Hard X-ray, Optical Transient Grating for Probing Ultrafast Dynamics in Solids

Jacob Feltman

A senior thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Bachelor of Science

Richard Sandberg, Advisor

Department of Physics and Astronomy

Brigham Young University

ABSTRACT

Hard X-ray, Optical Transient Grating for Probing Ultrafast Dynamics in Solids

Jacob Feltman
Department of Physics and Astronomy, BYU
Bachelor of Science

Femtosecond X-ray pulses give access to timescales faster than the best electronics, and spatial scales on the order of crystalline structures. The fastest electronics operate on the nanosecond timescale. The ability to modulate material properties on the femtosecond time scale could lead to a six-order of magnitude increase in computing speed. With X-ray free-electron laser (XFEL) facilities like the Linac Coherent Light Source (LCLS), femtosecond X-ray pulses are now accessible. However, XFEL sources experience X-ray jitter—a variation in temporal profile and arrival time of each pulse. This work focuses on experiments conducted at LCLS operating at 10 keV (0.12 nm), purposed to study the X-ray light-matter interaction in BGO, YAG, and ZnO crystals, and to understand material response on the femtosecond timescale. X-ray-optical transient grating was demonstrated, solutions to X-ray jitter were implemented, THz range optical phonons were identified, and analysis of THz phonons was attempted with THz time-domain spectroscopy.

Keywords: X-Ray, Transient Grating, Ultrafast Spectroscopy, Optical Phonon, XFEL

ACKNOWLEDGMENTS

Thank you to Dr. Pam Bowlan and Dr. Prashant Padmanabhan for teaching me and allowing me to work on interesting problems with them. I would also like to thank Dr. Richard Sandberg for guiding me to a career in physics.

Use of the Linac Coherent Light Source (LCLS), SLAC National Accelerator Laboratory, is supported by the U.S. Department of Energy, Office of Science, Office of Basic Energy Sciences under Contract No. DE-AC02-76SF00515. Research presented in this article was supported by the Laboratory Directed Research and Development program of Los Alamos National Laboratory under project number 20180242ER.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Probing Small and Fast Scales

Conventional computers operate on the nanosecond timescale, or with clock speeds on the order of gigahertz. Thus in order to gather more computing power with conventional computers, scaling the size of computer chips is the simplest option. Thus we see massive computing power requiring large spaces and massive amounts of energy and cooling in the form of supercomputers [1]. Conventional computers work because we have been able to understand, study, and control the material properties of silicon on the nanosecond timescale and the spatial scale of the crystal structure. If we could understand, study, and control the material properties of any material on faster and smaller scales, it could lead to orders of magnitude increase in computing power while occupying less space.

Visible wavelengths and larger (> 400nm) have long been used to understand the physical world. These methods are well-understood and easy to use. However, visible methods only allow us to probe materials down to the scale of hundreds of nanometers. The best possible resolution with a given wavelength is given by the Rayleigh criterion:$\theta \approx 1.22\frac{\lambda}{D}$, where θ is the angular resolution, is the wavelength of light, and D is the diameter of the lens [2]. We can see with the Rayleigh

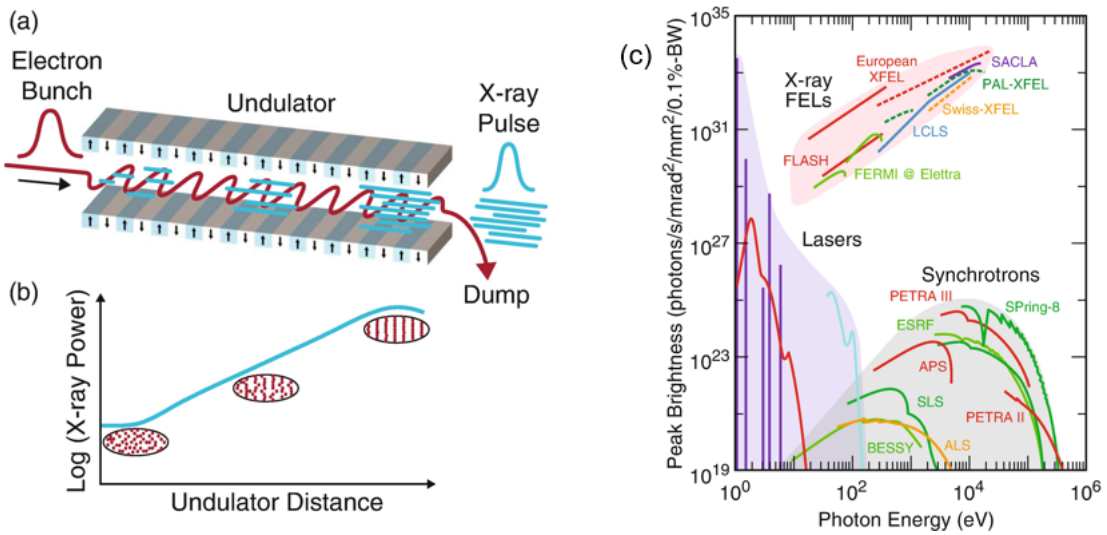

Figure 1.1 (a) XFELs create femtosecond X-ray pulses by accelerating an electron bunch through an undulator. The undulator is an array of magnets with alternating poles. (b) The X-ray power increases exponentially with undulator distance. (c) Many brilliant synchrotron and XFEL X-ray sources are in operation. This research was conducted at the Linac Coherent Light Source (LCLS) at 10 keV photon energy. This figure was originally produced by S. Boutet et al. (2018). [3]

criterion that if we want to probe below the scale of hundreds of nanometers, shorter than visible wavelengths–or higher energy photons–are required.

1.2 Femtosecond X-Ray Pulses are Now Accessible

A tool for probing and controlling smaller and faster timescales is femtosecond X-Ray pulses. A femtosecond is 10^{-15} seconds. To contextualize how fast a femtosecond really is let's compare it to a minute. There are as many femtoseconds in one second as there are minutes in the age of the earth. Light pulses on the order of femtoseconds are very fast events, and they occur six orders of magnitude faster than the nanosecond or conventional computing timescale (10^{-9} seconds).

In recent decades, femtosecond X-Ray pulses have become an accessible reality with the construction of X-Ray Free Electron Laser (XFEL) facilities. There are seven constructed XFEL

facilities in the world today [3] and more are coming online. The photon energy and brightness that these facilities are able to achieve is shown in Figure 1.1(c), on which we can see that high energy and high brightness X-ray lasers are achievable. X-ray pulses are created by accelerating an electron bunch through an undulator, which an array of magnets with regularly alternating poles (see Fig. 1.1(b)). The electron bunch wiggles as it traverses the undulator which begins to create coherent and femtosecond-pulsed X-Rays up to ~10^4 eV (0.01 nm). The brightness of these pulses is related to the length of the undulator (see Fig. 1.1(b)) which explains the $\sim$1 kilometer length of the Linac Coherent Light Source (LCLS) XFEL facility.

1.3 X-Ray Light-matter Interaction in Bulk Crystals

To begin understanding material properties on these fast and small scales, an experiment was conducted at the LCLS XFEL facility to study the X-Ray light-matter interaction in these bulk crystals: BGO (Bi12GeO20), YAG (Y3Al5O12) and Zinc Oxide (ZnO). This experiment was additionally purposed to demonstrate the ability to extend a well-known optical technique called Transient Grating (TG) into the X-ray regime. Note also that this experiment is one of only two to date which has been able to accomplish X-ray Transient Grating [4]. Specifically in this thesis, I describe my main contributions to the experiment which were the analysis of the data recorded at the LCLS XFEL facility and the implementation of some solutions to XFEL jitter (described in the Methods section).

1.4 Los Alamos Internship

My work on this research was the result of several internships at Los Alamos National Laboratory (LANL) in Los Alamos, NM. Dr. Pamela Bowlan in the LANL physical chemistry group gave me an amazing opportunity to work for her in the summer of 2021 remotely and also in the summer

of 2022 in person. The experiment at LCLS was conducted previous to my first summer, so the summer of 2021 consisted of learning how to access the massive amounts of recorded data on LCLS servers and converting that data into a more usable format. Analysis of that data had begun but was not completed at that time.

The summer of 2022 consisted of working more closely with Dr. Bowlan (as a result of my physical presence) on extracting reliable information from the recorded data and implementing solutions to XFEL jitter (described in the Methods section). The time also consisted of working for and learning from Dr. Prashant Padmanabhan at the Center for Integrated Nanotechnologies (CINT) at LANL and Sandia National Laboratory (SNL) in Albuquerque, NM in order to conduct experiments to confirm results from the LCLS XFEL experiment. The combined time working at LANL culminated in a paper that has been submitted and is currently under review for publication titled "Hard X-ray – optical four-wave mixing using a split-and-delay line" and authored by William K. Peters, Jacob Feltman, Travis Jones, Sanghoon Song, Matthieu Chollet, Joseph Robinson, Prashant Padmanabhan, Laura Foglia, Filippo Bencivenga, Ryan Coffee, and Pamela Bowlan [5].

The experience was a valuable one and also one that helped me understand what science is–not a class or a textbook with an exam, but rather a curious investigation.

Chapter 2

Methods

In this section, I will summarize the extension of the Transient Grating technique to the X-Ray regime, describe the analysis of the data recorded at the LCLS XCS hutch, and describe the method for correcting inconsistencies in the recorded data due to XFEL jitter.

2.1 X-Ray Transient Grating

In order to study the bulk crystal materials on the femtosecond and nanometer scales, a Transient Grating pump-probe method was employed with the LCLS XFEL producing pulses of 50 femtoseconds and photon energies of 10 keV (0.1 nm) at a 120 Hz repetition rate. The geometry of this particular transient grating experiment is shown in Figure 2.1. The pump, propagating from left to right in Figure 2.1, is an XFEL pulse which is directed into the novel X-Ray Split and Delay (SnD) line at LCLS [6]. Because designing and manufacturing optics for X-ray frequencies is difficult–a novel array of mirrors and beam-splitters is used in the SnD line [6]. The XFEL pulse is split into two pulses which gives the capability to introduce a time-delay between them. This is accomplished by moving the mirrors in Figure 2.1 to increase the path length for one of the pulses. The pulses can then be recombined and focused onto a sample where they interact with the material, hence the

