    """
    if type(images) is list:
        no_plots = len(images)
    else:
        no_plots = images.shape[0]
    fig, axs = plt.subplots(1, no_plots, figsize=figsize)
    for i in range(no_plots):
        axs[i].imshow(images[i][ROI], cmap='gray', vmin=vmin, vmax=vmax)
    plt.show()

def remove_nans(data):
    """Set all NaN's from a dataset to 0."""

    nans = np.isnan(data)
    data[nans] = 0
    return data

def make_sino(data):
    """Make sinograms from projection data."""

    return np.transpose(data, axes=(1,0,2))


def extract_3_images(data, ff, amp=False, dpc=False, dci=False):
    """Convert dataset into absorption, differential phase contrast and dark field
    images.

    Currently uses only the first flatfield.

    Parameters:
    data (4D numpy array, 3D if single_step): contains phase steps of projections
        in case of 4D: 0 dimension - for different projections
                       1 dimension - same projection, step scan
                       2,3 - image itself
        in case of 3D: same as above for 1-3
    ff (4D numpy array, 3D if single_step): contains flatfield phase steps
    amp (bool): default True
    dpc (bool): default True
    dci (bool): default True

    Return:
    imgs (tuple): 3D numpy arrays containing the three respective sets of
    projections
    """
    if data.ndim is 3:
```

```
        A0ref, A1ref, P1ref = fba(ff)
        A0sam, A1sam, P1sam = fba(data)

        amp_arr = -np.log(A0sam/A0ref)
        dpc_arr = np.angle(np.exp(1j*(P1sam-P1ref)))
        dci_arr = -np.log(A1sam/A1ref*A0ref/A0sam)

        return amp_arr, dpc_arr, dci_arr

    if amp:
        amp_arr = np.zeros((data.shape[0],data.shape[2], data.shape[3]))
    if dpc:
        dpc_arr = np.zeros((data.shape[0],data.shape[2], data.shape[3]))
    if dci:
        dci_arr = np.zeros((data.shape[0],data.shape[2], data.shape[3]))

    # flatfield iteration to be implemented
    A0ref, A1ref, P1ref = fba(ff[0]) # currently only using the first flatfield

    for i in np.arange(data.shape[0]):
        A0sam, A1sam, P1sam = fba(data[i])

        if amp:
            amp_arr[i] = -np.log(A0sam/A0ref)
        if dpc:
            dpc_arr[i] = np.angle(np.exp(1j*(P1sam-P1ref)))
        if dci:
            dci_arr[i] = -np.log(A1sam/A1ref*A0ref/A0sam)

    imgs = []
    if amp:
        imgs.append(amp_arr)
    if dpc:
        imgs.append(dpc_arr)
    if dci:
        imgs.append(dci_arr)

    if len(imgs) is 1:
        return imgs[0]

    return tuple(imgs)


def get_masterlist(folder, master_max=None):
    """Create a list of masterfiles from folder. (EIGER ONLY)

    Parameters:
    folder (str): path to folder with masterfiles
    master_max (int): seqID of last file

    Returns:
    masters (list of str): list containing paths to masterfiles
    """
    if master_max is not None:
```

```
        masters = [None]*master_max
        for i in range(1, master_max+1):
            masters[i-1] = get_masterfile(folder, i)
    else:
        contains_master = lambda x: ("_master") in x
        path = os.path.join(folder, '')
        files = list(filter(os.path.isfile and contains_master, glob.glob(path + '*'
    )))
        masters = sorted(files, key = lambda x: int(re.findall(r'\d+', os.path.
    splitext(os.path.basename(x))[0])[0]))

    return masters

def get_masterfile(path, sequence_id):
    """Return path to master file with specified sequence ID. (EIGER ONLY)

    Parameters:
    path (str): path to folder which contains masterfiles
    sequence_id (int): which file to get path from

