Methods for Improving Tabletop Ptychography

Hyrum Taylor

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

Methods for Improving Tabletop Ptychography

Hyrum Taylor
Department of Physics and Astronomy, BYU
Bachelor of Science

Ptychography is a powerful imaging method that allows for wavelength-limited resolution. Beamlines are in high demand for ptychography measurements, so tabletop HHG photon sources are a useful way to increase the amount of samples measured, but it comes with additional challenges. In this thesis, I show a method to improve ptychography by creating a two-camera two-mirror controller in order to form a closed-loop Proportional-Integral beam stabilizer, and show that the resulting reconstruction process is improved. I also show images produced from the ptychography reconstruction process, and talk about the data reconstruction process.

Keywords: Ptychography, Tabletop, High Harmonic Generation, Beam Stabilization, Phase Retrieval

ACKNOWLEDGMENTS

I would like to thank Richard Sandberg, my research advisor, for the effort and time he has put in to help me grow as a researcher. I would also like to thank Taylor Buckway, a graduate student whose advice and willingness to work with me on the HHG system was invaluable, as well as his code. In addition, I would like to thank Nick Porter for writing the Ptychodactyl code, as well as the many other people in the research group who helped me understand concepts. Of course, none of this would have been possible if it weren't for my parents, siblings, friends, and other family members who have supported me these past few years. We thank the BYU College of Computational, Mathematical, and Physical Sciences and Department of Physics and Astronomy for funding this work.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A prerequisite for understanding the world is the ability to observe it. The higher the precision with which we can see the world, the more that science is able to do. For this reason, it is vital for a variety of scientific fields that we develop high-resolution imaging techniques.

One common method of high-resolution imaging in the short-wavelength regime is diffraction imaging. Diffraction imaging is (in theory) wavelength limited, allowing us to resolve images limited only by the wavelength of the probe [1, 2]. This is because it is a form of lensless imaging, meaning that lens quality is not a limiting factor. Ptychography is one of the most common methods of diffraction imaging [3], and is the method I have used. This powerful tool is often used in beamlines, which produce very high frequency light in the EUV or x-ray range. See Figure 1.2.

However, beamline construction cannot keep up with demand. As these powerful tools become widely known and appreciated, the demand for high-resolution imaging increases. Thus, finding a method of performing tabletop ptychography (i.e. without needing to go to a beamline) would increase the accessibility of measurements. One way of doing this is through High Harmonic Generation (HHG), which is a method of turning a powerful large-wavelength coherent source into a short-wavelength coherent source.

1.1 Phase Retrieval

The process of diffraction imaging is more complex than traditional imaging. To take a diffraction imaging measurement, the researcher places a sample in the path of a coherent laser beam, and measures the diffracted light far downstream. Thus, the image measured on a camera (CCD) is not the shape of the sample. Instead, it is the intensity of the Fourier transform of the sample. This has the complication that the phase information is lost, making it impossible to correctly take the Inverse (Fast) Fourier Transform (IFFT). By phase information, we are referring to the phase delay that the beam undergoes after it has propagated through the sample.

The most common solution to the phase problem is iterative phase retrieval [4]. In essence, we guess the diffraction pattern's phase, then computationally take the IFFT. The result is an extremely inaccurate guess for the object's shape in real space. We then add constraints in the real space such as being limited to a certain area (for small samples), or requiring it to match with its neighboring images (for ptychography). After applying these constraints, we take the computational Fast Fourier Transform (FFT). The resulting image is not the same as the measured image, so we keep the phase of this new image but replace the intensity with the measured values. If we do this process repeatedly, then we will eventually approach the true solution (i.e. the image we are trying to see). For a visual explanation, see Figure 1.1.

We have used Tike; a program available on Github developed at Argonne National Lab [5]; to do this process. According to Aaron Redd's thesis [6], Tike is a good option for this reconstruction process when compared to other popular ptychography programs. It works well and can be run without overly specialized hardware, such as at the Fulton Supercomputing Lab at BYU.

In order to understand this work, we need to understand what ptychography is. Ptychography is a way of taking high-resolution images of samples over a larger area than would normally be possible. Why is ptychography able to get such a high resolution? Because it is a lensless imaging method that is also wavelength limited. Wavelength limited means that the smaller the wavelength

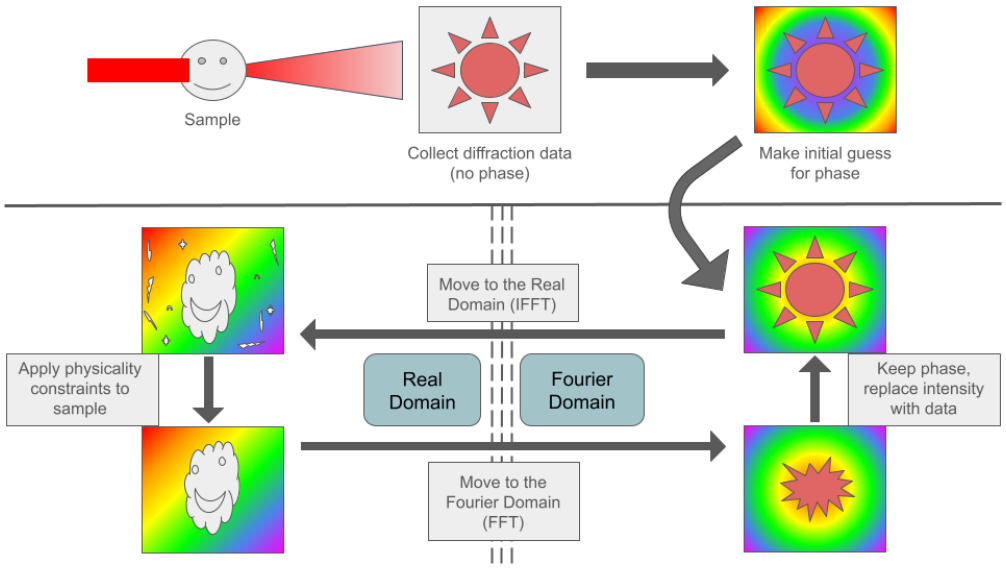

Figure 1.1 Phase Retrieval Cycle: This figure shows how a diffraction image is taken, and the retrieval cycle is run some number of times, until the real space image is a good representation of the sample. This real space image is the image that we are trying to measure, and it approaches the true shape of the sample as more iterations are performed.