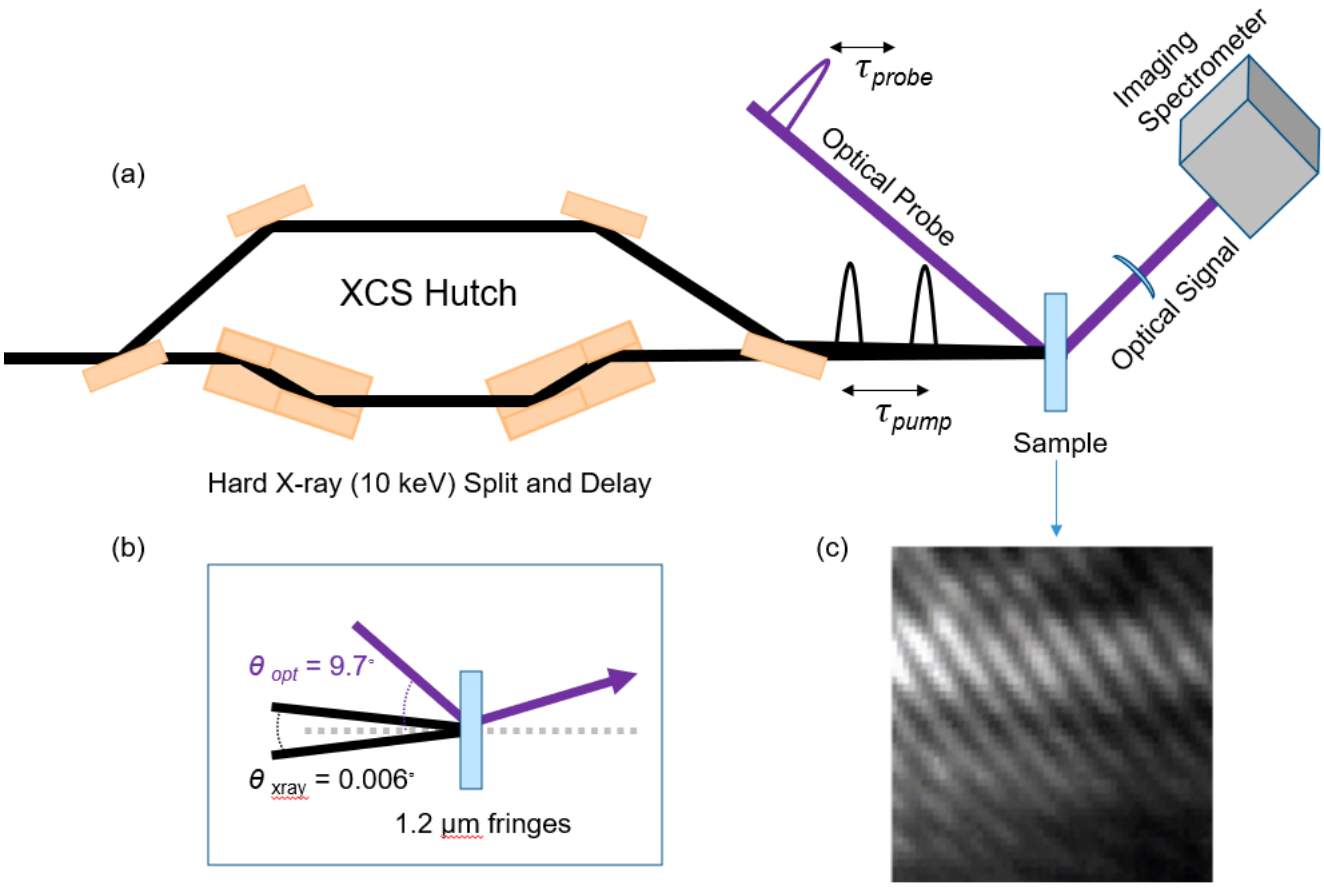

Figure 2.1 (a) The experiment was conducted at the LCLS XCS hutch. X-rays are copied and could be delayed in time with respect to each copy. The X-rays were directed to a sample where they recombined and created fringes. An optical probe was directed at these transient fringes and diffracted into a spectrometer. (b) A transient grating geometry with X-ray pulses at a small crossing angle, causing a transient fringe pattern to appear. The optical probe diffracts from the fringe pattern into a spectrometer. (c) An example of fringes caused by the interference of X-rays.

pulses act as a "pump" for the experiment. An additional beam, called the "probe", is an optical laser (Ti:Sapphire, 50 femtosecond pulse, 400 nm) which is also focused on the sample. The probe pulse can be delayed with respect to the pump pulses by changing the path length of the pulse with staged mirrors, similar to the pump. The light from the optical probe is then analyzed down-stream from the sample in an imaging spectrometer with an Andor Zyla detector.

In this particular experiment, the pump pulses were not delayed with respect to one another.

Instead, they were adjusted to arrive at the sample at the same time. Instead, the optical probe was delayed in time with respect to the arrival time of the XFEL pulse copies. This means that if data were taken at various time-delay values of the optical probe, then time-resolved spectrum information could be recorded by the imaging spectrometer downstream (see Fig. 2.1). The SnD line is designed so that the pump pulses can arrive at normal incidence to the surface of the sample, but rather than aligning the system for the pulses to arrive at normal incidence, the system was misaligned so that the pump pulses had a small crossing angle of $\sim 0.006^\circ$ at the sample. Due to the wave nature of light, this small crossing angle created micrometer-scale fringes on the sample similar to a diffraction grating (see fringes on a YAG screen in Fig. 2.1(c)). With the imaging spectrometer placed at the diffraction angle of this grating, the optical probe pulse then diffracts from the Transient Grating into the imaging spectrometer. This illuminates what the term Transient Grating means in simpler terms–a temporary grating. The diffraction-grating-like fringes at the sample are temporary, only existing after the pump pulses arrive, and eventually disappearing after the interaction is over. This also means the spectrometer, having been placed at the diffraction angle, only "sees" the diffracted probe when the grating is active, or in other words after the XFEL pulses have arrived. The X-ray TG technique then can give time-resolved spectral information about what is happening in the sample on the femtosecond timescale.

2.2 Data Analysis

With the X-Ray TG technique in mind, in order to understand the temporal and spectrum information due to the X-Ray light-matter interaction, several important points of data were recorded and analyzed.

2.2.1 Recorded Data

As previously mentioned and shown in Fig 2.1, the optical probe pulse was diffracted from the sample into a spectrometer at specific time-delay values with respect to the arrival time of XFEL pulses at the sample. The time-delay of the probe pulse is created by changing the path length traveled by the optical pulse via mirrors mounted on motorized stages. Thus the relative time-delay can be calculated via the relative path length difference between optical and XFEL pulses and the speed of light: $\tau = d/c$, where d is the relative path length difference. The spectrometer image in Figure 2.2 is an example of a typical image recorded at a particular time-delay. These stage locations were recorded along with intensity measurements of the incoming XFEL pulse via an intensity-position monitor (IPM) upstream from the experiment which allows for background subtraction, step averaging, and sorting (described in the following sections) [7].

2.2.2 X-Ray Jitter and the Time-Tool

Though femtosecond X-Ray pulses are accessible, they are not without issues. XFEL pulses suffer from what is called X-Ray Jitter. X-Ray Jitter is defined as the variation in pulse shape, spectral profile, and arrival time of each pulse. All three of these important properties could vary dramatically between XFEL pulses. Work done by N. Hartmann et al. is shown in Figure 2.3 where we can see the variation in spectral content and temporal profile of several XFEL pulses [8]. If femtosecond time-resolved information is what we are after, then this issue needs to be addressed.

To address the issue of X-Ray jitter, the LCLS XFEL facility has a novel instrument called a time-tool [9]. The instrument utilizes the fact that X-Rays change the way that some materials respond to light. More specifically, Si3N4 changes its transmission profile when X-Rays are present, thus the presence of X-Ray pulses can be detected by measuring the transmission of white light on a Si3N4 window. We can see this transmission change in Figure 2.4 [9]. The pixel position of the middle of the "edge" in Fig. 2.4 can be calibrated and mapped to an X-Ray pulse arrival

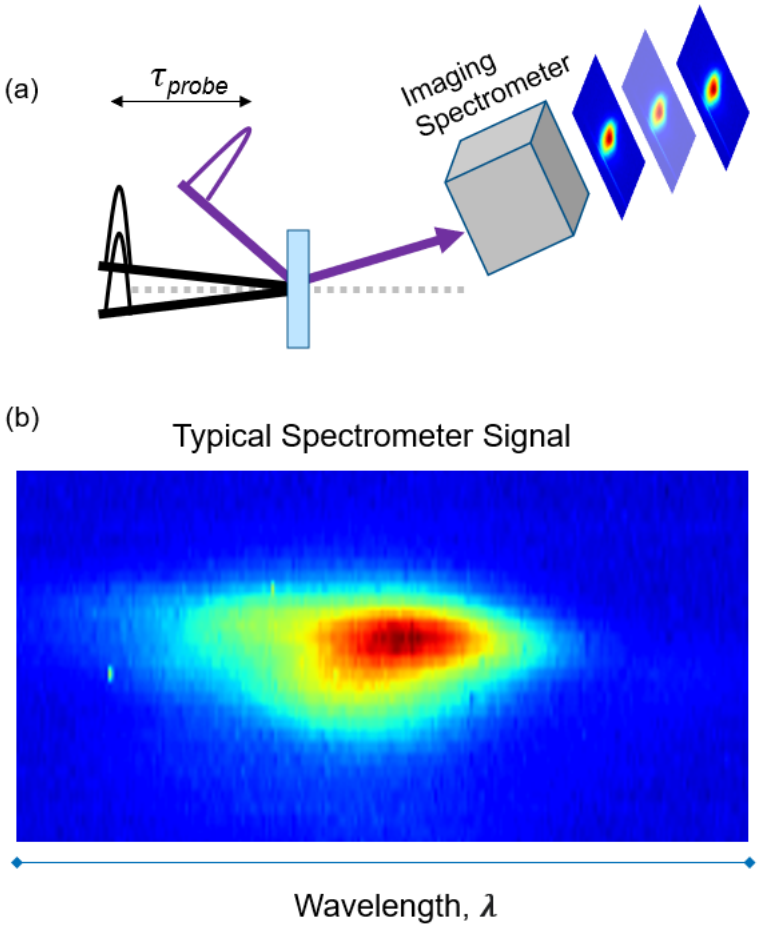

Figure 2.2 (a) Data was recorded by delaying the optical probe in time with respect to X-ray pulses. Intensity and spectral information from the diffracted probe signal were recorded at a spectrometer. Time-dependent variations of the diffracted optical probe intensity and spectrum indicate changes in the optical properties of the sample. (b) A typical diffracted optical probe signal at the spectrometer.

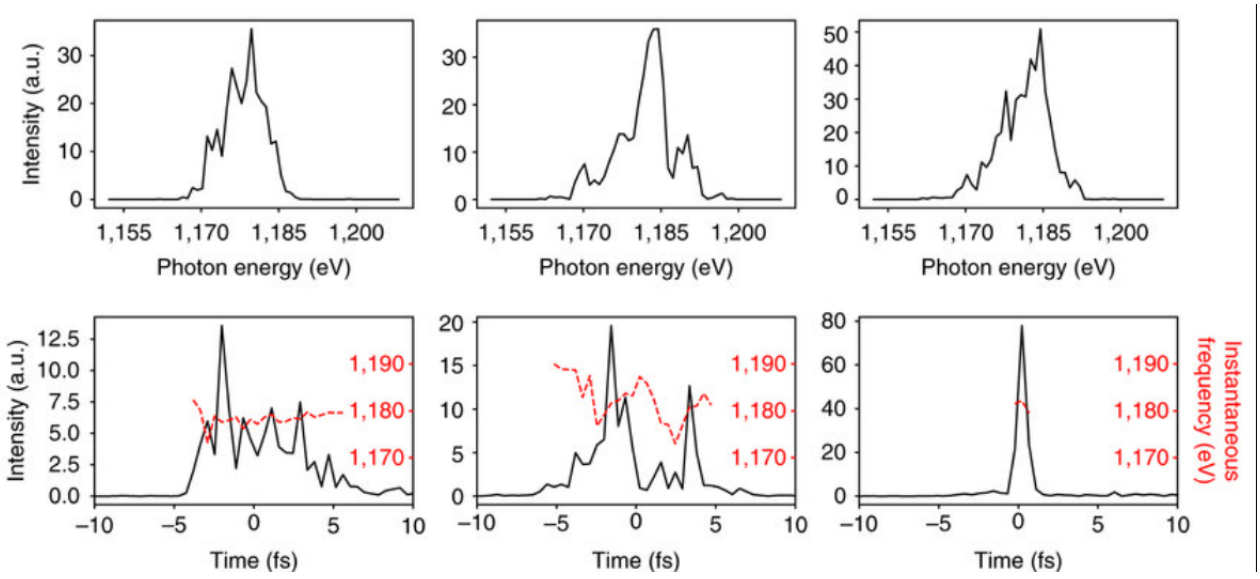

Figure 2.3 XFEL sources have an inherent X-ray "jitter" or significant variations in the temporal and spectral profiles of each femtosecond pulse. X-ray jitter also causes variations in the arrival time of each femtosecond pulse. In experiments where time-dependent information is important, there must be a solution to X-ray jitter. This figure was originally produced by N. Hartmann et al. (2018) [8].

time correction value, with an error less than 10 femtoseconds [9]. In practice, the instrument is placed upstream from the experiment where a portion of the XFEL beam is analyzed with this tool. For each pulse, the pixel positions of the "edge" seen in Fig. 2.4 are recorded and converted to a correction which can be used to re-sort the recorded data downstream in the correct order. This is an incredible technique considering that measuring the transmission of light through a material is considerably easier than the impossible task of seeing a femtosecond light pulse. Figure 2.4 also shows an example of how the time-resolved spectrum appears before and after the correction.