    Returns:
    file (str): path to masterfile
    """
    contains_id = lambda x: ("_" + str(sequence_id) + "_master") in x
    path = os.path.join(path, '')
    file = list(filter(os.path.isfile and contains_id, glob.glob(path + '*')))
    return file[0]

def master_3D(masters, no_proj, sino_slice=np.s_[:]):
    """Create 3D array from list of masterfiles. (EIGER ONLY)

    Parameters:
    masters (list of str): list of paths to sorted masterfiles
    no_proj (int): number of total projections
    sino_slice (np.s_ object): specify which sinograms in vert direction to load

    Returns:
    data (3D numpy array): array containing projections
    """
    with h5py.File(masters[0], 'r') as f:
        x = f['entry']['data']['data_000001'][()].shape[1]
        y = f['entry']['data']['data_000001'][()].shape[2]

    x = len(np.zeros(x)[sino_slice])
    data = np.zeros((no_proj, x , y))

    enumerator = 0

    for master in masters:
        with h5py.File(master, 'r') as f:
            for datafile in list(f['entry']['data'].keys()):
                try:
                    data[enumerator:enumerator+5] = f['entry']['data'][datafile][()
    ][:][sino_slice]
```

```
                    enumerator += 5
                except:
                    print("dangerous situation, check ", master, datafile)
                    break

    return data

def master_4D(masters, no_of_projections, sino_slice=np.s_[:]):
    """Create 4D array from list of masterfiles. (EIGER ONLY)

    Parameters:
    masters (list of str): list of paths to sorted masterfiles
    no_proj (int): number of total projections
    sino_slice (np.s_ object): specify which sinograms in vert direction to load

    Returns:
    data (4D numpy array): array containing phase steps of projections
    """
    mylist = []
    for master in masters:
        with h5py.File(master, 'r') as f:
            for datafile in list(f['entry']['data'].keys()):
                try:
                    mylist.append(f['entry']['data'][datafile][()][:, sino_slice])
                except:
                    print("dangerous situation, check ", master, datafile)
                    break
    mylist = np.asarray(mylist)
    if mylist.shape[0] == no_of_projections:
        return mylist
    else:
        print("Check if everything has been added")
        return mylist

def master_to_array(file):
    """Return 3D numpy array of images stored in master file. (EIGER ONLY)

    Parameters:
    file (str): path to h5 master file containing images
    onylfirst (bool): helpful if only one set of phase steps needed

    Returns:
    arr (3D numpy array): dataset from this file
    """
    arr = []
    with h5py.File(file, 'r') as f:
        for i in list(f['entry']['data'].keys()):
            try:
                arr.append(f['entry']['data'][i][()])
            except:
                print("Some images might not have been added due to corrupted file links")
                break
    return np.asarray(arr)[0]
```

List. B.4: `Reconstruction.py`

```python
import astra
import numpy as np

class Reconstruction:
    def __init__(self, geometry, algorithm, rotcenter, angles, source_origin, origin_detector, pixel_size, data, filtertype=None, vertcenter=None, phase_contrast=False):
        """Create new instance of Tomographic Reconstruction.

        Parameters:
        geometry (str): 'fanflat' or 'cone'
        algorithm (str): 'SIRT3D_CUDA', 'FDK_CUDA' for cone; 'FBP_CUDA', 'SIRT_CUDA' for fanflat; see astra-toolbox.com
        rotcenter (float): pixel specifying rotation center from left side of projection
        angles (numpy array): listing the projection angles in radians
        source_origin (float): distance in mm
        origin_detector (float): distance in mm
        pixel_size (float): size in mm
        data (3D numpy array): sinogram data
        filtertype (str): default 'ram-lak', check astra-toolbox.com
        vertcenter (float): pixel specifying center of cone beam from top (IMPORTANT: center sino must be part of dataset)
        phase_contrast (bool): default False, set True if reconstructing DPC images
        """
        self.geometry = geometry
        self.algorithm = algorithm
        self.rotcenter = rotcenter
        self.vertcenter = vertcenter
        self.angles = angles
        self.source_origin = source_origin
        self.origin_detector = origin_detector
        self.pixel_size = pixel_size
        self.data = data
        self.detector_rows = self.data.shape[0]
        self.detector_cols = self.data.shape[2]
        self.filtertype = filtertype

        sino_center = self.data.shape[2] / 2
        off_center = sino_center - self.rotcenter

        if vertcenter is not None:
            proj_center = self.data.shape[0] / 2
            vert_offcenter = proj_center - self.vertcenter