of the particle (photon) that is used to probe the sample, the smaller the features of the sample that we can see. Lensless means that we do not need lenses for this method, which is important because refraction-based lenses do not work well in the EUV and x-ray regime [7].

So, how does ptychography work? As we can see in Figure 1.2, we scan the beam across a sample and measure the diffraction pattern from each location. From there, we use the same process as for single-image diffraction imaging. For the real-space constraint, we use the probe overlap to say that the same features must show up in the same place in different beam images: to summarize, we stitch the images together. This is a simple result of the fact that the sample does not change from image to image.

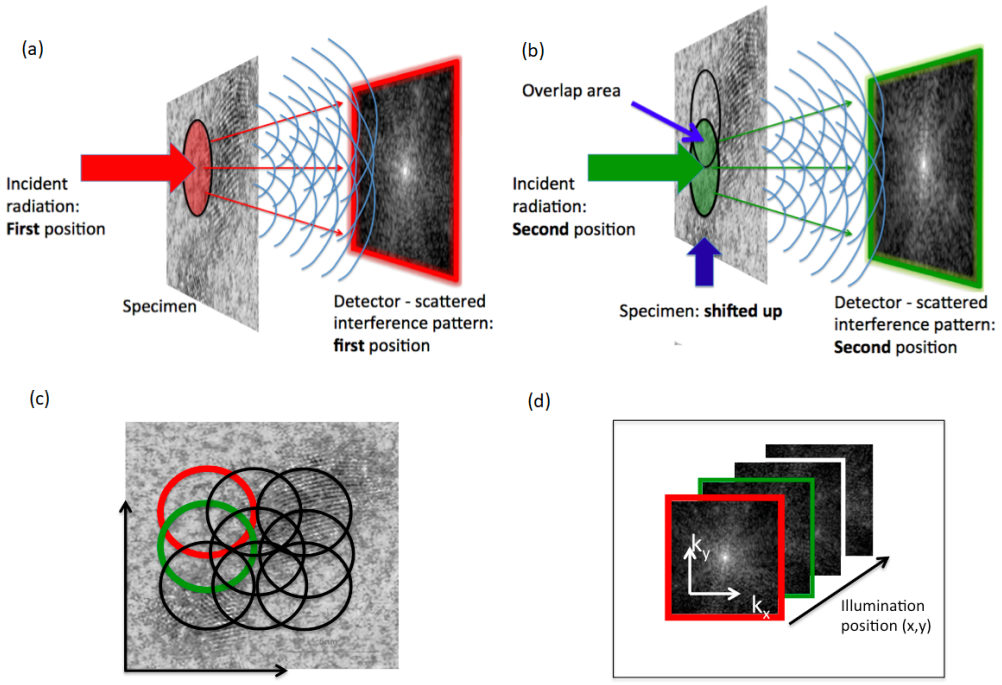

Figure 1.2 Visual description of ptychography: When taking a ptychographic scan, we scan the beam over as much of the sample as we want to image. Image from [8].

1.2 Beam Stabilization Motivation

One assumption that ptychography makes is that the beam is staying stable. If the beam is moving uncontrollably, then the position of the beam is not known properly, meaning that the stitching process in the reconstruction starts to make faulty predictions. This beam drift causes serious problems for the reconstructions. This is exasperated by the nonlinearity of HHG [9]. To fix this problem, I am developing a method to actively stabilize the beam .

In the rest of this thesis, we will be describing the processes we use to reconstruct images and stabilize beam motion. After that, we will discuss the results of these improvements.

Chapter 2

Methods

When we decide to image a sample, there are two main steps: data collection and data processing. Data collection is when we use HHG (or some other light source) to scan our sample with high-energy coherent photons, and measure the diffraction patterns. Data processing is when we take that data and use iterative phase retrieval programs to create an image of the sample. Beam stabilization is performed during the data collection phase, but the positive results are seen in the data processing phase.

In this chapter, we will first discuss the HHG setup in U150 ESC. Then, we will give an overview of how to use Tike to perform phase retrieval reconstructions. Finally, we will describe how the active beam stabilization code corrects for beam deviations.

2.1 HHG Setup in U150 ESC

Our lab in U150 ESC uses a HHG system [10]. See Figure 2.1 for the layout. This setup uses a Solstace Ace pulsed laser (800 nm). For more information on how this is set up, look at reference [10]. This laser produces infrared light, which is used for HHG. While describing how HHG works is beyond the scope of this paper, the produced light is tuned to either 42 or 57.2