2.2.3 Plot Creation

Figure 2.3 shows shot-to-shot variations in pulse intensity, arrival time, and spectral content. Intense pulses are preferred for accurate time-tool measurements, thus thousands of spectrometer images were collected at each time-delay and characterized with an instrument upstream from the

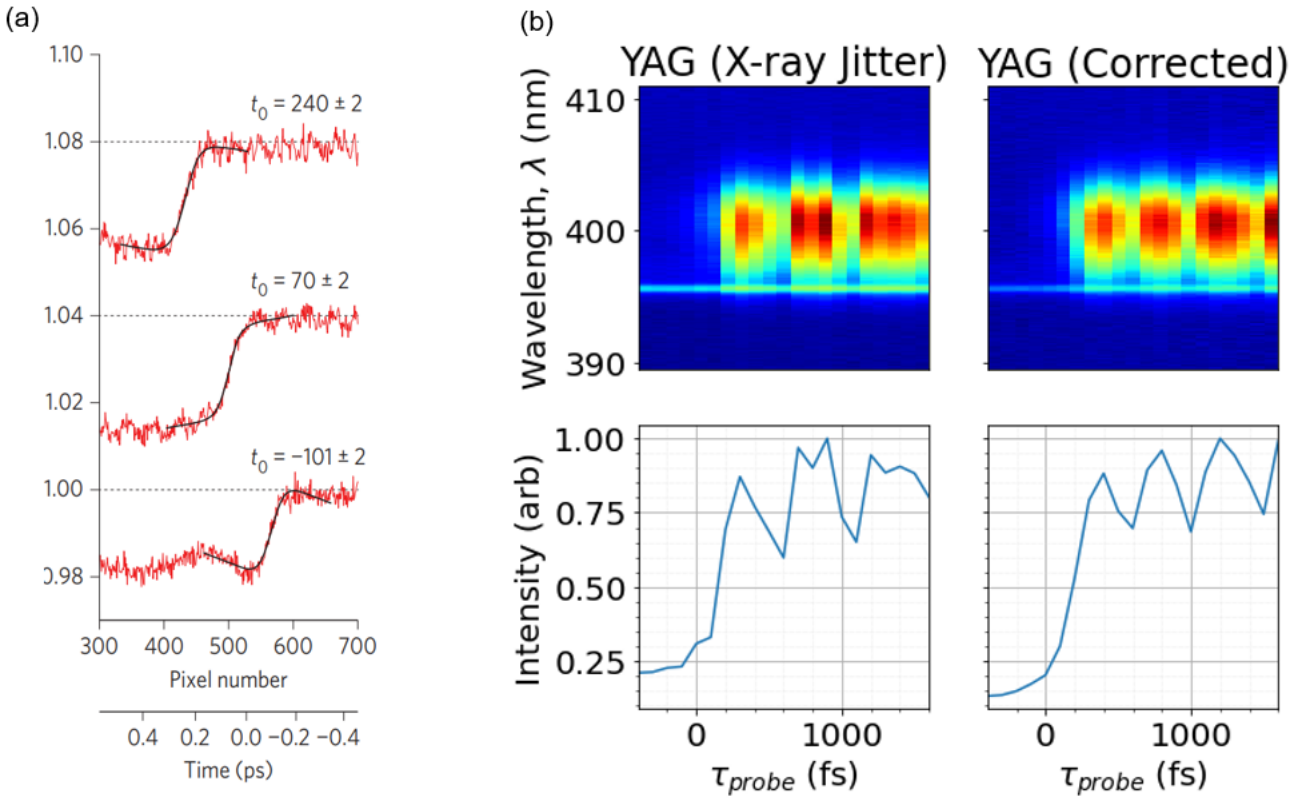

Figure 2.4 (a) Variations in the arrival time of X-ray pulses can be corrected with the LCLS timing tool upstream from the experiment. The presence of X-rays causes a change in transmissivity in SiNi. The location of this transmissivity change can be associated with a time correction [9]. The pixel position of the "edge" discussed in the Methods section is the center of the step function-like shape of these plots. (b) The LCLS timing tool corrects noise from X-ray jitter in the signal from a YAG sample.

experiment called the Intensity Position Monitor (IPM) [7].

All images and associated data were sorted into time-bins according to the arrival-time correction measured by the time-tool. The size of the time-bins depends on the resolution of the time-tool, which according to M. Harmond Et al. can be as low as a few femtoseconds [9], though larger bins (100 femtoseconds) were used in this experiment due to a high amount of noise.

Those spectrometer images which the IPM instrument characterized as low-intensity were labeled as the background of the image and were averaged together for each time-bin. The images with higher-intensity XFEL pulses were labeled as foreground and were also averaged together for each time-bin. The background was then removed from the foreground in each time-bin. What results from this analysis is a time-sorted, averaged, and background-subtracted spectrum with a temporal resolution of approximately 100 femtoseconds. This was accomplished by writing code in Python which is included in the appendix of this thesis.

It is important to note that the time-sorting algorithm developed in Python was tested and confirmed to be accurate by creating and operating on a simulated data set similar to the real LCLS XFEL data. Simulated spectrometer data (Fig. 2.5(a)) was created based on a typical spectrometer image (shown in Fig. 2.2 (b). X-ray-optical time-delay data was created and associated with each simulated image. The images and time-delay values were then placed together in random order with a new time-delay (Fig. 2.5(c)). The difference between the new and old time-delay values were recorded as the simulated time-tool correction. The sorting algorithm was then run on the simulated data to see if the original shape was recovered, which was indeed the case (Fig. 2.5(b)). The sorting algorithm was therefore demonstrated to be capable of sorting correctly.

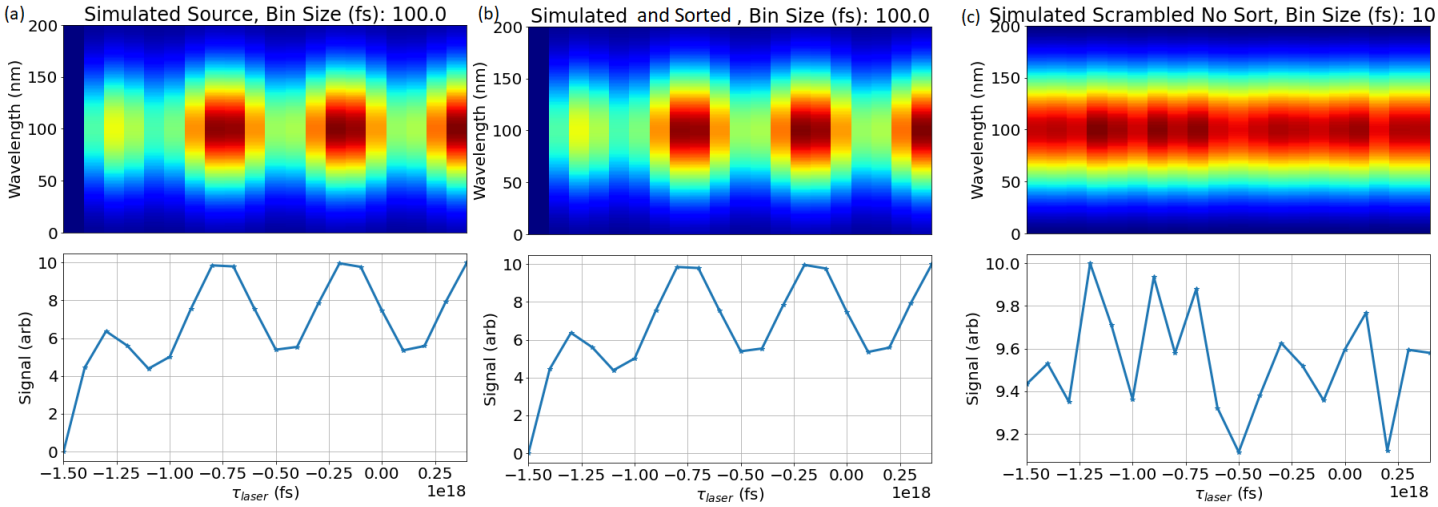

Figure 2.5 (a) Simulated data based on typical spectrometer images. (b) The resulting data from applying sorting code to (c). (c) Randomly sorted data from (a).

Chapter 3

Results and Discussion

The plots in Figure 3.1 show time-resolved spectrograms after background subtraction, averaging, and time-sorting. A periodic modulation of intensity can be seen in all three plots in Figure 3.1 which has a frequency of $\sim$2 THz. We have identified these periodic intensity modulations in each crystal as optical phonons [10].

Optical phonons are a collective optical excitation or vibration of the atoms in the crystal lattice [10]. An optical phonon is analogous to a bell. The geometry of the bell determines the frequency that it produces, whereas optical phonon frequencies are determined by the crystal lattice structure. In this case, the XFEL pulses would be similar to a mallet that strikes the bell to produce sound waves, and the periodic intensity modulations found in the optical probe would be similar to the frequency of the bell's acoustical waves.

There are indeed known phonon modes in the 2 THz range in the three crystals. BGO has modes at $\sim$2 and $\sim$2.5 THz [11], YAG at $\sim$2.5 THz [12], and ZnO at $\sim$3 THz [13]. Another recent X-ray Transient Grating experiment operating at 7 keV excited a 2.5 THz optical phonon in BGO [4]. In addition to the literature revealing these known phonon modes, we conducted a short Terahertz Time-Domain Spectroscopy experiment to confirm these phonon modes in BGO and YAG.

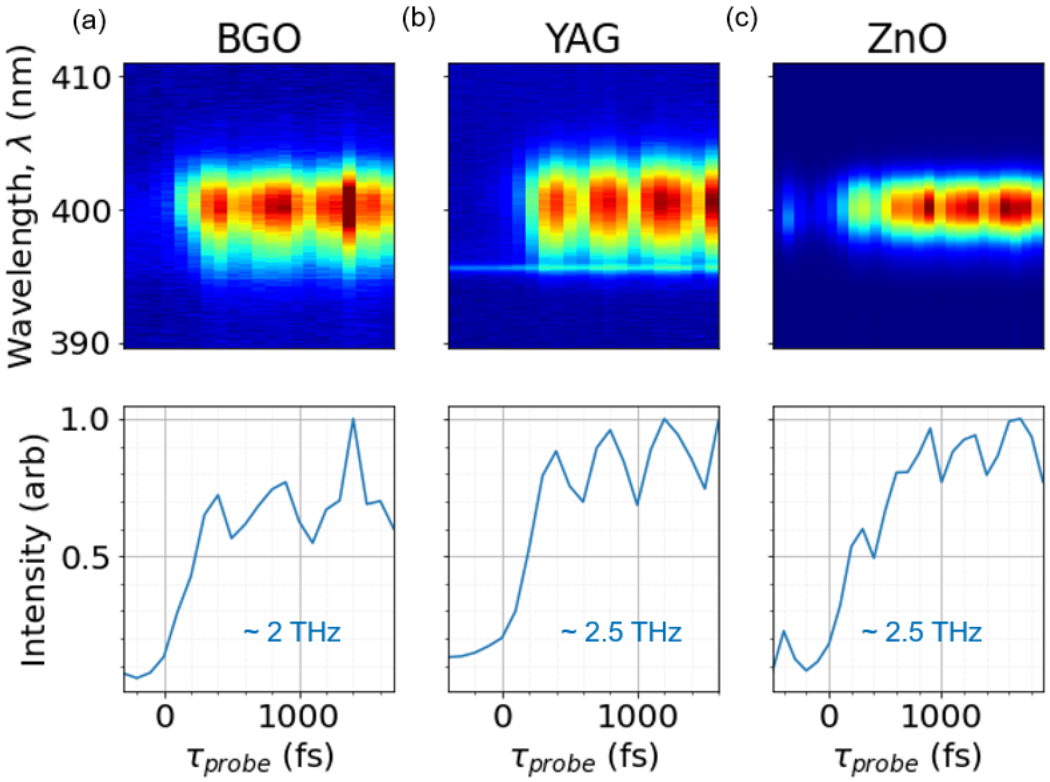

Figure 3.1 The top row of plots is generated by integrating the spectrometer images for each delay over the non-wavelength axis. The bottom row of plots is generated by integrating the entire spectrometer image for each delay. (a) A BGO sample shows an intensity modulation at the spectrometer of ∼2 THz. (b) A YAG sample shows an intensity modulation at the spectrometer of ∼2.5 THz. (c) A ZnO sample shows an intensity modulation at the spectrometer of ∼2.5 THz.

