6.3 Pointing Stabilization and Alignment Systems

Time-resolved ARPES has very low data throughput due to space-charge effects limiting the number of electrons that can be photoemitted per light pulse to around one electron per pulse, making data acquisition over long periods of time essential to successful studies. However, free-space Ti:Sapphire lasers are notoriously difficult to work with, and their pointing can be unstable on both fast and slow time-scales, with the former often related to vibrations from pumps or air currents and the latter largely driven by thermal fluctuations. Additionally, our broadly tunable IR pump source has moving parts in it to offer the tunability, and while the pointing drift is minimal when changing the wavelength, it is not zero, and our ARPES chamber is several meters away, making our sensitivity to pointing drift from our Ti:sapphire source and tunable mid-IR pump very sensitive. This created a problem for our time-resolved ARPES system, which I solved with a combination of cameras for measuring the pointing, piezo actuated mirrors for active control, and a custom software system implemented in Python [111] to operate the system and provide logic for the active control.

It should be mentioned that the pointing stabilization system may also be used for alignment of the laser after large drifts of the pointing, which I find is at least required every time the laser is turned off and on, using the coarse adjustment on the piezo mirrors. Of course, there are other methods of aligning the beam after large drifts in the pointing (e.g. using irises), but in my experience, these are usually only rough, requiring reoptimization of some signal to restore optimal alignment. However, the pointing stabilization system measures the pointing of the beam so precisely that when aligning the laser to this system, I find that all down stream systems are aligned as well as before, requiring no further optimization. Therefore, aligning the pointing stabilization system is a quick and highly accurate way to restore previous alignment of all downstream setups even days or weeks later, saving a tremendous amount of time.

6.3.1 Selecting Core System Components

To create a solution, I reviewed commercial options available from Thorlabs, Newport, and MRC systems; however, they did not offer detectors that would work with our long wavelength IR pump ($4 - 20$ μm), in addition to being quite costly. Thus, a custom solution would be required, meaning I needed to find position detectors and piezo controllers. There are primarily two options for position detectors: quadrant photodiodes and cameras. Quadrant photo detectors are usually preferred over cameras for their larger bandwidths, allowing for correction of faster instabilities; however, the primary concern for us is correcting slow thermal drifts to allow long-time data acquisition. Furthermore, cameras clearly acquire vastly more data than just the center of mass of the beam, which could allow for more sophisticated data processing layers to be applied in the future. Additionally, we had a legacy camera based system for the Ti:Sapphire laser, which I knew that I could expand upon. Therefore, I found some suitable IR cameras from FLIR systems, FLIR Bosons, which had spectral responsivity out towards 20 μm, which is rather difficult to find in general. The cameras use an uncooled VO_x microbolometer array for pixels. The Ti:Sapphire laser system initially used Mightex cameras, but I ultimately switched to newer FLIR Blackfly S cameras. Thorlabs MDT693B piezo controllers with Thorlabs Polaris piezo actuated mirrors were used for active control of the laser pointing. Thus, I acquired all of the core hardware components necessary to build the system.

6.3.2 Operating Theory

Optical Setup: Acquiring Pointing Information In both cases, the optical setup for measuring and controlling the pointing is the same and depicted in Fig. 6.17, where the beam pointing through BS1 is measured by cam 1 and cam 2 and controlled by PM1 and PM2. The pointing direction is measured by focusing the laser onto cam 2, because the focal position is only dependent on the angle of the beam into the lens and thereby into BS1. This is due to the fact that a lens Fourier transforms a laser beam. Meanwhile, cam 1 captures the beam before the focal