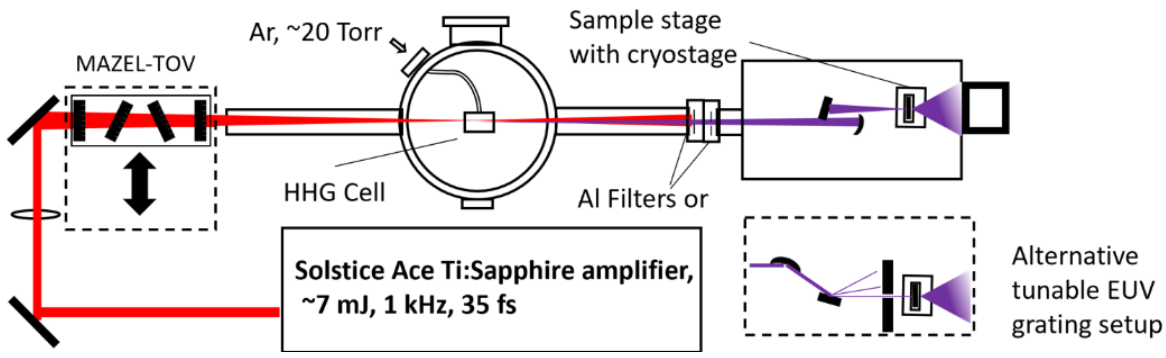

Figure 2.1 HHG setup in U150 ESC Diagram: In U150 ESC, the HHG setup follows this general pattern. This figure is taken from [10]. An infrared pulsed laser is produced by the Solstice Ace (bottom center), then might pass through the MAZEL-TOV (a polarizer). It is then focused into the gas cell where HHG occurs. The beam has the infrared components filtered, leaving an x-ray beam for scanning the sample.

eV [10] (significantly higher energy than infrared). This is produced when the infrared light is focused into the HHG cell.

2.2 Tike for Phase Retrieval

Tike is a Github repository [5] that has many common algorithms, such as the ePIE (extended ptychographical iterative engine) algorithm [11], least squared gradient [12], and error reduction [13]. Taylor Buckway used Tike to create a ptychography reconstruction code, which we have used here.

An overview of the steps of the code are as follows:

2.2.1 Import the Measured Data

Every data collection method uses a different format, but most put data into the hdf5 format. The code to extract data will be different depending on the data source. This processing takes place via the load.py file. After processing, this data is stored in a data.data variable, no matter the source.

2.2.2 Pre-Process Data

This is where the data is binned and cropped if desired, has noise reduced with a filter, and low signal removed via thresholding (data_processing.py). The parameters in this part need to be tuned depending on the dataset. The ideal post-processed data will have the diffraction pattern be bright, while any other part of the image should be set to zero. For testing reconstructions, this is where cropping and binning occurs. Binning and cropping will make the reconstructions run faster. Remember that binning in the Fourier domain (diffraction patterns) causes the real domain (sample) to be cropped, and vice versa (Fourier cropping creates real binning).

2.2.3 Run Algorithms

This is where we use Tike's rPIE, least squared gradient, and other phase retrieval algorithms. Often, trial and error is the way to determine which algorithm type to use. To be a bit more scientific about it, we recommend looking at the paper at citation [4].

As the algorithms run, the general shape of the object is resolved quickly. The majority of the iterations are spent on resolving the fine details, and filling out the outer areas of the object image that aren't directly targeted by the beam. See Figure 2.2 for a visual description of how the algorithms reconstruct the object shape, as well as Figure 2.3 for the image we are trying to reconstruct.

2.2.4 Save Data

We save the processed diffraction patterns, N steps throughout the reconstruction process, and the final result. We also produce images and gifs in order to quickly check how well the reconstruction went. The full data arrays are saved as .hdf5 files, which can be opened in python using the h5py library [15]. This allows for better figures to be created, or for analysis of code. The