3.1 Next Steps

What is most interesting is that a similar frequency was excited in three very different materials. This may be like hitting three different bells with the same mallet and hearing roughly the same frequency. The question remains as to why this small frequency range has been excited. As a result of this question, suggestions for future work include a similar experiment with the added ability to adjust the polarization of the XFEL pump pulses. Some optical phonon modes are said to be fully symmetric, which activate with unpolarized light and may not activate with specific polarization configurations [14]. This additional ability to control XFEL pump polarization would give added information to understand exactly which phonon modes were activated to resolve these remaining questions. Adjusting XFEL polarization is non-trivial, yet the LCLS XFEL now has a Delta undulator in operation which provides a tunable polarization ability [15].

3.2 Conclusion

We have demonstrated the ability to successfully correct for X-ray jitter to understand material dynamics in BGO, YAG, and ZnO crystals on the femtosecond time-scale. We were also able to demonstrate the ability to extend Transient Grating into the X-Ray regime. This is an important step as the extension of TG into the X-ray regime is a requirement for applying a technique called Transient Grating Frequency-Resolved Optical Gating (TG FROG) to X-ray pulses. TG FROG is a method of characterizing the pulse duration, energy, and phase of ultrafast light-pulses [16]. As discussed in the Methods section XFELs experience X-ray jitter, thus pulse characterization is important for correct data interpretation, and working to apply TG FROG in the X-ray regime is valuable research.

Appendix A

X-ray Jitter Correction Code

I have written the following code in Python to use the time-tool data to correct for X-ray Jitter. It is designed to be run on LCLS servers.

```
"""
Author: Jake Feltman
Contact: jfeltman@lanl.gov
"""

import numpy as np
from matplotlib import pyplot as plt
import h5py
import pandas as pd
# from scipy.signal import savgol_filter


def characterize_run(runnumber):
    numsteps = 0
    numshots = 0