        # implement 2D geometries here
        if self.geometry is 'fanflat':
            self.algos = []
            for sino in np.arange(self.data.shape[0]):
                proj_geom = astra.create_proj_geom(self.geometry, 1, self.
```

```
detector_cols, self.angles, (self.source_origin + self.origin_detector) / self.pixel_size, 0)
                proj_geom = astra.functions.geom_postalignment(proj_geom, off_center)

                vol_geom = astra.creators.create_vol_geom(self.detector_cols)

                projections_id = astra.data2d.create('-sino', proj_geom, self.data[sino])

                reconstruction_id = astra.data2d.create('-vol', vol_geom, data=0)

                alg_cfg = astra.astra_dict(self.algorithm)
                alg_cfg['ProjectionDataId'] = projections_id
                alg_cfg['ReconstructionDataId'] = reconstruction_id
                if self.algorithm is 'FBP_CUDA':
                    alg_cfg['option'] = {'FilterType': self.filtertype}

                if self.algorithm is 'SIRT_CUDA':
                    alg_cfg['option'] = {'MinConstraint': 0}

                if phase_contrast:
                    alg_cfg['option'] = {'FilterType': 'Hilbert'}

                algorithm_id = astra.algorithm.create(alg_cfg)

                self.algos.append([proj_geom, vol_geom, projections_id, reconstruction_id, algorithm_id])

        # implement 3D geometries here
        if self.geometry is 'cone':
            self.proj_geom = astra.create_proj_geom('cone', 1, 1, self.detector_rows, self.detector_cols, self.angles, (self.source_origin + self.origin_detector) / self.pixel_size, 0)
            self.proj_geom = astra.functions.geom_postalignment(self.proj_geom, [off_center, vert_offcenter])

            self.projections_id = astra.data3d.create('-sino', self.proj_geom, self.data)

            self.vol_geom = astra.creators.create_vol_geom(self.detector_cols, self.detector_cols, self.detector_rows)

            self.reconstruction_id = astra.data3d.create('-vol', self.vol_geom, data=0)

            self.alg_cfg = astra.astra_dict(self.algorithm)
            self.alg_cfg['ProjectionDataId'] = self.projections_id
            self.alg_cfg['ReconstructionDataId'] = self.reconstruction_id
            if self.algorithm is 'FDK_CUDA':
                self.alg_cfg['option'] = {'FilterType': self.filtertype}

            if self.algorithm is 'SIRT3D_CUDA':
                self.alg_cfg['option'] = {'MinConstraint': 0}
```

```python

            if phase_contrast:
                self.alg_cfg['option'] = {'FilterType': 'Hilbert'}

            self.algorithm_id = astra.algorithm.create(self.alg_cfg)

    def run(self, iterations=None):
        """Run the created algorithm. Specify number of iterations for SIRT"""

        if self.algorithm is 'SIRT3D_CUDA':
            astra.algorithm.run(self.algorithm_id, iterations)
            recon = astra.data3d.get(self.reconstruction_id)
        elif self.algorithm is 'FDK_CUDA' or self.algorithm is 'BP3D_CUDA':
            astra.algorithm.run(self.algorithm_id)
            recon = astra.data3d.get(self.reconstruction_id)
        else:
            recon = []
            for sino in self.algos:
                proj_geom = sino[0]
                vol_geom = sino[1]
                projections_id = sino[2]
                reconstruction_id = sino[3]
                algorithm_id = sino[4]
                if self.algorithm is 'SIRT_CUDA':
                    astra.algorithm.run(algorithm_id, iterations)
                else:
                    astra.algorithm.run(algorithm_id)
                recon.append(astra.data2d.get(reconstruction_id))
            recon = np.asarray(recon)
        return recon