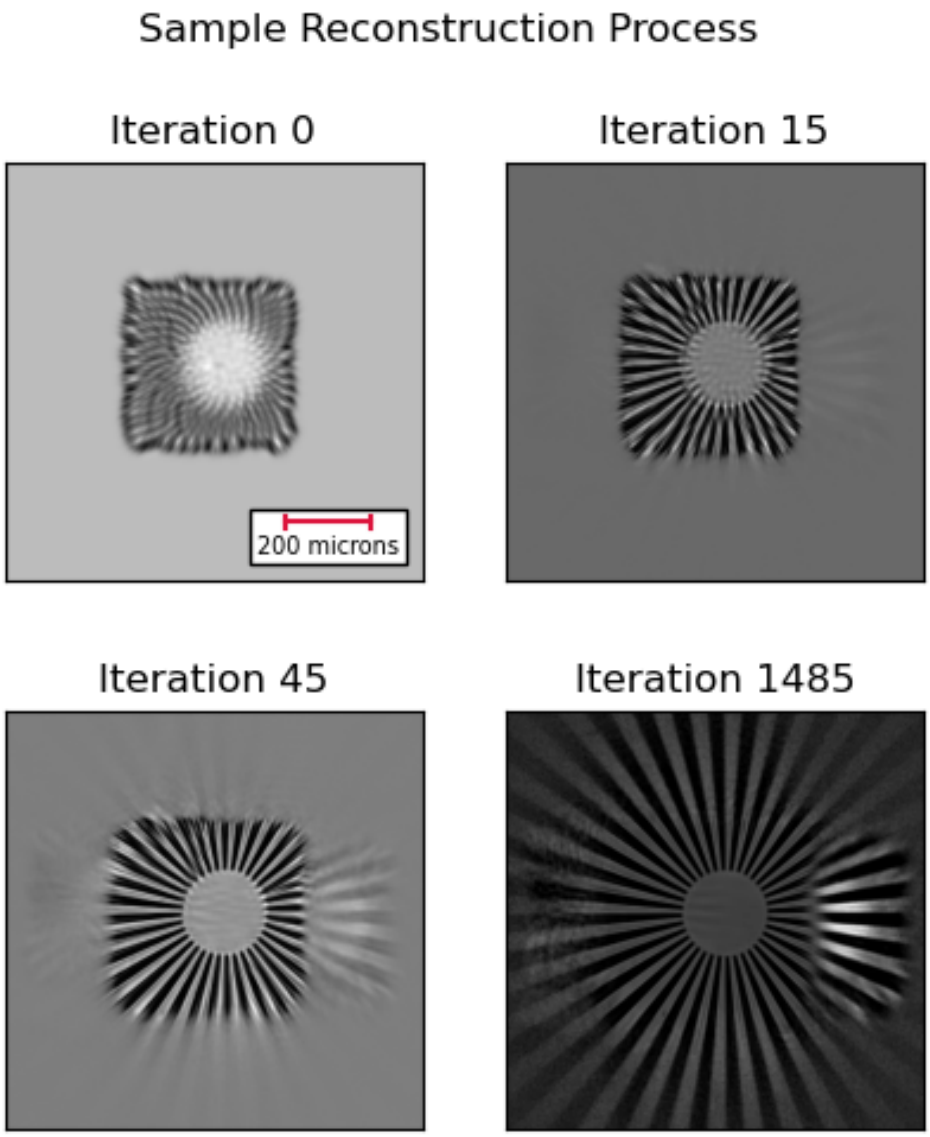

Figure 2.2 Sample Reconstruction Process: At iteration 0, the shape is nonexistent, and the only structure visible is the ptychographic scan pattern (Fermat spiral). By iteration 15, the general shape of the sample has been resolved, but it is still blurry and has weird fringes. At 45 iterations, the cente r of the scan has mostly converged to the correct shape, and only fine-tuning is required. Once we have fine-tuned the reconstruction significantly, the figure is much crisper, and the outskirts have resolved for the most part. To the right, there is a nonphysical blob due to uncorrected vertical bleed. These images come from the Stabilized Scan 1:1 dataset reconstruction.

Figure 2.3 Sector Star Manufacturers Image: This is an image from the Thorlabs website where we purchased the test sample [14]. The outer diameter is 10 mm, and the diameter of the inner circle is 200 μm. The spokes do not blur together at the center, they only appear to do so here due to limited resolution and sampling.

automatically generated images are more meant as a metric. Some of the more important images are the following:

- *reconstructing_anim.gif* can help see if the image is converging or not. This is helpful when trying to find the right number of iterations.
- *psi_phase.png* and *psi_complex.png* are the reconstructed sample images. If these don't look like anything, the reconstruction failed.
- *probe_phase.png* and *probe_complex.png* are the reconstructed probe. Generally, if the psi doesn't reconstruct, this will not either. This can be useful if trying to import a probe for a future reconstruction.

- *diffraction_scan.gif* shows the post-processed data. This can be helpful for finetuning the data processing code, as it allows you to see and compare the results of different processing methods.

2.3 Beam Stabilization

As I was developing the beam stabilization code, I tested it with one mirror and one camera, and then increased it to two mirrors and two cameras, as that is necessary to correct both beam position and angle. Strictly speaking, ptychography will not be harmed by small beam angle shifts; however, the cameras we are using do not measure EUV light and are not vacuum compatible, so we need to place them in the infrared beam. This means that we have to place them far from the sample. As a result, any angle shifts in the beam will produce a positional shift at the sample, so both types of shift (positional and angular) must be handled. When I talk about stabilizing the beam, I am referring to keeping the beam in the same location and direction over time so that the sample's movements can be properly tracked without beam movement impacting the scan location.

The code was set up to be a PI controller, a subtype of the common Proportional-Integral-Derivative controller where the derivative term is ignored. The cameras are placed in the beam, and adjusted so that the beam easily fits within the CCD without saturation. We place the cameras as far downstream as possible in order to minimize undetectable beam shifts, and use them to measure the center of mass of the beam. Knowing the location of the center of the beam on two cameras allows us to track both the position and angle of the beam as it goes into the sample.

The PI controller tries to keep the beam stabilized on the cameras. In order to tell the motors how far to move, it needs a way to turn the number of pixels to move into the number of motor steps. For this, I wrote an auto-calibration code, that moves the motors some number of steps, then records how many pixels the beam moves on the camera. The ratios between these values are then

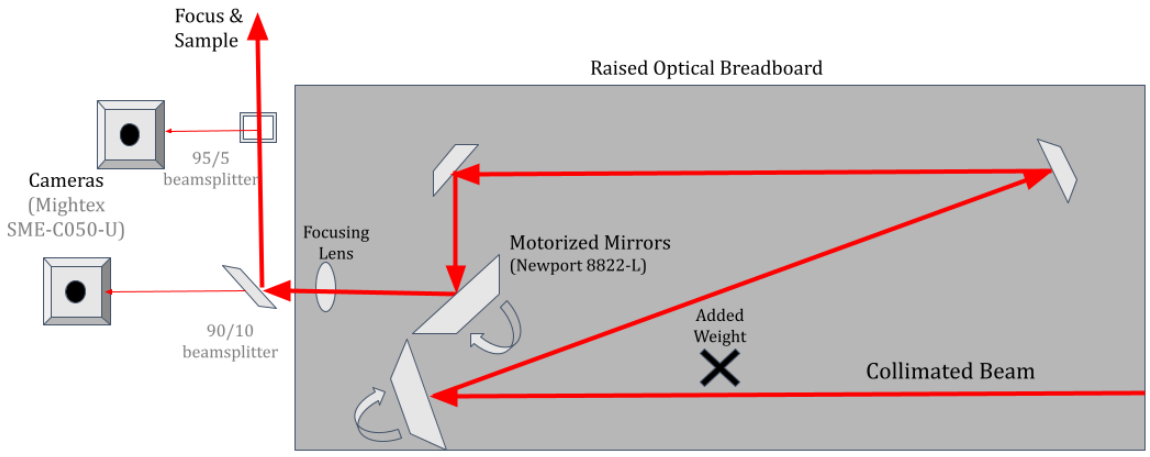

Figure 2.4 Stabilized Setup Diagram: Diagram of the test HeNe setup with two motorized mirrors stabilizing the beam on two cameras. The two cameras (on the left) record where the beam is, which is then used to decide how much to move the two motorized mirrors (with the arrows to demonstrate movement) in order to correct for beam drift. Lengths and angles in this diagram are only approximate, but the raised optical table is 30 cm by 90 cm. The weight used to introduce the error is placed at the black X.

recorded in "vals.py", and used to transform pixel error on the cameras into number of steps for the motor to take.