```

```
17      data = pd.read_csv('ScansTemplate_index_current.csv')
18      data_array = np.asarray(data)
19
20      # runindex = runnumber - 1
21      runindex = runnumber - 2
22
23      runtype = str(data_array[runindex, 1])
24      crystal = str(data_array[runindex, 8])
25      if crystal == "":
26          crystal = "UNKNOWN"
27      print("Run num from list:", data_array[runindex, 0])
28      print("Run type:", runtype)
29      print("Crystal type:", crystal)
30
31      return runtype, crystal
32
33  def show_image(array, title):
34      plt.figure(num=None, figsize=(15, 15))
35      plt.title(title)
36      # plt.pcolormesh(array, cmap='jet')
37      plt.imshow(array, aspect='auto', cmap='jet')
38      plt.colorbar(shrink=0.25)
39      plt.show()
40      return None
41
42
43  def show_plot_two_arg(ds1, ds2, title):
44      plt.figure(num=None, figsize=(15, 15))
45      plt.plot(ds1, ds2, 'c-', marker="None")
46      # plt.plot(ds1, ds2, '--', marker="None", color='.5')
```

```
47      plt.title(title)
48      plt.show()
49      return None
50
51  def show_plot(ds1, title):
52      plt.figure(num=None, figsize=(15, 15))
53      plt.plot(ds1, 'c-', marker="None")
54      # plt.plot(ds1, ds2, '--', marker="None", color='.5')
55      plt.title(title)
56      plt.show()
57      return None
58
59
60  def calc_step_shots(information_array_param, delta_t_param):
61
62      maxstep = np.max(information_array_param[:, 1])
63      minstep = np.min(information_array_param[:, 1])
64
65      step_range = np.abs(maxstep - minstep)
66      # if our defined step size doesn't divide into an integer, we need
        more bins
67      num_steps = int(np.ceil(step_range / delta_t_param) + 1) # FIXME: does
        this always work?
68      print("numsteps:", num_steps)
69
70      # initialize the step_shot_array
71      step_shot = np.zeros((num_steps, 2), dtype=float)
72
73      new_bins = np.linspace(minstep, maxstep, num=num_steps)
74      step_shot[:, 1] = new_bins
```

```
75
76      step_counter = 0
77      total_shots = 0
78
79
80      for i in range(step_shot.shape[0]):
81          if i == step_shot.shape[0] - 1:
82              print("last step!")
83              # this last step doesn't really have a "range". it's those
    shots at the last step value.
84              # for this reason I have adjusted the range to include one
    more bin in the condition to count the shots at the last delay value
85              # the extra bin is simply for counting and won't be added to
    the true bins
86              condition = np.logical_and((information_array_param[:, 3] >=
    step_shot[-1, 1]), (information_array_param[:, 3] < (step_shot[-1, 1] +
     delta_t_param)))
87              num_shots = np.count_nonzero(condition)
88              print("step range:", step_shot[-1, 1], step_shot[-1, 1] +
    delta_t_param)
89              print("num_shots:", num_shots, "\n")
90              total_shots += num_shots
91              step_shot[i, 0] = num_shots
92              num_shots = 0
93              information_array_param[np.where(condition), 5] = step_shot.
    shape[0] - 1
94          else:
95              condition = np.logical_and((information_array_param[:, 3] >=
    step_shot[i, 1]), (information_array_param[:, 3] < step_shot[i + 1, 1])
    )
```

```
            num_shots = np.count_nonzero(condition)
            print("step range:", step_shot[i, 1], step_shot[i + 1, 1])
            print("num_shots:", num_shots, "\n")
            total_shots += num_shots
            step_shot[i, 0] = num_shots
            num_shots = 0
            information_array_param[np.where(condition), 5] = i
            # information_array = information_array[np.argsort(
    information_array[:, 3])]

    print(step_shot)
    print("total_shots", total_shots)

    return step_shot, information_array_param


def calc_slope(step_shot_array_param):
    slope_bool = None
        # lets find out if the delay is increasing or decreasing
    if step_shot_array_param[1, 1] - step_shot_array_param[0, 1] > 0.0:
        slope_bool = True
        print("it's positive")
    elif step_shot_array_param[1, 1] - step_shot_array_param[0, 1] < 0.0:
        slope_bool = False

    return slope_bool


def rebin(tt_ttCorr_param, delay_vector_param, zyla_ladm_roi_param,
    ipm_vector_param, delta_t_param, tt_correction_param, simulation,
```

```
    ipm_low, ipm_high):
124
125     # we need to round to the 15th decimal place because the delays are in
        the format -1.499999E-12 seconds.
126     # this causes problems with counting
127     delay_vector_param = np.round(delay_vector_param, decimals=15)
128     # tt_ttCorr_param = np.round(tt_ttCorr_param, decimals=16) # knocks
    off extra digits - the precision isn't as good as the length of the
    float
129     pd.DataFrame(delay_vector_param).to_csv('rounded_delay_vector_param.
    csv')
130
131     # convert to units of fs
132     tt_ttCorr_param *= 1E15
133     delay_vector_param *= 1E15
134     delta_t_param *= 1E15
135
136     # let's create a 2d array with column 1 being the original shot
    location, 2 being the delay
137     # 3 being the tt correction, and 4 being the sum of 2 and 3 (simply
    the corrected delay)
138     # 5 being the ipm_vector, 6th being the step number associated with
    that shot (after sorting)
139     # the shape should be number of total shots by 5 columns
140     information_array = np.zeros((zyla_ladm_roi_param.shape[0], 6), dtype=
    float)
141     first_column = np.arange(0, zyla_ladm_roi_param.shape[0], 1)
142     information_array[:, 0] = first_column # shot index locations, 0 to
    total number of shots
143     information_array[:, 1] = delay_vector_param
```

```
144     if tt_correction_param:
145         information_array[:, 2] = tt_ttCorr_param
146     if not tt_correction_param:
147         information_array[:, 2] = 0.
148         # information_array[:, 3] = delay_vector_param
149     information_array[:, 3] = information_array[:, 1] + information_array
    [:, 2]
150     information_array[:, 4] = ipm_vector_param
151 
152 
153     step_shot_array_param, information_array = calc_step_shots(
    information_array, delta_t_param)
154 
155     # need to redefine the max and min steps after doing step_shot array
    calculation
156     # the reason is that the binsize we define may not divide the step
    range
157     # into an integer number of steps. the function adds bins if that's
    the case
158 
159 
160     pd.DataFrame(information_array).to_csv('information_array_beforesort.
    csv')
161     # now sort information_array based on the adjusted delay column (index
     = 3)
162     # this puts the shots in their correct order
163     information_array = information_array[np.argsort(information_array[:,
    3])]
164 
```

```
165     pd.DataFrame(information_array).to_csv('information_array_aftersort.
    csv')
166 
167     # now throw out the shots that are outside of the range
168     maxstep = np.max(step_shot_array_param[:, 1])
169     minstep = np.min(step_shot_array_param[:, 1])
170     print("minstep:", minstep)
171     print("maxstep:", maxstep)
172     # indexes_to_delete = np.where(np.logical_or(information_array[:, 3] >
     maxstep, information_array[:, 3] < minstep))
173     # i changed the line below to only delete delay values above max step
    + a bin size. I think this is correct
174     # based on how i've been calculating the number of shots in each step
175     indexes_to_delete = np.where(np.logical_or(information_array[:, 3] > (
    maxstep + delta_t_param), information_array[:, 3] < minstep))
176     information_array = np.delete(information_array, indexes_to_delete,
    axis=0)
177 
178     if not simulation:
179         if tt_correction_param:
180             # delete entries with no timetool data
181             indexes_to_delete = np.where(information_array[:, 2] == 0.)
182             information_array = np.delete(information_array,
    indexes_to_delete, axis=0)
183         # delete entires that are greater than the upper intensity limit
184         indexes_to_delete = np.where(information_array[:, 4] > ipm_high)
185         information_array = np.delete(information_array, indexes_to_delete
    , axis=0)
186 
```

```
187     pd.DataFrame(information_array).to_csv('information_array_deleted.csv'
        )
188 
189     print("information_array after deleted shape:", information_array.
        shape)
190 
191     step_shot_array_param, information_array = calc_step_shots(
        information_array, delta_t_param)
192     pd.DataFrame(information_array).to_csv('information_array_shots.csv')
193 
194     zyla_ladm_roi_adjusted = np.zeros((information_array.shape[0],
        zyla_ladm_roi_param.shape[1], zyla_ladm_roi_param.shape[2]), dtype=
        float)
195 
196     for i in range(information_array.shape[0]):
197         zyla_ladm_roi_adjusted[i, :, :] = zyla_ladm_roi_param[int(
        information_array[i, 0]), :, :]
198 
199     print("zyla_ladm_roi_adjusted.shape", zyla_ladm_roi_adjusted.shape)
200 
201     return information_array, step_shot_array_param,
        zyla_ladm_roi_adjusted
202 
203 
204 def build_save_plots(SRTGParam, TGParam, TIMEParam, MaterialParam,
        runNumParam, binSizeParam, saveFigBool, isFilteredBool, withBgBool):
205     binSizeParam = binSizeParam * 10**15
206 
207     plt.rcParams['axes.xmargin'] = 0  # this makes the plot start and end
        at the edges
```

```
208     font = {'family' : 'DejaVu Sans',
209         'weight' : 'normal',
210         'size'   : 20}
211     plt.rc('font', **font)
212 
213     fig, [ax,ax2] = plt.subplots(2,1,figsize=(10,10))
214     fig.subplots_adjust(hspace=0.1, left=0.2)
215     # plt.gca().invert_xaxis()
216     # ax.pcolormesh(SRTGParam,cmap='jet',vmax=.8,vmin=.05)
217     # ax.pcolormesh(SRTGParam,cmap='jet', vmin=1000, vmax=12000)
218     # ax.pcolormesh(SRTGParam,cmap='jet', vmax=1400)
219 
220     # it's needed to match the SRTG and TG plots together
221     ax.pcolormesh(SRTGParam,cmap='jet') # this is the default
222 
223 
224 
225     ax.set_xticks([])
226     ax2.set_xlabel('Pump-Probe Delay (fs)')
227     ax.set_ylabel('Wavelength (nm)')
228 
229 
230     # ax2.plot(TIMEParam[:],(TGParam[:])*10,linewidth=3)
231     # ax2.plot(TIMEParam,(TGParam)*10,linewidth=3)
232     ax2.plot(TIMEParam,(TGParam)*10,linewidth=3, marker='*')  # this is to
        mark points on the TG line to count bins. remove later
233     plt.rcParams['axes.xmargin'] = 0  # this makes the plot start and end
    at the edges
234     # it's needed to match the SRTG and TG plots together
235     ax2.set_xlabel(r'$\tau_{laser}$ (fs)')
```

```
236     ax2.set_ylabel('Signal (arb)')
237     ax2.grid(True)
238     ax.set_title(MaterialParam + ', Run ' + str(runNumParam) + ', Bin Size
        (fs): ' + str(binSizeParam))
239 
240     if saveFigBool:
241         if isFilteredBool:
242             if withBgBool:
243                 plt.savefig('working_plots/Plots'+str(runNumParam)+'
    _Filtered_' +  '_withbg' + '.png',bbox_inches='tight')
244             if not withBgBool:
245                 plt.savefig('working_plots/Plots'+str(runNumParam)+'
    _Filtered_' + str(binSizeParam) + 'fs.png',bbox_inches='tight')
246         if not isFilteredBool:
247             if withBgBool:
248                 plt.savefig('working_plots/Plots'+str(runNumParam)+ '
    _withbg_' + str(binSizeParam) + 'fs' + '.png',bbox_inches='tight')
249             if not withBgBool:
250                 plt.savefig('working_plots/Plots'+str(runNumParam)+'_'+str
    (binSizeParam)+'fs'+'.png',bbox_inches='tight')
251     plt.show()
252     # plt.plot()
253 
254     return None
255 
256 
257 # TODO: this function is not finished. it's a place holder
258 def xray_delay_scan(xray_delay_param, tt_ttCorr_param, rebin_bool):
259     # Populate a vector with number of shots in each step
260     step_shot_array = calc_step_shots(xray_delay_param)
```

```
261     numsteps = len(step_shot_array)
262     print(step_shot_array)
263     print(numsteps)
264     print(np.sum(step_shot_array))  # this is simply to calculate the
        total events
265     # to make sure that it matches the scan template
266
267     # Rebin if the parameter is True
268     return
269
270
271 def laser_delay_scan(runnum_param, laser_delay_param, tt_ttCorr_param,
        zyla_ladm_roi_param, ipm_vector_param,
272 ipm_bkg_limit_param, ipm_low_limit_param, ipm_high_limit_param,
        save_images_bool, delta_t_param, crystalString_param,
        tt_correction_bool, simulation_bool):
273
274
275     information_array, step_shot_array, zyla_ladm_roi_param = rebin(
        tt_ttCorr_param, laser_delay_param,
276         zyla_ladm_roi_param, ipm_vector_param, delta_t_param,
        tt_correction_param=tt_correction_bool, simulation=simulation_bool,
        ipm_low=ipm_low_limit_param, ipm_high=ipm_high_limit_param)
277     tt_ttCorr_param = information_array[:, 2]
278     laser_delay_param = information_array[:, 1]
279     ipm_vector_param = information_array[:, 4]
280
281     positiveDelaySlope = calc_slope(step_shot_array)
282
```

```
283     # the number of steps will have increased if we are refining the bin
    size
284     numsteps = len(step_shot_array)
285     totalevents = np.sum(step_shot_array[:, 0])
286 
287     #I am checking here if any zyla images have nan's in them.
288     # indicies = np.where()
289     # indicies = np.where(np.logical_and(information_array[:, 5] == i,
    information_array[:, 4] < ipm_bkg_limit_param))
290     indicies = np.argwhere(np.isnan(zyla_ladm_roi_param))
291     nan_indicies = indicies[:, 0]
292     print("nan indicies: ", indicies)
293     pd.DataFrame(indicies).to_csv('nanindicies.csv')
294     pd.DataFrame(indicies[:,0]).to_csv('nanindicies_1col.csv')
295     # let's set the nan's data to zero so it doesn't show up in the data
296     # the zyla image and the ipm data need to be set to 0.
297     zyla_ladm_roi_param[nan_indicies, :, :] = 0.
298     information_array[nan_indicies, 4] = 0.
299 
300     # Background subtraction with condition that each shot has
301     # time tool data. No IPM intensity condition
302     # If there is a time tool data entry, then the shot should have
303     # already passed some sort of IPM intensity condition when
304     # the data was created
305 
306     background_array = np.zeros((numsteps, zyla_ladm_roi_param.shape[1],
307     zyla_ladm_roi_param.shape[2]), dtype=float)
308     foreground_array = np.zeros((numsteps, zyla_ladm_roi_param.shape[1],
309     zyla_ladm_roi_param.shape[2]), dtype=float)
310     image_array = np.zeros((numsteps, zyla_ladm_roi_param.shape[1],
```

```
311     zyla_ladm_roi_param.shape[2]), dtype=float)
312     image_withbg_array = np.zeros((numsteps, zyla_ladm_roi_param.shape[1],
313     zyla_ladm_roi_param.shape[2]), dtype=float)
314     ipm_mean_dark = np.zeros(numsteps, dtype=float)
315     ipm_mean_bright = np.zeros(numsteps, dtype=float)