def project_phantom(angles, source_origin, origin_detector, pixel_size, phantom,
    rotcenter=None, vertcenter=None):
    """Create a set of projections using a cone beam shaped projector"""

    det_rows = phantom.shape[0]
    det_cols = phantom.shape[1]

    if rotcenter is not None:
        sino_center = phantom.shape[1] / 2
        off_center = sino_center - rotcenter
    else:
        off_center = 0

    if vertcenter is not None:
        proj_center = phantom.shape[0] / 2
        vert_offcenter = proj_center - vertcenter
    else:
        vert_offcenter = 0


    vol_geom = astra.creators.create_vol_geom(det_cols, det_cols, det_rows)

    phantom_id = astra.data3d.create('-vol', vol_geom, data=phantom)
```

```

    proj_geom = astra.create_proj_geom('cone', 1, 1, det_rows, det_rows, angles, (
    source_origin + origin_detector) / pixel_size, 0)
    proj_geom = astra.functions.geom_postalignment(proj_geom, [off_center,
    vert_offcenter])

    projections_id, projections = astra.creators.create_sino3d_gpu(phantom_id,
    proj_geom, vol_geom)

    projections = np.transpose(projections, axes=(1, 0, 2))
    return projections
```

References

[1] University of Calgary, ed. *CT Image Reconstruction.* URL: `http://199.116.233.101/index.php?title=CT_Image_Reconstruction#Sinogram_and_Radon_transform` (visited on 05/22/2021).

[2] William B. Case et al. "Realization of optical carpets in the Talbot and Talbot-Lau configurations". In: *Opt. Express* 17.23 (Nov. 2009), pp. 20966–20974. DOI: `10.1364/OE.17.020966`. URL: `http://www.opticsexpress.org/abstract.cfm?URI=oe-17-23-20966`.

[3] Tilman Donath et al. "Inverse geometry for grating-based x-ray phase-contrast imaging". In: *Journal of Applied Physics* 106 (Oct. 2009), pp. 054703–054703. DOI: `10.1063/1.3208052`.

[4] M Engelhardt et al. "The fractional Talbot effect in differential x-ray phase-contrast imaging for extended and polychromatic x-ray sources". In: *Journal of microscopy* 232 (Nov. 2008), pp. 145–57. DOI: `10.1111/j.1365-2818.2008.02072.x`.

[5] Aaron Filler. "The History, Development and Impact of Computed Imaging in Neurological Diagnosis and Neurosurgery: CT, MRI, and DTI". In: *Internet Journal of Neurosurgery* 7 (May 2010), pp. 1–85. DOI: `10.1038/npre.2009.3267.5`.

[6] Gabor T. Herman. *Fundamentals of Computerized Tomography, Image Reconstruction from Projections.* Second Edition. Springer-Verlag London Limited, 2009.

[7] Gabor T. Herman. *Fundamentals of Computerized Tomography, Image Reconstruction from Projections.* 2009.

[8] Dectris Ltd., ed. *EIGER2 R for Laboratory.* URL: `https://www.dectris.com/products/eiger2/eiger2-r-for-laboratory/` (visited on 05/19/2021).

[9] Dectris Ltd., ed. *User Manual Eiger R/X Detector Systems.* URL: `https://media.dectris.com/UserManual_EIGER.pdf` (visited on 05/05/2021).

[10] Daniël M. Pelt et al. "Integration of TomoPy and the ASTRA toolbox for advanced processing and reconstruction of tomographic synchrotron data". In: *Journal of Synchrotron Radiation* 23.3 (May 2016), pp. 842–849. DOI: `10.1107/S1600577516005658`. URL: `https://doi.org/10.1107/S1600577516005658`.

[11] Franz Pfeiffer et al. "Phase retrieval and differential phase-contrast imaging with low-brilliance X-ray sources". In: *Nature Physics* 2.4 (Apr. 2006), pp. 258–261. ISSN: 1745-2481. DOI: 10.1038/nphys265. URL: https://doi.org/10.1038/nphys265.