For a more in depth description of how the code works, see Appendix A, or see the Github repository that stores the code [16].

For the cameras, we used two Mightex SME-C050-U (Color) cameras. For the motors, we used Newport 8822-L motorized mirror mount (2 in). The setup for beam stabilization used in this thesis is shown in Figure 2.4.

2.3.1 Beam Stabilization Code Overview

The stabilization code has a large number of experimental and supporting files, but the codes that are essential to run are described in Appendix A, and on the README file on GitHub.

Chapter 3

Results

This work has demonstrated a stabilization program that significantly improves the quality of reconstructed images. This was demonstrated on a HeNe setup, and will help stabilize the HHG setup to improve the resolution toward our goal of 50 nm.

3.1 Error Characterization

In order to both develop intuition for how well the stabilization code is working and to tune the PID controller, we included in the stabilization code an error tracker that keeps track of how far off the original placement the beam is on the cameras. This tracked the error during the ptychography scans, allowing us to see the response of the stabilization code to perturbations.

It is a good idea to test code on a difficult scenario, so I added error to the setup in order to test how the stabilization code responds to large errors over time. The way that I added error was by placing or removing a textbook (930 grams) on the setup table at one minute intervals (timed by a 1 minute beep from an audio track). This introduced a small consistent error that could be duplicated between scans. Figures 3.1 and 3.2 show the difference between when the stabilization was running

and when it was not. This pixel error is the same data used by the PI controller to determine how much to move the motors.

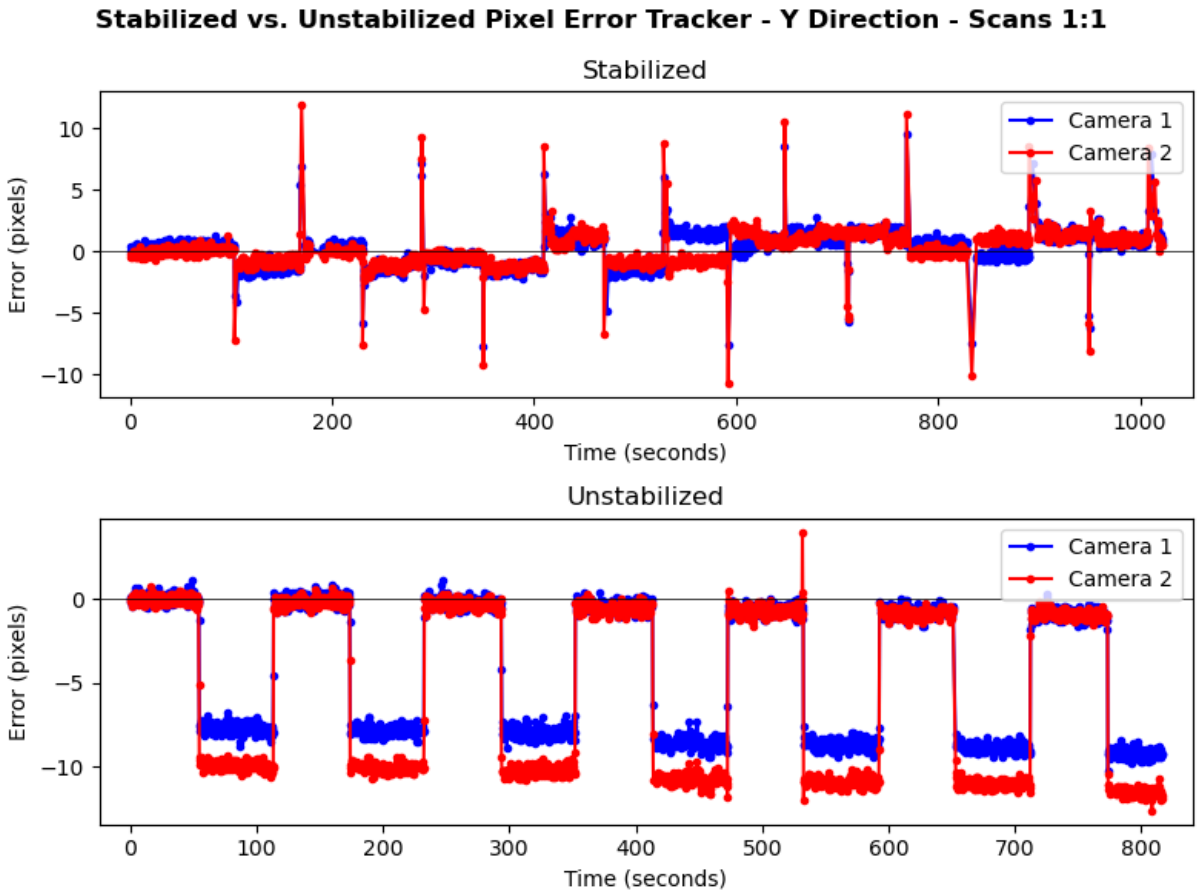

Figure 3.1 Error Over Time Stabilized vs. Unstabilized Comparison: Beam movement in the Y direction, showing that the beam drifts much less when the stabilization code is running than when it is not. Error was introduced by placing and removing a textbook on the setup once a minute. These were recorded during the ptychography data collection for scans 1:1. The top plot is when the beam was stabilized, the bottom is when it was not.