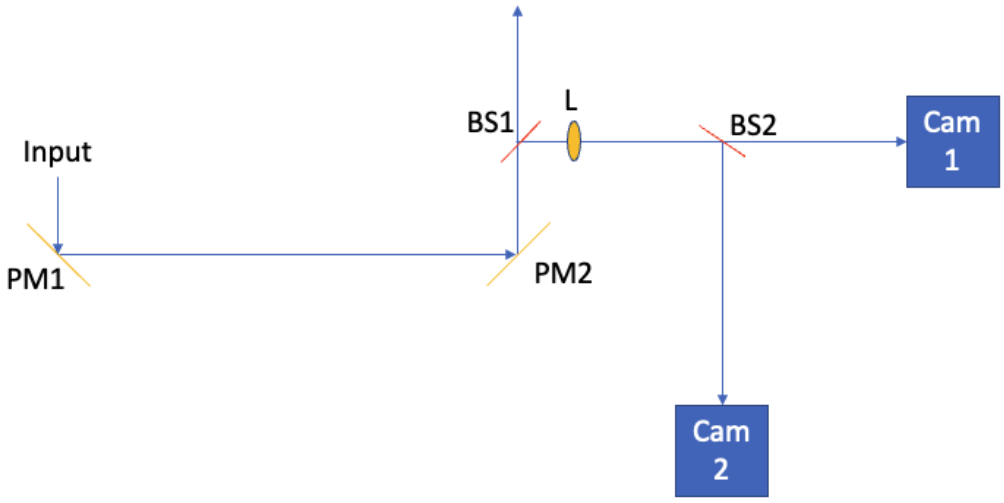

Figure 6.17: Pointing measuring system: PM1 and PM2 are piezo actuated mirrors. BS1 and BS2 are beam splitters. L is a lens. The pointing through BS1 is ultimately measured by the two cameras, Cam 1 and Cam 2, and stabilized by PM1 and PM2.

position, which is sensitive to the offset of the beam on the lens and thereby BS1. The sensitivity to the offset outside of the focus is obvious by visualizing intersecting a ray headed towards the focus from a location off-axis of the lens. Therefore, this system captures the pointing direction and offset at BS1, and by stabilizing these parameters with PM1 and PM2, the beam that transmits BS1 will also be stabilized, which is the beam used in experiments.

Of course, the vastly different wavelengths between the mid-IR and the Ti:Sapphire laser require not only different cameras but also different transmissive materials. For the Ti:Sapphire beam, anti-reflection coated UV Fused Silica is used for BS1, which reflects $\lesssim 1\%$. UV Fused Silica was also used for the plano-convex lens, f=250 mm, and BS2 (50/50). Additionally, although not shown, neutral density filters are used to avoid over saturating the camera. For the mid-IR, no coatings are broad enough to cover our spectral range. Therefore, uncoated KBr is used for BS1 to have the lowest reflection possible ($\sim 10\%$). And a ZnSe lens, f=250 mm, and beam splitter (BS2) are used. Again, neutral density filters are used as needed.

Update Logic The laser stabilization system is built around a core update logic that has

three steps: 1) calculate the position on the cameras, 2) find the displacement vector between this position and the desired position, and 3) map that displacement vector onto a new voltage to apply. For step 1, multiple options are available, but the most robust choice is actually a simple center of mass calculation. Obviously step 2 is just the difference between the set position and the new center of mass, $\vec{d}$, which is a 4-d displacement vector (2 coordinates on each camera); however, at this stage one can implement a PID controller by putting the displacement through a PID formula:

$$\vec{d}_n^{\text{pid}} = P\vec{d}_n + I\sum_{i=0}^{n}\vec{d}_i + D(\vec{d}_n - \vec{d}_{n-1}), \tag{6.4}$$

where P, I, and D are constants that control the strength of the proportional, integral, and derivatives, respectively. The parameters, P, I, and D are often just chosen manually to optimize stability of the pointing.

Finally, the key logical component of the update is the mapping from $\vec{d}_n^{\text{pid}}$ to some change in the voltages, $\delta\vec{V}_n$, to apply to the mirrors to remove this displacement, restoring the pointing to the desired positions. This mapping is found by sweeping the voltages on the mirror and looking at the change in position, which can in general be fit by a polynomial expansion in powers of $\delta\vec{V}_n$. However, it is theoretically justified to use only a linear fit because 1) the hysteresis of the piezos is small, making the displacement of the piezos linear in voltage, and 2) under small angle approximations, the position displacements (offset and angle) are linear in displacement of the piezos. To illustrate this, see Fig. 6.18. Neglecting the shift in the surface position due to the piezo expansion, the changing beam pointing in 1D can be accounted for by the changing surface normal of the mirror, which shifts by $\delta\theta$. From basic trig, $\delta x/M = \tan(\delta\theta)$, where M is the distance between the pivot point and piezo stack for the mirror mount. However, $M = \mathcal{O}(1")$ for a 1" mirror, and $\delta x = \mathcal{O}(10\mu m)$ for the full range of the piezo. Therefore, $\delta x/M = \mathcal{O}(10\mu m/1") = \mathcal{O}(10^{-3})$ is small enough to justify the small angle approximation for the resulting $\delta\theta$, i.e. $\tan(\delta\theta) \approx \delta\theta$, making $\delta\theta \approx \delta x/M = \mathcal{O}(10^{-3})$ radians. Therefore, $\delta\theta \propto \delta x \propto \delta V$. As mentioned before, the camera in the focus is directly sensitive to the angle of the beam pointing, thus the position on this camera is linearly proportional to δV on the piezo. As for the displacement of the beam in the