316     DRK = np.zeros(zyla_ladm_roi_param.shape[0], dtype=float)
317     SCT = np.zeros(zyla_ladm_roi_param.shape[0], dtype=float)
318
319
320     # ipm normalization and dark current subtraction
321     for i in range(zyla_ladm_roi_param.shape[0]):
322     #     # zyla_ladm_roi_param[i, :, :] = zyla_ladm_roi_param[i, :, :] /
    (information_array[i, 4]**2)
323         DRK[i] = np.sum(zyla_ladm_roi_param[i, 0:100, 0:100])/10000
324         SCT[i] = np.sum(zyla_ladm_roi_param[i, 100:200, 500:600] - DRK[i])
     /10000 # this is what is in Pam's script - but it doesn't really make
    sense to me
325         # zyla_ladm_roi_param[i, :, :] = zyla_ladm_roi_param[i, :, :] - (
    np.sum(zyla_ladm_roi_param[i, 0:100, 0:100])/10000)
326         zyla_ladm_roi_param[i, :, :] = zyla_ladm_roi_param[i, :, :] - DRK[
    i]
327
328     # ipm normalization
329     # for i in range(step_shot_array.shape[0]):
330     #     indicies = np.where(information_array[:, 5] == i)
331     #     ipm_mean[i] = np.mean(information_array[indicies, 5])
332
333
334     zero_dark=0
335     zero_bright=0
```

```
336     for i in range(step_shot_array.shape[0]):
337         # dark_shots_array = np.zeros((zyla_ladm_roi_param.shape[1],
    zyla_ladm_roi_param.shape[2]), dtype=float)
338         # find all indicies in the current step that also pass the
    intensity condition
339         indicies = np.where(np.logical_and(information_array[:, 5] == i,
    information_array[:, 4] < ipm_bkg_limit_param))
340         indicies = np.asarray(indicies)
341         indicies = indicies[0,:]
342         # print("indicies", indicies)
343         dark_counter = len(indicies)
344         ipm_mean_dark[i] = np.mean(information_array[indicies, 4])
345         ipm_mean_step = np.mean(information_array[indicies, 4])
346         if dark_counter == 0:
347             print("dark_counter is 0. zero background will be subtracted
    off.")
348             zero_dark += 1
349         else:
350             print("dark_counter:", dark_counter)
351 
352             # print("indicies.shape", indicies.shape)
353             dark_shots_array = zyla_ladm_roi_param[indicies, :, :]
354             print("dark_shots_array.shape", dark_shots_array.shape)
355             # background_array[i, :, :] = np.sum(zyla_ladm_roi_param[
    indicies, :, :], axis=0) / dark_counter
356             background_array[i, :, :] = np.sum(dark_shots_array, axis=0) /
     dark_counter
357             # background_array[i, :, :] = np.sum(dark_shots_array, axis=0)
    / (dark_counter * ipm_mean_step * ipm_mean_step)
358             print("background_array.shape", background_array.shape)
```

```
359
360
361     for i in range(step_shot_array.shape[0]):
362         # bright_shots_array = np.zeros((zyla_ladm_roi_param.shape[1],
    zyla_ladm_roi_param.shape[2]), dtype=float)
363         indicies = np.where(np.logical_and(information_array[:, 5] == i,
    information_array[:, 4] > ipm_low_limit_param))
364         # indicies = np.where(np.logical_and(information_array[:, 5] == i,
     information_array[:, 4] > ipm_low_limit_param, information_array[:, 4]
     < 15000))
365         indicies = np.asarray(indicies)
366         indicies = indicies[0,:]
367         # print("indicies", indicies)
368         bright_counter = len(indicies)
369         ipm_mean_bright[i] = np.mean(information_array[indicies, 4])
370         ipm_mean_step = np.mean(information_array[indicies, 4])
371         print("bright_counter", bright_counter)
372         if bright_counter == 0:
373             print("bright_counter is 0. zero foreground will be added.")
374             zero_bright += 1
375         else:
376
377
378             bright_shots_array = zyla_ladm_roi_param[indicies, :, :]
379
380             print("bright_shots_array.shape", bright_shots_array.shape)
381             foreground_array[i, :, :] = np.sum(bright_shots_array, axis=0)
     / bright_counter
382             # foreground_array[i, :, :] = np.sum(bright_shots_array, axis
    =0) / (bright_counter * ipm_mean_step * ipm_mean_step)
```

```
            image_array[i, :, :] = foreground_array[i, :, :] -
    background_array[i, :, :]
            image_withbg_array[i, :, :] = foreground_array[i, :, :]

    print("zero_dark", zero_dark)
    print("zero_bright", zero_bright)


    # Select ROI based off of the averaged image at some index
    # TODO: this will be hard coded until I write the compute_refined_roi
    () function
    # for 210
    # y1 = 75
    # y2 = 125
    # x1 = 200
    # x2 = 400
    # y1 = 82
    # y2 = 105
    # x1 = 265
    # x2 = 405
    display_index = 5

    # for later runs ~280
    # y1 = 100
    # y2 = 150
    # x1 = 100
    # x2 = 300

    y1 = 0
    y2 = -1
```

```
411     x1 = 0
412     x2 = -1
413
414
415     # SRTG calculation
416     NX = np.shape(image_array[display_index, y1:y2, x1:x2])[1]
417     SRTG = np.zeros((numsteps, NX), dtype=float)
418     SRTG_withbg = np.zeros((numsteps, NX), dtype=float)
419
420     for i in range(numsteps):
421         SRTG[i, :] = np.sum(image_array[i, y1:y2, x1:x2], 0)
422         SRTG_withbg[i, :] = np.sum(image_withbg_array[i, y1:y2, x1:x2], 0)
423
424         # indicies = np.where(information_array[:, 5]==i)
425         # ipmnorm = np.mean(information_array[indicies, 4]) * np.mean(
    information_array[indicies, 4])
426         # sctmean = np.mean(SCT[indicies])
427         # SRTG[i, :] = np.sum(image_array[i, y1:y2, x1:x2], 0) / ipmnorm /
     sctmean
428         # SRTG_withbg[i, :] = np.sum(image_withbg_array[i, y1:y2, x1:x2],
    0) / ipmnorm / sctmean
429     SRTG = np.transpose(SRTG)
430     SRTG_withbg = np.transpose(SRTG_withbg)
431
432     # Display the averaged image at some step index
433     show_image(image_withbg_array[display_index], "zyla_ladm_roi without
    background subtraction")
434     show_image(image_array[display_index, :, :], "zyla_ladm_roi with
    background subtraction")
```

```
435     show_image(background_array[display_index, :, :], "background at index
    ")
436     show_image(SRTG, "SRTG w/ BG subtraction")
437     # print(SRTG[:,8]) #TODO: remove later
438     show_image(SRTG_withbg, "SRTG w/o BG subtraction")
439     show_image(image_array[display_index, y1:y2, x1:x2], "roi on zyla")
440
441
442     # TG calculation
443     TG = np.zeros((numsteps), dtype=float)
444     TG1 = np.zeros((numsteps), dtype=float)
445     TG_withbg = np.zeros((numsteps), dtype=float)
446     for i in range(numsteps):
447         TG[i] = np.sum(image_array[i, y1:y2, x1:x2])
448         TG_withbg[i] = np.sum(image_withbg_array[i, y1:y2, x1:x2])
449
450         # indicies = np.where(information_array[:, 5]==i)
451         # ipmnorm = np.mean(information_array[indicies, 4]) * np.mean(
    information_array[indicies, 4])
452         # sctmean = np.mean(SCT[indicies])
453         # TG[i] = np.sum(image_array[i, y1:y2, x1:x2])
454         # TG1[i] = np.sum(image_array[i, y1:y2, x1:x2]) / ipmnorm /
    sctmean
455         # TG_withbg[i] = np.sum(image_withbg_array[i, y1:y2, x1:x2])
456
457     # this fixes the scale of the signal(arb) on the left axis
458     TG = TG/max(TG)
459     # TG = TG/max(TG)*max(TG1) # for run 210
460     TG_withbg = TG_withbg/max(TG_withbg)
461
```

```
462
463     # flip the order of the array
464     SRTG = np.flip(SRTG, axis=1)
465     SRTG_withbg = np.flip(SRTG_withbg, axis=1)
466     TG = np.flip(TG)
467     TG_withbg = np.flip(TG_withbg)
468
469     # time_axis = np.zeros(step_shot_array.shape[0], dtype=float)
470     time_axis = np.zeros(numsteps, dtype=float)
471     for i in range(len(time_axis)):
472         # time_axis[i] = step_shot_array[i, 1]*10**15
473         time_axis[i] = step_shot_array[i, 1]
474
475     print("STRG.shape", SRTG.shape)
476     print("TG.shape", TG.shape)
477     print("time_axis.shape", time_axis.shape)
478
479
480     # Bring both plots together and decide to save
481     if positiveDelaySlope:
482         build_save_plots(SRTG, TG, time_axis, crystalString_param + ',
    ipm_bkg=' + str(ipm_bkg_limit_param) + ' & ipm_low=' +
483                          str(ipm_low_limit_param), runnum_param,
    delta_t_param,saveFigBool=save_images_bool, isFilteredBool=False,
    withBgBool=False)
484         build_save_plots(SRTG_withbg, TG_withbg, time_axis,
    crystalString_param + ' with bg', runnum_param, delta_t_param,
    saveFigBool=save_images_bool, isFilteredBool=False, withBgBool=True)
485
486         # SRTG_savgol = savgol_filter(SRTG, 5, 3, axis=1)
```

```
487         # TG_savgol = savgol_filter(TG, 5, 3)
488         # build_save_plots(SRTG_savgol, TG_savgol, time_axis,
    crystalString_param + (" savgol"), runnum_param, delta_t_param,
    saveFigBool=save_images_bool, isFilteredBool=True, withBgBool=False)
489     if not positiveDelaySlope:
490         build_save_plots(SRTG, -TG, time_axis, crystalString_param,
    runnum_param, delta_t_param, saveFigBool=save_images_bool,
    isFilteredBool=False, withBgBool=False)
491         build_save_plots(SRTG_withbg, -TG_withbg, time_axis,
    crystalString_param + ' with bg', runnum_param, delta_t_param,
    saveFigBool=False, isFilteredBool=False, withBgBool=True)
492
493     # save data to h5. uncomment if you don't want to do this
494     # name = str(runnum_param) + "_TG_SRTG.h5"
495     name = str(runnum_param) + "_TG_SRTG_nosorted.h5"
496     file = h5py.File(name, 'w')
497     file.create_dataset("SRTG", data=SRTG)
498     file.create_dataset("SRTG_withbg", data=SRTG_withbg)
499     file.create_dataset("TG", data=TG)
500     file.create_dataset("TG_withbg", data=TG_withbg)
501     file.create_dataset("time_axis", data=time_axis)
502     file.close()
503
504
505     show_plot_two_arg(np.arange(numsteps), np.fft.fft(TG), "TG FFT")
506     show_plot_two_arg(np.arange(numsteps), np.fft.fft(TG_savgol), "TG FFT"
    )
507     show_plot_two_arg(np.arange(-numsteps/2,numsteps/2, 1), np.fft.
    fftshift(np.fft.fft(TG)), "TG FFT Shift")
```

```
508     show_plot_two_arg(np.arange(-numsteps/2,numsteps/2, 1), np.fft.
        fftshift(np.fft.fft(TG_savgol)), "TG FFT Shift")
509     return None
510
511
512 def main():
513     # Define run number and file location
514     runNum = 220
515     # runNum = 221
516     # runNum = 209
517     # runNum = 210
518     # runNum = 183
519     # runNum = 238
520     # runNum = 242
521     # runNum = 232
522     # runNum = 221
523     # runNum = 198
524     # runNum = 179
525     # runNum = 240
526     # runNum = 242
527     # runNum = 280
528     # runNum = 267
529
530     # binsize = 125E-15
531     binsize = 100E-15  # 100 fs (Default bin size without time sorting)
532     # binsize = 90E-15
533     # binsize = 75E-15 #75 fs
534     # binsize = 50E-15  # 50 fs
535     # binsize = 30E-15
536     # binsize = 25E-15
```

```
537     # binsize = 15E-15
538
539     # Define bool parameters
540
541     simulation = False
542     save_images = False
543     tt_correction = False
544     ipm_low = 5000
545     ipm_high = 300000
546     # ipm_high = 14000
547     ipm_background = 1000
548
549
550
551     expName = "xcslv3118"
552     filename = '/cds/data/psdm/xcs/xcslv3118/hdf5/smalldata/
    smalldata_backup/xcslv3118_Run%03d.h5' % runNum
553     # simlated data files follow
554     # filename = '/cds/data/psdm/xcs/xcslv3118/hdf5/smalldata/
    smalldata_backup/simulated_data.h5'
555     # filename = '/cds/data/psdm/xcs/xcslv3118/hdf5/smalldata/
    smalldata_backup/simulated_data_source.h5'
556
557     # Open h5 file and print data sets
558     print('%s %s' % (expName, runNum))
559     f = h5py.File(filename, 'r')
560     ds = f['/']
561     print(ds.keys())
562
563     # Alias PV's to python variables
```

```
564     zyla_ladm_roi = np.asarray(f['zyla_ladm/ROI0_area'], dtype=float)
565     ipm4_sum = np.asarray(f['/ipm4/sum'], dtype=float)
566     laser_delay = np.asarray(f['/epics/lxt_ttc'], dtype=float)  # i
    believe this is laser delay (s), not in (ps)
567     TOPT = laser_delay
568     xray_delay = np.asarray(f['/snd_epics/snd_delay'], dtype=float)  # i
    believe this is what Pam calls "xray delay (ps)" in BGORun210.ipynb
569     # tt_ttCorr = np.asarray(f['/tt/ttCorr'], dtype=float)
570     # tt_ttCorr = tt_ttCorr * 10**(-12)
571     # pd.DataFrame(tt_ttCorr).to_csv('tt_ttCorr_orig.csv')
572     # convert pixels to fs with calibration, but take only non zero
    entries
573
574
575
576
577     # Characterize the run type
578     # Possible run types: snd_delay(xray delay), lxt(time tool calibration
    , I believe),
579     # lxt_ttc(laser delay), snd_t4_chi1
580     runType, crystalString = characterize_run(runNum)
581     if simulation:
582         tt_ttCorr = np.asarray(f['/tt/ttCorr'], dtype=float)
583         crystalString = "Simulation"
584     if not simulation:
585         tt_ttCorr = np.asarray(f['/tt/FLTPOS'])
586
587         # NOTICE: this is for using Ryan's timetool locations. comment out
     if not using ryan's locations
588 #         tt_ttCorr[:] = 0.
```

```
589 #         tt_loc_del = np.asarray(pd.read_csv('tt_locations_210_deleted.
    csv'), dtype=float)
590 #         for i in range(tt_loc_del.shape[0]):
591 #             index = int(tt_loc_del[i, 2])
592 #             # print("index", index, "value", tt_loc_del[i, 3])
593 #             tt_ttCorr[index] = tt_loc_del[i, 3]
594
595
596         indicies = np.argwhere(tt_ttCorr)
597         # tt_ttCorr[indicies] = ((tt_ttCorr[indicies] - 200) * (-1.962146)
    ) * 10**(-15)
598         # tt_ttCorr[indicies] = ((tt_ttCorr[indicies] - 600) * (-1.962146)
    ) * 10**(-15)
599         tt_ttCorr[indicies] = ((tt_ttCorr[indicies] - 500) * (-1.962146))
    * 10**(-15)
600
601         # tt_ttCorr[indicies] = ((tt_ttCorr[indicies] - 500) * (-1.962146)
    )
602         # indicies2 = np.where(np.abs(tt_ttCorr) >= 100)
603         # indicies2 = np.asarray(indicies2)
604         # indicies2 = indicies2[0, :]
605         # print("indicies2", len(indicies2))
606         # print("ratio (nozero):", len(indicies2)/len(indicies))
607         # print("ratio (zero):", len(indicies2)/len(tt_ttCorr))
608
609         # tt_ttCorr[indicies] = ((tt_ttCorr[indicies] - 800) * (-1.962146)
    ) * 10**(-15)
610
611         pd.DataFrame(tt_ttCorr).to_csv('test.csv')
612
```

```
613     if runType == "lxt_ttc":
614         laser_delay_scan(runNum, laser_delay, tt_ttCorr, zyla_ladm_roi,
    ipm4_sum,
615         ipm_bkg_limit_param=ipm_background, ipm_low_limit_param=ipm_low,
    ipm_high_limit_param=ipm_high, save_images_bool=save_images,
    delta_t_param=binsize, crystalString_param=crystalString,
    tt_correction_bool=tt_correction, simulation_bool=simulation)
616     elif runType == "snd_delay":
617         # xray_delay_scan(xray_delay, tt_ttCorr, rebin_bool=False)
618         print("run is a split and delay xray scan. this function hasn't
    been written yet.")
619     else:
620         print("run type is not lxt_ttc or snd_delay.")
621 
622     f.close()
623     return
624 
625 
626 if __name__ == '__main__':
627     main()
```