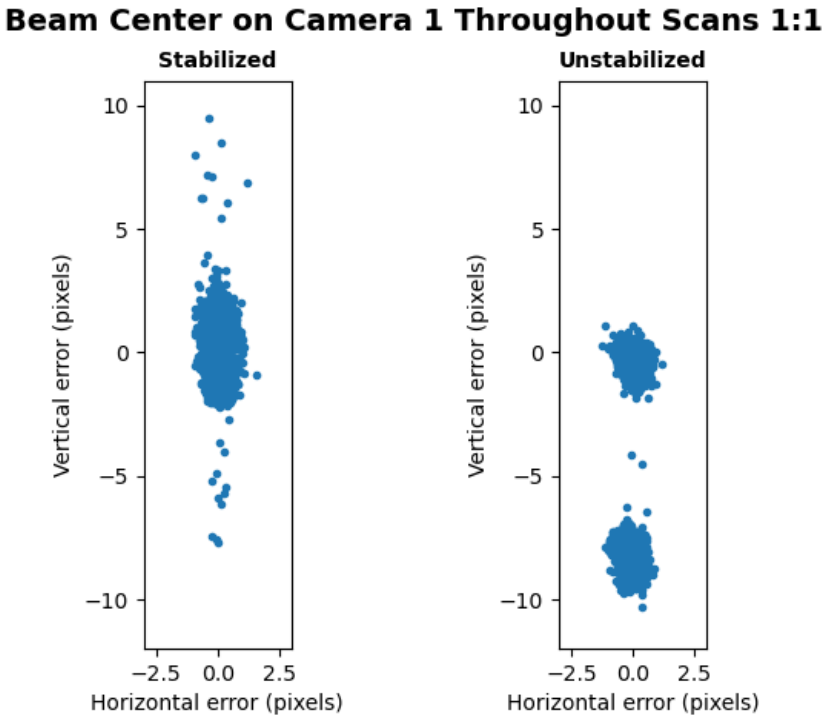

Figure 3.2 Beam Center Plot Comparison: Two plots showing where the beam center is on camera 1 throughout the course of Stabilized and Unstabilized Scans 1:1, relative to the first location recorded. The stabilized scan is concentrated closer to zero error than the unstabilized scan. Notice that the error mostly occurs in the vertical direction because the error was introduced by placing a book on an optical table, which has very little horizontal component.

3.2 Reconstructions

Once the stabilization code worked, we tested it while taking ptychography scans with and without it running. We used the same method to introduce beam error as described in 3.1 for all runs. We then used Tike to reconstruct a Sector Star test sample (Figure 2.3) for comparison. For consistency, the same script and parameters were used for all reconstructions. As we can clearly see in Figure 3.3, reconstructions from datasets with stabilization code active are much better quality than without the code running. Note that the images are a cropped central portion. The resolution improvement from stabilization will be described numerically in the next section.