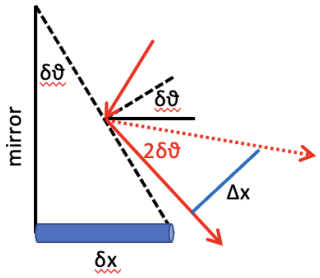

Figure 6.18: Illustration of linearity of displacement of the piezo (δx) to the angle (2δ) and displacement (dx) of the beam leaving the mirror in 1 dimension. The mirror surface (black line) is displaced about a pivot point to a new surface (dashed black line) by the piezo displacement (blue cylinder). This shifts the surface normal (black line) to a new normal (dashed black line). Finally, the original beam path (solid red arrows) is shifted to a new beam path (dashed red arrow) due to the deflection by the piezo. Note, nothing is drawn to scale—figure is only illustrative. Furthermore, the shift in the position of incidence on the mirror surface is not shown for simplicity.

direction perpendicular to the original propagation, Δx, $\Delta x = \tan(2\delta\theta)D \approx 2D\delta\theta \propto \delta V$, where D is the distance of propagation from the mirror to where the displacement is measured (i.e. the lens) and the small angle approximation is again employed. Note, the neglected contribution to the displacement of the beam from the displaced position of the mirror surface and thus position of incidence of the beam on the surface is also linear in voltage for similar reasons. Clearly, the 2 degrees of freedom (DOF) (displacement and angular change) is the reason we need 2 mirrors (with 2 DOF each to handle the vertical and horizontal planes) to fully control the laser pointing. Systems that only control the angle and neglect displacement are also possible with only one mirror. In any case, the point of the above is that $\vec{d} \approx A\delta\vec{V}$, where $A_{i,j} = d_i/\delta V_j$, and A is found in the calibration process that sweeps voltages and measures changes in position. Finally,

$$\delta\vec{V}_n = A^{-1}\vec{d}_n^{\text{pid}} \tag{6.5}$$

is the mapping that we need to stabilize the pointing. Here, $\delta\vec{V}_n$ is the voltage change that would have induced the change in pointing based on the calibration process. Therefore, subtracting $\delta\vec{V}_n$ from the current voltages will restore the pointing.

Unfortunately, the finite range of the piezos means that sometimes the pointing drift is too large to fully restore. So, the update logic must decide how to handle this case. My algorithm

first looks to additional DOF (if present or skips this step if not), then if the pointing cannot be restored, based on the user input, quits locking or shifts to locking only the angle of the pointing allowing drift of the offset. My system allows the user to use additional mirrors to compensate the drift enough to allow the fast mirrors to continue updating as described above. If used, there are more than 4 DOF for 4 positions, and the full matrix mapping from position to voltages that is found during the calibration may not be inverted. So, I separate the DOF into 4 DOF used for fast updates as described above, which has an invertible matrix, and 2 DOF for slow updates whenever the 4 fast DOF are not enough. When the update voltage is found to be outside of the allowed voltages, I look for a step along the null vectors of the full matrix with all DOF that brings all motors to a voltage inside of their allowed ranges. I check for the existence of such a step by breaking the 6 inequalities ($0 \leq V_i \leq 150$ V) for the 6 voltages into pairs, which have 2 DOF for 2 equations (step-size along each of two null vectors). Thus, the inequality equations may be solved for the step-sizes, which then defines a parallelogram that bounds the step-sizes that bring the voltages into their bounds. Repeating this for all 6 equations gives 3 parallelograms, and I look for their overlapping region using Shapely [112]. If an overlapping region exists, then I look for the smallest step size in the region and add this step to the initially out of bounds update. Since the step is along null-vectors, this still restores the pointing by virtue of the initial calculation of update voltages, and now all voltages are in bounds. If there is no overlapping region of the parallelograms, then no solution exists that restores the pointing; so, I take the step that brings the pointing closest to restoration. Then, I proceed with the steps described above where the offset center of mass is allowed to drift as minimally as possible, while the angular coordinate of the pointing is fully restored, which is more important. However, the user may require the system to quit locking in this case. In any case, if the angular pointing cannot be fully restored, locking is forcibly stopped, requiring coarse adjustment by the user before continuing.