[12] Lord Rayleigh. *The London, Edinburgh and Dublin Philosophical Magazine and Journal of Science*. XXV. On Copying Diffraction-gratings, and some Phenomena connnected therewith. Taylor & Francis, 1881. URL: https://books.google.ch/books?id=O5EOAAAAIAAJ.

[13] Tom Roelandts, ed. *Tomography, Part 3: Reconstruction*. URL: https://tomroelandts.com/articles/tomography-part-3-reconstruction (visited on 05/18/2021).

[14] Tom Roelandts, ed. *Tomography, Part 3: Reconstruction*. URL: https://tomroelandts.com/articles/tomography-part-3-reconstruction (visited on 05/18/2021).

[15] A. Snigirev et al. "On the possibilities of x-ray phase contrast microimaging by coherent high-energy synchrotron radiation". In: *Review of Scientific Instruments* 66.12 (1995), pp. 5486–5492. DOI: 10.1063/1.1146073. eprint: https://doi.org/10.1063/1.1146073. URL: https://doi.org/10.1063/1.1146073.

[16] Thomas J. Suleski. "Generation of Lohmann images from binary-phase Talbot array illuminators". In: *Appl. Opt.* 36.20 (July 1997), pp. 4686–4691. DOI: 10.1364/AO.36.004686. URL: http://ao.osa.org/abstract.cfm?URI=ao-36-20-4686.

[17] T. Thuering and M. Stampanoni. "Performance and optimization of X-ray grating interferometry". In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 372.2010 (2014), p. 20130027. DOI: 10.1098/rsta.2013.0027. eprint: https://royalsocietypublishing.org/doi/pdf/10.1098/rsta.2013.0027. URL: https://royalsocietypublishing.org/doi/abs/10.1098/rsta.2013.0027.

[18] Thomas Thuering et al. "Sensitivity in X-ray grating interferometry on compact systems". In: *AIP Conference Proceedings* 1466.1 (2012), pp. 293–298. DOI: 10.1063/1.4742307. eprint: https://aip.scitation.org/doi/pdf/10.1063/1.4742307. URL: https://aip.scitation.org/doi/abs/10.1063/1.4742307.

[19] 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.

[20] Mathias Tomandl. "Realisierung von optischen Talbot- und Talbot-Lau-Teppichen". PhD thesis. 2013. URL: http://othes.univie.ac.at/9413/1/2010-04-22_0402570.pdf.

[21] Joan Vila-Comamala et al. “High sensitivity X-ray phase contrast imaging by laboratory grating-based interferometry at high Talbot order geometry”. In: *Opt. Express* 29.2 (Jan. 2021), pp. 2049–2064. DOI: 10.1364/OE.414174. URL: http://www.opticsexpress.org/abstract.cfm?URI=oe-29-2-2049.

[22] Timm Weitkamp et al. “Tomography with grating interferometers at low-brilliance sources”. In: *Developments in X-Ray Tomography V*. Ed. by Ulrich Bonse. Vol. 6318. International Society for Optics and Photonics. SPIE, 2006, pp. 249–258. DOI: 10.1117/12.683851. URL: https://doi.org/10.1117/12.683851.

[23] Timm Weitkamp et al. “X-ray phase imaging with a grating interferometer”. In: *Opt. Express* 13.16 (Aug. 2005), pp. 6296–6304. DOI: 10.1364/OPEX.13.006296. URL: http://www.opticsexpress.org/abstract.cfm?URI=oe-13-16-6296.

Image Sources

[1] University of Calgary, ed. *CT Image Reconstruction*. URL: http://199.116.233.101/index.php?title=CT_Image_Reconstruction#Sinogram_and_Radon_transform (visited on 05/22/2021).

[7] Gabor T. Herman. *Fundamentals of Computerized Tomography, Image Reconstruction from Projections*. 2009.

[13] Tom Roelandts, ed. *Tomography, Part 3: Reconstruction*. URL: https://tomroelandts.com/articles/tomography-part-3-reconstruction (visited on 05/18/2021).