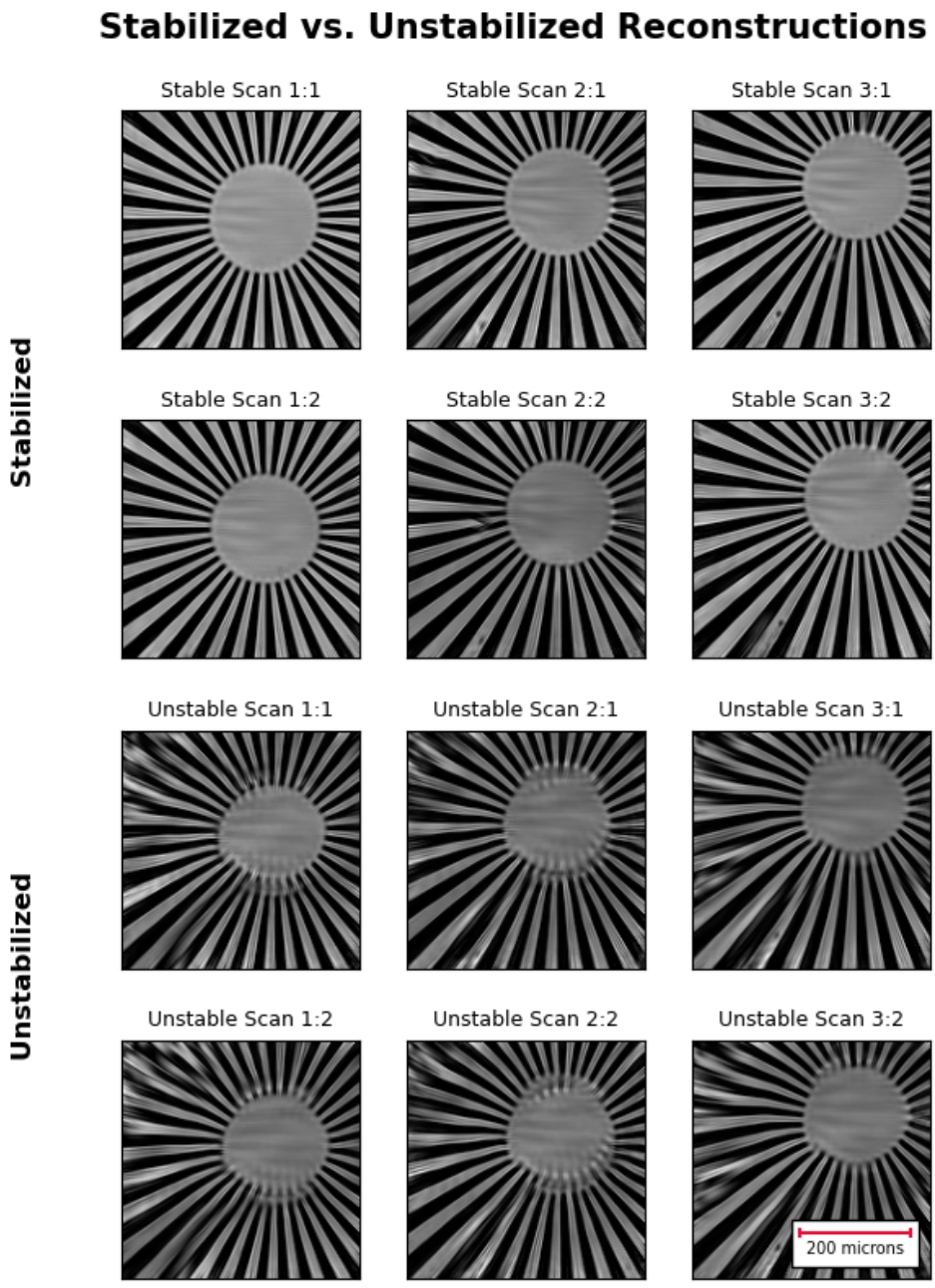

Figure 3.3 Stabilized Vs. Unstabilized Reconstructions of a Sector Star: Each of the twelve images are reconstructed from individual ptychography scans of the center of a Sector star. The images on the top were taken while the code was stabilizing the beam, and the ones on the bottom were unstabilized. It is qualitatively obvious that the stabilized scans are crisper and more accurate than the unstabilized scans. Note that these images were cropped and cross-correlated from a larger reconstruction image in order to properly use them in Fourier Ring Correlation.

3.3 FRC Resolution Comparison

Fourier Ring Correlation (FRC) is a widely known method for finding resolution for 2D images. In essence, FRC compares two images of the same object to see how much changes between the two images to see how much of the images is noise, and how much is actual data. I used FRC to compare the resolution of three pairs of stabilized reconstructions, and three unstabilized. Using the numbering introduced in Figure 3.3, I used FRC to compare Stabilized Scans 1:1 and 1:2, Unstabilized Scans 1:1 and 1:2, and so on. In Tables 3.1 and 3.2 are the resolutions in units of pixels, using the FRC $1/7$ benchmark. Pixel size is 2.8 μm.

Table 3.1 Reconstruction Resolution in Pixels

Scans	Stabilized	Unstabilized
1	2.35 pix	3.62 pix
2	2.41 pix	3.54 pix
3	2.08 pix	3.07 pix
Average	2.28 pix	3.41 pix

Table 3.2 Reconstruction Resolution in Microns

Scans	Stabilized	Unstabilized
1	6.6 μm	10.1 μm
2	6.8 μm	9.9 μm
3	5.8 μm	8.6 μm
Average	6.4 μm	9.5 μm

As we can see, the average stabilized reconstruction has a resolution length of around $2/3$ that of the unstabilized reconstructions. A 33% resolution improvement is significant, and demonstrates that this stabilization code is beneficial to ptychography.

Chapter 4

Discussion and Conclusion

In this thesis, we have verified that Tike [5] can successfully reconstruct images. Using this code, we were able to reconstruct data sets taken with and without the stabilization code running (Figure 3.3).

One limitation of ptychography in general is that it tends to have a hard time dealing with probe movement. We chose to use a stabilization system involving two motorized mirrors and two cameras to fix this problem. One strength of our stabilization method is the ease of calibration on a new system: instead of needing to create a model of the setup with precise measurements, all that is required is running a calibration code a few times to find constants that describe how much impact a certain motor movement will have on the beam position on the cameras. This makes the code adaptable to new systems with minimal effort. The software that stabilizes the HeNe code is described in appendix A, and can be found at [16].

In conclusion, we have shown that our stabilization method and code have produced clearer and higher resolution ptychography images when beam noise was introduced (Figure 3.3) with roughly a 33% resolution improvement (Table 3.2).

In the near future, this stabilization will be added to the HHG system, so that we can see improvements in the ability and ease of reconstruction on the main system that our lab is trying to

work on. This will allow for significantly better reconstructions, as HHG is a non-linear process, which amplifies any beam deviation [17]. This will help improve the resolution toward our goal of 50 nm.

Appendix A

Beam Stabilization Code

The beam stabilization code is written in python, and is backed up in a Github repository [16].

A.1 Main.py

As the name suggests, *main.py* is the main file that is run when we want to finally stabilize the beam. (Note: before running this file, make sure to read the *vals.py* and *auto-calibration.py* file descriptions). This file has a couple of main steps:

1. Callback Function: In terms of code run time, cameras take images very slowly. To avoid this lag, a "callback function" is created that can accept the camera image data. This function is passed as an argument in the camera initialization step. When the camera takes an image, it creates a new thread, and runs the callback function with that image data. The callback function is "FrameHook(info,data)". I used global variables to communicate between these threads and the original code's thread that is running simultaneously.

2. Ctrl+C Handler: Due to the fact that we are using .DLL files in this code, the code will fail disastrously if it is not terminated properly. The way I handled this was to have the code search for Ctrl+C events, and if one is detected it will break the loop at the next iteration.

3. Camera Connector: In order to make the code easier to read, I put all the camera connection code in a single function ($CameraContext(cam\_dll)$). This works as a context manager, which means that the function will run until the "yield" command, then wait until the function is closed. This allows us to connect to the camera repeatedly without reconnecting for every new image.