6.3.3 Software Architecture

The software can be broken into three primary functions: communication with the hardware (cameras and piezo motor controllers), implementation of the update logic, and communication with the user through a Graphical User Interface (GUI). The hardware communication handles tasks such as grabbing frames and setting voltages, using the hardware's respective SDKs, and it is handled by 4 objects, two camera and two piezo motor objects. The control logic is implemented by an object that is constantly listening for new frames, using them to calculate update voltages, which are then sent to the piezo control objects that apply the voltages. Finally, the GUI is used to change parameters (e.g. exposure time of the camera), update the GUI with information (e.g. position on the cameras), and initiate the stabilization process. The GUI functions are all implemented in another object.

The software is implemented in Python, primarily using PyQt5 [**noauthor˙pyqt5˙nodate**], which is a python implementation of Qt. Qt is an event based programming package that implements and runs a GUI and allows communication between objects via sending signals that trigger slots (methods). Qt also allows easy multithreading, allowing each of the above mentioned objects to be on their own thread with independent event loops. A thread's event loop is essentially an infinite loop that peels off and processes events received in order or executes an efficient wait command until new events arrive. For example, the cameras send a signal indicating a new frame has arrived along with sending the actual frame. This signal then queues on the UpdateManager's event loop, and when the UpdateManager's event loop gets to this event, it triggers the UpdateManager to run its frame processing function (or in Qt lingo it calls the process frame slot). By running individual event loops and using queued connections between objects on different threads, Qt avoids any race conditions in the multithreading process. Although python has a global interpreter lock that prohibits python from running on multiple threads simultaneously, many python packages, including Qt, call external (non-pythonic) code that releases the global interpreter lock, allowing other threads to run concurrently during the execution of external code. Thus, multithreading in

python does still vastly improve the overall speed of the update loop—time to grab new frames and apply a new voltage ($\sim$ 500 ms $\rightarrow\sim$ 30 ms, where the latter is limited by the frame rate of the camera). However, the global interpreter lock is still a limitation to multithreading efficiency in python at times.

6.3.4 GUI Overview

The application's GUI is broken into a set of controls that are always visible and four different tabs: camera view, pointing view, piezo view, and unlock report. The ever present controls include a "Lock" button that initiates/stops the locking algorithm, a "Define New Home" button that captures and stores a new set position, an "Align" button that brings the piezos to their center voltage for alignment. Additionally, the user may decide whether the program should quit locking when the piezos go out of bounds after a settable time in seconds by checking the "Force Unlock" checkbox, or they can allow the program to continuously attempt to restore the pointing as best it can. I have noticed that sometimes the pointing will drift out of piezo bounds for only a short period before returning into bounds. The user can also choose to suppress various image, pointing, and piezo plots, which can be useful if the computer is struggling to keep up with the demands of the application, which can happen with cheaper computers. Finally, a file drop-down menu allows the user to initiate the calibration process, save the state of the system (all of the settings such as cameras to use, exposure times, etc.), load the home/set position by date, and load the state by date (allowing faster startup).

Figure 6.19 shows the GUI with the camera view tab displayed. On startup, this tab displays a "load recent visible" and "load recent IR" button, which when clicked loads the most recent state of the visible and IR systems for immediate use without selecting each parameter again. Alternatively, one can select the visible or IR system with the drop-down menu at the top of the tab, which then loads a view similar to that displayed in the figure. This view allows the user to select the cameras with a drop-down menu showing available cameras on the computer. Then, the user may set relevant parameters to the type of camera shown. For the visible, this includes

exposure time, gain, threshold for background subtraction, possibly averaging over frames, and generating and selecting a ROI on the camera graphically. Setting an ROI can increase the frame rate of the camera. The controls are set with an update button. Importantly, this view also allows the user to view a live stream of the incoming frames from each camera, with yellow cross-hairs showing the center of mass, and a green (locked)/red (unlocked) circle showing the set position. The color scheme of the images may be controlled as well.

Figure 6.20 shows the GUI view of the pointing view and piezo view tabs, which show a time series of all four degrees of freedom of the center of mass and piezo voltages, respectively. The pointing view tab also provides control over how many points are plotted in the figures, and gives the user the ability to acquire the piezo and/or center of mass data for a time in "[days, hours, minutes, seconds]." The piezo view tab also allows the user to select the motors to be used with the system. Additionally, the PID parameters may be set on this tab. Finally, the user must specify four channels on the motors to use for updating the voltages on each update. Additionally, the software allows the user to select two additional channels/DOF to change whenever the fast motors cannot restore the pointing, allowing longer stabilization times.

Finally, Fig. 6.21 displays a log of incidents when the pointing could not be fully restored due to the drift exceeding the compensation range. This report displays the time that the incident first began, how long the pointing stayed outside of the control range, and how far outside of the control range the pointing drifted.