Appendix B

Plot Creation Code

I have written the following code in Python to format and build the plots contained in this thesis.

```
import numpy as np
from matplotlib import pyplot as plt
import h5py
import pandas as pd
from scipy.signal import savgol_filter

def show_image(array, title):
    plt.figure(num=None, figsize=(15, 15))
    plt.title(title)
    # plt.pcolormesh(array, cmap='jet')
    plt.imshow(array, aspect='auto', cmap='jet')
    plt.colorbar(shrink=0.25)
    plt.show()
    return None


def show_plot_two_arg(ds1, ds2, title):
```

```
18      plt.figure(num=None, figsize=(15, 15))
19      plt.plot(ds1, ds2, 'c-', marker="None")
20      # plt.plot(ds1, ds2, '--', marker="None", color='.5')
21      plt.title(title)
22      plt.show()
23      return None
24
25  def show_plot(ds1, title):
26      plt.figure(num=None, figsize=(15, 15))
27      plt.plot(ds1, 'c-', marker="None")
28      # plt.plot(ds1, ds2, '--', marker="None", color='.5')
29      plt.title(title)
30      plt.show()
31      return None
32
33
34
35
36  runNum = 210
37  filename = str(runNum) + '_TG_SRTG_nosorted.h5'
38  f = h5py.File(filename, 'r')
39  ds = f['/']
40  SRTG_210_nosorted = np.asarray(f['/SRTG'], dtype=float)
41  TG_210_nosorted = np.asarray(f['/TG'], dtype=float)
42  time_axis_210_nosorted = np.asarray(f['/time_axis'], dtype=float)
43  f.close()
44
45  # filename = str(runNum) + '_TG_SRTG_sorted.h5'
46  # f = h5py.File(filename, 'r')
47  # ds = f['/']
```

```
# SRTG_210_sorted = np.asarray(f['/SRTG'], dtype=float)
# TG_210_sorted = np.asarray(f['/TG'], dtype=float)
# time_axis_210_sorted = np.asarray(f['/time_axis'], dtype=float)
# f.close()

# filename = str(runNum) + '_TG_SRTG_nosorted_norm.h5'
# f = h5py.File(filename, 'r')
# ds = f['/']
# SRTG_210_nosorted_norm = np.asarray(f['/SRTG'], dtype=float)
# TG_210_nosorted_norm = np.asarray(f['/TG'], dtype=float)
# time_axis_210_nosorted_norm = np.asarray(f['/time_axis'], dtype=float)
# f.close()

# filename = str(runNum) + '_TG_SRTG_nosorted_cap.h5'
# f = h5py.File(filename, 'r')
# ds = f['/']
# SRTG_210_nosorted_cap = np.asarray(f['/SRTG'], dtype=float)
# TG_210_nosorted_cap = np.asarray(f['/TG'], dtype=float)
# time_axis_210_nosorted_cap = np.asarray(f['/time_axis'], dtype=float)
# f.close()




runNum = 220
# filename = str(runNum) + '_TG_SRTG_nosorted.h5'
# f = h5py.File(filename, 'r')
# ds = f['/']
# SRTG_220_nosorted = np.asarray(f['/SRTG'], dtype=float)
# TG_220_nosorted = np.asarray(f['/TG'], dtype=float)
```

```
78 # time_axis_220_nosorted = np.asarray(f['/time_axis'], dtype=float)
79 # f.close()
80
81 filename = str(runNum) + '_TG_SRTG_sorted.h5'
82 # filename = str(runNum) + '_TG_SRTG_nosorted.h5'
83 f = h5py.File(filename, 'r')
84 ds = f['/']
85 SRTG_220_sorted = np.asarray(f['/SRTG'], dtype=float)
86 TG_220_sorted = np.asarray(f['/TG'], dtype=float)
87 time_axis_220_sorted = np.asarray(f['/time_axis'], dtype=float)
88 f.close()
89
90 filename = str(runNum) + '_TG_SRTG_nosorted.h5'
91 # filename = str(runNum) + '_TG_SRTG_nosorted.h5'
92 f = h5py.File(filename, 'r')
93 ds = f['/']
94 SRTG_220_nosorted = np.asarray(f['/SRTG'], dtype=float)
95 TG_220_nosorted = np.asarray(f['/TG'], dtype=float)
96 time_axis_220_nosorted = np.asarray(f['/time_axis'], dtype=float)
97 f.close()
98
99 # filename = str(runNum) + '_TG_SRTG_nosorted_norm.h5'
100 # f = h5py.File(filename, 'r')
101 # ds = f['/']
102 # SRTG_220_nosorted_norm = np.asarray(f['/SRTG'], dtype=float)
103 # TG_220_nosorted_norm = np.asarray(f['/TG'], dtype=float)
104 # time_axis_220_nosorted_norm = np.asarray(f['/time_axis'], dtype=float)
105 # f.close()
106
107 # filename = str(runNum) + '_TG_SRTG_nosorted_cap.h5'
```

```
# f = h5py.File(filename, 'r')
# ds = f['/']
# SRTG_220_nosorted_cap = np.asarray(f['/SRTG'], dtype=float)
# TG_220_nosorted_cap = np.asarray(f['/TG'], dtype=float)
# time_axis_220_nosorted_cap = np.asarray(f['/time_axis'], dtype=float)
# f.close()

# filename = str(runNum) + '_TG_SRTG_sorted_norm.h5'
# f = h5py.File(filename, 'r')
# ds = f['/']
# SRTG_220_sorted_norm = np.asarray(f['/SRTG'], dtype=float)
# TG_220_sorted_norm = np.asarray(f['/TG'], dtype=float)
# time_axis_220_sorted_norm = np.asarray(f['/time_axis'], dtype=float)
# f.close()



runNum = 280
filename = str(runNum) + '_TG_SRTG_nosorted.h5'
f = h5py.File(filename, 'r')
ds = f['/']
SRTG_280_nosorted = np.asarray(f['/SRTG'], dtype=float)
TG_280_nosorted = np.asarray(f['/TG'], dtype=float)
time_axis_280_nosorted = np.asarray(f['/time_axis'], dtype=float)
f.close()

# filename = str(runNum) + '_TG_SRTG_sorted.h5'
# f = h5py.File(filename, 'r')
# ds = f['/']
# SRTG_280_sorted = np.asarray(f['/SRTG'], dtype=float)
```

```
138 # TG_280_sorted = np.asarray(f['/TG'], dtype=float)
139 # time_axis_280_sorted = np.asarray(f['/time_axis'], dtype=float)
140 # f.close()
141
142 # filename = str(runNum) + '_TG_SRTG_nosorted_norm.h5'
143 # f = h5py.File(filename, 'r')
144 # ds = f['/']
145 # SRTG_280_nosorted_norm = np.asarray(f['/SRTG'], dtype=float)
146 # TG_280_nosorted_norm = np.asarray(f['/TG'], dtype=float)
147 # time_axis_280_nosorted_norm = np.asarray(f['/time_axis'], dtype=float)
148 # f.close()
149
150 # filename = str(runNum) + '_TG_SRTG_nosorted_cap.h5'
151 # f = h5py.File(filename, 'r')
152 # ds = f['/']
153 # SRTG_280_nosorted_cap = np.asarray(f['/SRTG'], dtype=float)
154 # TG_280_nosorted_cap = np.asarray(f['/TG'], dtype=float)
155 # time_axis_280_nosorted_cap = np.asarray(f['/time_axis'], dtype=float)
156 # f.close()
157
158
159 # FIXME: REMOVE
160 # runNum = 267
161 # filename = str(runNum) + '_TG_SRTG_nosorted.h5'
162 # f = h5py.File(filename, 'r')
163 # ds = f['/']
164 # SRTG_210_nosorted = np.asarray(f['/SRTG'], dtype=float)
165 # TG_210_nosorted = np.asarray(f['/TG'], dtype=float)
166 # time_axis_210_nosorted = np.asarray(f['/time_axis'], dtype=float)
167 # f.close()
```

```
168
169
170 # adjust these to insert different data into the plots
171 bgo_srtg = SRTG_210_nosorted
172 bgo_tg = TG_210_nosorted
173 bgo_axis = time_axis_210_nosorted
174 bgo_tg = bgo_tg/np.max(bgo_tg)
175
176 yag_srtg = SRTG_220_sorted
177 yag_srtg2 = SRTG_220_nosorted
178 yag_tg = TG_220_sorted
179 yag_tg2 = TG_220_nosorted
180 yag_axis = time_axis_220_sorted
181 yag_axis2 = time_axis_220_nosorted
182
183 zno_srtg = SRTG_280_nosorted
184 zno_tg = TG_280_nosorted
185 zno_axis = time_axis_280_nosorted
186
187 bgo_center_index = 0
188 for i in range(bgo_srtg.shape[1]):
189     maxval = np.max(bgo_srtg[:, i])
190     index = np.where(bgo_srtg[:, i] == maxval)
191     index = index[0]
192     index = index[0]
193     bgo_center_index = bgo_center_index + index
194 bgo_center_index = int(np.round(bgo_center_index / bgo_srtg.shape[1]))
195 print(bgo_center_index)
196
197 yag_center_index = 0
```

```
198 for i in range(yag_srtg.shape[1]):
199     maxval = np.max(yag_srtg[:, i])
200     index = np.where(yag_srtg[:, i] == maxval)
201     index = index[0]
202     index = index[0]
203     yag_center_index = yag_center_index + index
204 yag_center_index = int(np.round(yag_center_index / yag_srtg.shape[1]))
205 print(yag_center_index)
206 
207 zno_center_index = 0
208 for i in range(zno_srtg.shape[1]):
209     maxval = np.max(zno_srtg[:, i])
210     index = np.where(zno_srtg[:, i] == maxval)
211     index = index[0]
212     index = index[0]
213     zno_center_index = zno_center_index + index
214 zno_center_index = int(np.round(zno_center_index / zno_srtg.shape[1]))
215 print(zno_center_index)
216 
217 yag_center_index = bgo_center_index + 8
218 zno_center_index = zno_center_index + 4
219 bgo_srtg = bgo_srtg[bgo_center_index-160:bgo_center_index+160, :]
220 yag_srtg = yag_srtg[yag_center_index-160:yag_center_index+160, :]
221 yag_srtg2 = yag_srtg2[yag_center_index-160:yag_center_index+160, :]
222 zno_srtg = zno_srtg[zno_center_index-160:zno_center_index+160, :]
223 
224 centernm = 400.26
225 nmperpixel = -0.0668
226 nmrange = 320 * (-nmperpixel)
```

```
srtg_yaxis = np.linspace(centernm-(nmrange/2), centernm+(nmrange/2), num
    =320)
print(srtg_yaxis)
print(srtg_yaxis.shape)

print(bgo_srtg.shape)
print(yag_srtg.shape)
print(zno_srtg.shape)

step = 100
bgo_center = 3
bgo_axis = np.arange(0-(bgo_center*step), (bgo_axis.shape[0]-(bgo_center))
    *step, step)
print(bgo_axis)
yag_center = 4
yag_axis = np.arange(0-(yag_center*step), (yag_axis.shape[0]-(yag_center))
    *step, step)
yag_axis2 = np.arange(0-(yag_center*step), (yag_axis2.shape[0]-(yag_center
    ))*step, step)
print(yag_axis)
zno_center = 5
zno_axis = np.arange(0-(zno_center*step), (zno_axis.shape[0]-(zno_center))
    *step, step)
print(zno_axis)



plt.rcParams['axes.xmargin'] = 0  # this makes the plot start and end at
    the edges
font = {'family' : 'DejaVu Sans',
```

```
251     'weight' : 'normal',
252     'size'   : 20}
253 # font = {'family' : 'DejaVu Sans',
254 #     'weight' : 'normal'}
255 plt.rc('font', **font)
256 # plt.rc('text', usetex=False)
257 fig, axes = plt.subplots(2, 3, figsize=(10,7), sharey='row')
258 # plt.rcParams['text.usetex'] = True
259 # plt.rcParams['text.latex.preamble'] = [r"\usepackage{lmodern}"]
260 # plt.rcParams['text.usetex'] = True
261 # plt.rc('text', usetex=True)
262 # fig.suptitle("Title")
263
264
265 # axes[0, 0].pcolormesh(bgo_srtg, cmap='jet')
266 # axes[0, 1].pcolormesh(yag_srtg, cmap='jet')
267 # axes[0, 2].pcolormesh(zno_srtg, cmap='jet')
268 axes[0, 0].pcolormesh(time_axis_210_nosorted, srtg_yaxis, bgo_srtg, cmap='
    jet', vmax=1000)
269 axes[0, 1].pcolormesh(time_axis_220_sorted, srtg_yaxis, yag_srtg, cmap='
    jet')
270 axes[0, 2].pcolormesh(time_axis_280_nosorted, srtg_yaxis, zno_srtg, cmap='
    jet')
271
272
273 axes[1, 0].plot(bgo_axis, bgo_tg)
274 axes[1, 1].plot(yag_axis, yag_tg)
275 axes[1, 2].plot(zno_axis, zno_tg)
276
277 # axes[0, 0].set_yticks(srtg_yaxis)
```

```
278 # axes[0, 1].set_yticks(srtg_yaxis)
279 # axes[0, 2].set_yticks(srtg_yaxis)
280
281 axes[0, 0].xaxis.set_visible(False)
282 axes[0, 1].xaxis.set_visible(False)
283 axes[0, 2].xaxis.set_visible(False)
284 # axes[0, 1].yaxis.set_visible(False)
285 # axes[0, 2].yaxis.set_visible(False)
286 # axes[1, 1].yaxis.set_visible(False)
287 # axes[1, 2].yaxis.set_visible(False)
288
289 axes[1, 0].minorticks_on()
290 axes[1, 1].minorticks_on()
291 axes[1, 2].minorticks_on()
292
293 # axes[1, 0].grid(True, which='minor')
294 axes[1, 0].grid(True)
295 axes[1, 1].grid(True)
296 axes[1, 2].grid(True)
297
298 # FIXME: Test - remove when done
299 axes[1, 0].minorticks_on()
300 axes[1, 1].minorticks_on()
301 axes[1, 2].minorticks_on()
302 axes[1, 0].grid(which='minor', color='#DDDDDD', linestyle=':', linewidth
    =0.5)
303 axes[1, 1].grid(which='minor', color='#DDDDDD', linestyle=':', linewidth
    =0.5)
304 axes[1, 2].grid(which='minor', color='#DDDDDD', linestyle=':', linewidth
    =0.5)
```

```
305
306 plt.rcParams['axes.xmargin'] = 0  # this makes the plot start and end at
    the edges
307 font = {'family' : 'DejaVu Sans',
308     'weight' : 'normal',
309     'size'   : 20}
310 # font = {'family' : 'DejaVu Sans',
311 #     'weight' : 'normal'}
312 plt.rc('font', **font)
313
314
315 axes[0, 0].set_title("BGO")
316 axes[0, 1].set_title("YAG")
317 axes[0, 2].set_title("ZnO")
318
319 axes[1, 0].set_xlabel(r'$\tau_{probe}$ (fs)')
320 axes[1, 1].set_xlabel(r'$\tau_{probe}$ (fs)')
321 axes[1, 2].set_xlabel(r'$\tau_{probe}$ (fs)')
322 # axes[1, 2].set_xlabel(r'$\mathit{\tau_{pump}}$ (fs)')
323
324 # axes[0, 0].set_ylabel(r'Wavelength, $\mathit{\lambda}$ (nm)')
325 axes[0, 0].set_ylabel(r'Wavelength, $\lambda$ (nm)')
326 axes[1, 0].set_ylabel('Intensity (arb)')
327
328 # axes[0, 0].text(0.0, 1.0, 'a)', verticalalignment='top')
329 # axes[0, 0].annotate('a)', xy=(-0.15, 1.06), xycoords='axes fraction')
330 # axes[1, 0].annotate('b)', xy=(-0.15, 1.05), xycoords='axes fraction')
331 # axes[0, 1].annotate('c)', xy=(-0.15, 1.06), xycoords='axes fraction')
332 # axes[1, 1].annotate('d)', xy=(-0.15, 1.05), xycoords='axes fraction')
333 # axes[0, 2].annotate('e)', xy=(-0.15, 1.06), xycoords='axes fraction')
```

```
# axes[1, 2].annotate('f)', xy=(-0.15, 1.05), xycoords='axes fraction')


# plt.savefig('plots_opticsexpress_laser.png', facecolor='white')
plt.savefig('plots_opticsexpress_laser_moregrid.png', facecolor='white')



# This cell is to build a figure with two samples/runs

plt.rcParams['axes.xmargin'] = 0  # this makes the plot start and end at
    the edges
font = {'family' : 'DejaVu Sans',
    'weight' : 'normal',
    'size'   : 20}
# font = {'family' : 'DejaVu Sans',
#     'weight' : 'normal'}
plt.rc('font', **font)
# plt.rc('text', usetex=False)
fig, axes = plt.subplots(2, 2, figsize=(7,7), sharey='row')
# plt.rcParams['text.usetex'] = True
# plt.rcParams['text.latex.preamble'] = [r"\usepackage{lmodern}"]
# plt.rcParams['text.usetex'] = True
# plt.rc('text', usetex=True)
# fig.suptitle("Title")


# axes[0, 0].pcolormesh(bgo_srtg, cmap='jet')
# axes[0, 1].pcolormesh(yag_srtg, cmap='jet')
# axes[0, 2].pcolormesh(zno_srtg, cmap='jet')
```

```
363
364 axes[0, 1].pcolormesh(time_axis_220_sorted, srtg_yaxis, yag_srtg, cmap='
        jet')
365 axes[0, 0].pcolormesh(time_axis_220_nosorted, srtg_yaxis, yag_srtg2, cmap=
        'jet')
366
367
368 axes[1, 1].plot(yag_axis, yag_tg)
369 axes[1, 0].plot(yag_axis2, yag_tg2)
370
371 axes[0, 0].xaxis.set_visible(False)
372 axes[0, 1].xaxis.set_visible(False)
373
374
375 axes[1, 0].minorticks_on()
376 axes[1, 1].minorticks_on()
377
378 # axes[1, 0].grid(True, which='minor')
379 axes[1, 0].grid(True)
380 axes[1, 1].grid(True)
381
382 # FIXME: Test - remove when done
383 axes[1, 0].minorticks_on()
384 axes[1, 1].minorticks_on()
385 axes[1, 0].grid(which='minor', color='#DDDDDD', linestyle=':', linewidth
        =0.5)
386 axes[1, 1].grid(which='minor', color='#DDDDDD', linestyle=':', linewidth
        =0.5)
387
```

```
388 plt.rcParams['axes.xmargin'] = 0  # this makes the plot start and end at
        the edges
389 font = {'family' : 'DejaVu Sans',
390     'weight' : 'normal',
391     'size'   : 20}
392 # font = {'family' : 'DejaVu Sans',
393 #     'weight' : 'normal'}
394 plt.rc('font', **font)
395
396
397 axes[0, 0].set_title("YAG (X-ray Jitter)")
398 axes[0, 1].set_title("YAG (Corrected)")
399
400 axes[1, 0].set_xlabel(r'$\tau_{probe}$ (fs)')
401 axes[1, 1].set_xlabel(r'$\tau_{probe}$ (fs)')
402 # axes[1, 2].set_xlabel(r'$\mathit{\tau_{pump}}$ (fs)')
403
404 # axes[0, 0].set_ylabel(r'Wavelength, $\mathit{\lambda}$ (nm)')
405 axes[0, 0].set_ylabel(r'Wavelength, $\lambda$ (nm)')
406 axes[1, 0].set_ylabel('Intensity (arb)')
407
408 # axes[0, 0].text(0.0, 1.0, 'a)', verticalalignment='top')
409
410
411 # plt.savefig('plots_opticsexpress_laser.png', facecolor='white')
412 plt.savefig('plots_bothyag.png', facecolor='white', bbox_inches="tight")
```