4. Main Loop: This is where the main structure of the code is located. The primary part of this function is a while loop that will not shut down until a Ctrl+C event is detected. Inside this loop, the code checks if new images have been taken: if so, it will use the PI controller to decide how far to move the motors. When the Ctrl+C event occurs, the code will print some useful debugging plots, print stats that give basic information about how well the code is performing, and save the error over time to a .CSV file.

A.2 Helper.py

This is a file with a variety of small functions that would have cluttered *main.py*, as well as three important functions.

- PI Calculator: PID(axis, error_tracker, n) is a function that takes a list of n recorded errors from the past 20 images recorded, and uses them to decide the ideal distance to shift the beam before the next iteration. The error is calculated using the scipy center of mass function. For specifics, look at the code.

- Print Stats: This function runs when the code shuts down. This is useful as a diagnostic tool to see how and where the code can be improved.

- Sleep Modifier: This function is not essential for the code to run, but is useful in cases where corrections need to happen at the fastest speed possible. I did not need this active for the results in this thesis. This function should be used in the same location in main.py as the camera context.

  What does this function do? For efficiency reasons, computers do not check every program if it should be running as often as possible. Often, a computer will check which programs need to run once every 15 ms. This means that trying to get a python program to sleep for less than 15 ms is not possible to accurately do without modifying this underlying behavior. The Sleep Modifier function will decrease this time to a lower time, which allows the main function to check if there has been an image taken more often than would otherwise be possible. This comes at the cost of increased power and CPU usage.

A.3 Auto-Calibration.py

One of my goals writing the code was to make it easy to transfer between systems. To do this, I wrote the code so that the only calibration when changing systems is to run this calibration code, which returns a value that tells how many motor steps are required to move the beam one pixel on a camera. Note that this does not require any knowledge of beam path length, mirror type, etc., which simplifies the process significantly. This needs to be done once for each camera-motor pair. In my case, I assumed that a movement in the x direction would have minimal impact on the beams' y shift, and vice-versa. This means that for the x direction, the calibration needs to be run for the camera1-motor1, camera2-motor1, camera1-motor2, and camera2-motor2 case. Similarly, for the y direction. These values are put into "vals.py".

A.4 Vals.py

Vals.py is a file that is meant to be a convenient location to put commonly changing values in one place. This file stores calibration numbers (see A.3), as well as file paths and camera configuration numbers.

Bibliography

[1] J. Miao, T. Ishikawa, I. K. Robinson, and M. M. Murnane, “Beyond crystallography: Diffractive imaging using coherent x-ray light sources” Science **348,** 530–535 (2015).

[2] Z. Chen, Y. Jiang, Y.-T. Shao, M. E. Holtz, M. Odstrčil, M. Guizar-Sicairos, I. Hanke, S. Ganschow, D. G. Schlom, and D. A. Muller, “Electron ptychography achieves atomic-resolution limits set by lattice vibrations” Science **372,** 826–831 (2021), publisher: American Association for the Advancement of Science.

[3] P. D. Baksh, M. Odstrčil, H.-S. Kim, S. A. Boden, J. G. Frey, and W. S. Brocklesby, “Wide-field broadband extreme ultraviolet transmission ptychography using a high-harmonic source” Optics Letters **41,** 1317 (2016).

[4] S. Marchesini, “Invited Article: A unified evaluation of iterative projection algorithms for phase retrieval” Review of Scientific Instruments **78,** 011301 (2007).

[5] D. Gursoy and D. Ching, “Tike", "https://www.osti.gov/doecode/biblio/28526”, 2022, language: en.

[6] A. Redd, “A Comparison Of Ptychography Programs For Lens-Less Imaging With A High Harmonic Generation Source” Bachelor’s Thesis, BYU (2024).

[7] D. Attwood and A. Sakdinawat, *X-Rays and Extreme Ultraviolet Radiation: Principles and Applications*, 2 ed. (Cambridge University Press, 2017).

[8] “Ptychography (Wikipedia)”, 2025, page Version ID: 1276908168.

[9] M. Odstrčil, P. Baksh, H. Kim, S. A. Boden, W. S. Brocklesby, and J. G. Frey, “Ultra-broadband ptychography with self-consistent coherence estimation from a high harmonic source” In *X-Ray Lasers and Coherent X-Ray Sources: Development and Applications XI*, A. Klisnick and C. S. Menoni, eds.,**9589,** 958912 (SPIE, 2015).

[10] T. J. Buckway, "Tabletop Extreme-Ultraviolet Source Using High Harmonic Generation for Polarization Sensitive Imaging", Master’s Thesis, BYU, 2022.

[11] A. M. Maiden and J. M. Rodenburg, “An improved ptychographical phase retrieval algorithm for diffractive imaging” Ultramicroscopy **109,** 1256–1262 (2009).

[12] M. Odstrčil, A. Menzel, and M. Guizar-Sicairos, “Iterative least-squares solver for generalized maximum-likelihood ptychography” Optics Express **26,** 3108 (2018).

[13] J. R. Fienup, “Phase retrieval algorithms: a comparison” Applied Optics **21,** 2758 (1982).

[14] Thorlabs, “Resolution Test Targets", "www.thorlabs.com/newgrouppage9.cfm?objectgroup_id=4338&pn=R1L1S2P#7495”.

[15] A. Collette, *Python and HDF5* (O’Reilly, 2013).

[16] H. Taylor, “PID-Beam-Stabilization", "https://github.com/byu-cxi/PID-Beam-Stabilization ”, 2025.

[17] M. D. Seaberg, "Nanoscale EUV Microscopy on a Tabletop: A General Transmission and Reflection Mode Microscope Based on Coherent Diffractive Imaging with High Harmonic Illumination", Ph.D. thesis, University of Colorado, 2024.

Index