Bibliography

[1] I. L. Markov, "Limits on fundamental limits to computation,", 2014.

[2] Rayleigh, "XXXI. Investigations in optics, with special reference to the spectroscope,", 1879.

[3] S. Boutet and M. Yabashi, in *X-Ray Free Electron Lasers and Their Applications: A Revolution in Structural Biology* (2018), pp. 1–21.

[4] J. R. Rouxel *et al.*, "(Supplemental Info) Hard X-ray transient grating spectroscopy on bismuth germanate," Nature Photonics **15,** 499–503 (2021).

[5] W. Peters *et al.*, "Hard x-ray, optical four-wave mixing using a split-and-delay line," Manuscript submitted for publication (2023).

[6] H. Shi and D. Zhu, "Multi-Axis Nanopositioning System for the Hard X-ray Split-Delay System at the LCLS," Synchrotron Radiation News **31,** 15–20 (2018).

[7] Y. Feng *et al.*, "A single-shot intensity-position monitor for hard x-ray FEL sources," In , **8140,** 81400Q (SPIE, 2011).

[8] N. Hartmann *et al.*, "Attosecond time-energy structure of X-ray free-electron laser pulses," Nature Photonics **12,** 215–220 (2018).

[9] M. Harmand *et al.*, "Achieving few-femtosecond time-sorting at hard X-ray free-electron lasers," Nature Photonics **7,** 215–218 (2013).

[10] R. Ruppin and R. Englman, "Optical phonons of small crystals," Reports on Progress in Physics **33,** 149 (1970).

[11] M. Couzi, J. R. Vingnalou, and G. Boulon, "Infrared and Raman Study of the Optical Phonons in Bi4Ge3O12 Single Crystal," (1976).

[12] X. Fu, Y. Guo, and J. Zhou, "Terahertz optical parameters and lattice vibration-induced resonance of Er3+-doped y 3Al 5O 12 crystal," Journal of Electromagnetic Waves and Applications **27,** 1792–1799 (2013).

[13] N. Ashkenov *et al.*, "Infrared dielectric functions and phonon modes of high-quality ZnO films," Journal of Applied Physics **93,** 126–133 (2003).

[14] Y. Gong, Y. Zhao, Z. Zhou, D. Li, H. Mao, Q. Bao, Y. Zhang, and G. P. Wang, "Polarized Raman Scattering of In-Plane Anisotropic Phonon Modes in α-MoO3," Advanced Optical Materials **10,** 2200038 (2022).

[15] A. A. Lutman *et al.*, "Polarization control in an X-ray free-electron laser," Nature Photonics **10,** 468–472 (2016).

[16] J. N. Sweetser, D. N. Fittinghoff, and R. Trebino, "Transient-grating frequency-resolved optical gating," **22,** 519–521 (1997).

Index