6.3.5 Characterizing the System

Fig. 6.22 shows 10 minutes of laser pointing data when locked (actively stabilized) and unlocked. The figure shows both time series and frequency data. From the data, it is clear that the algorithm achieves its primary purpose—removing slow thermal drifts that are large in amplitude and thus more disruptive. This can be seen directly in the time series, and it is also clear in the frequency data that low frequency noise has been suppressed. Furthermore, when there is significant drift over long timescales, it is clear that the standard deviation of the pointing is significantly

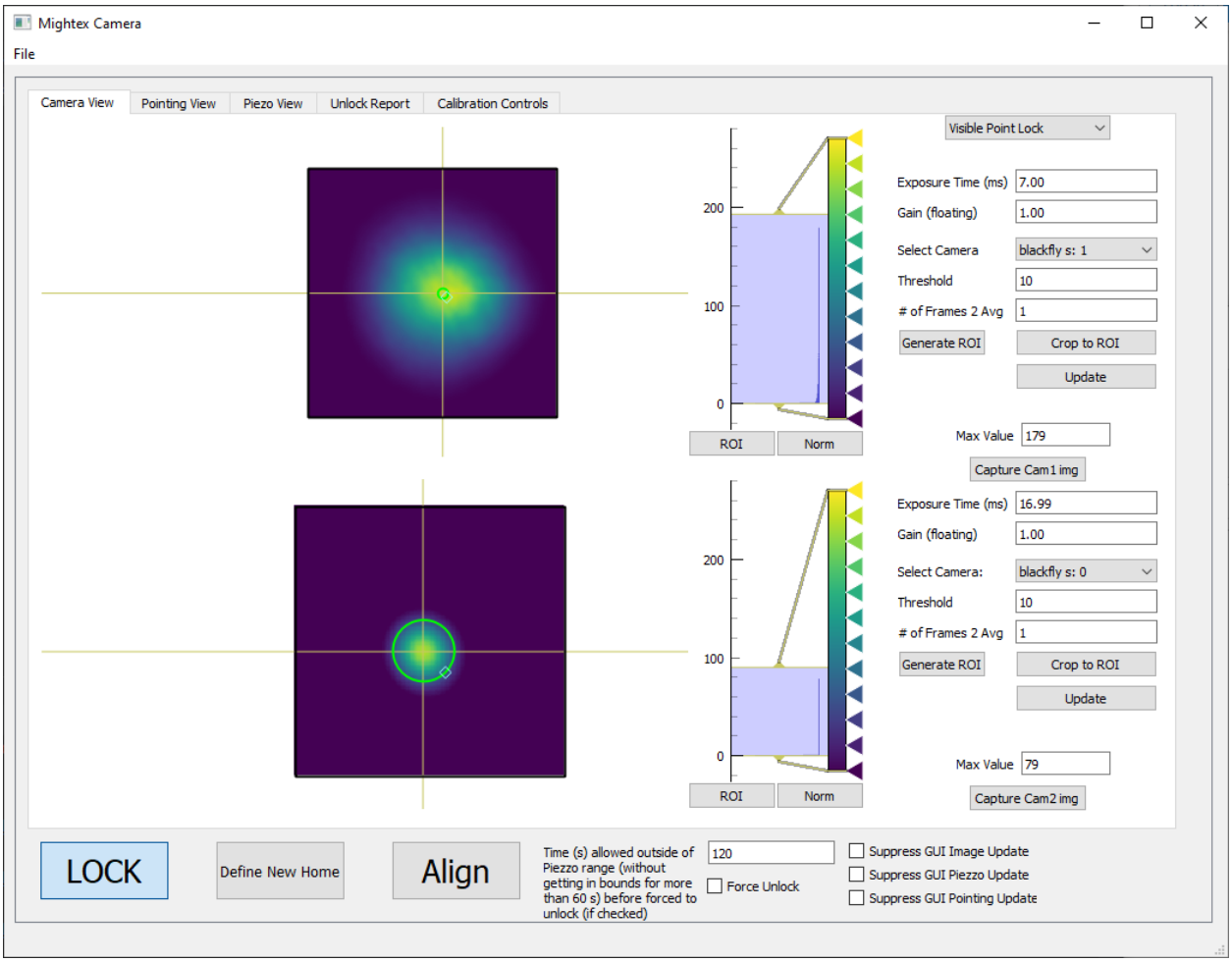

Figure 6.19: Point lock GUI Camera View: Figure displays the full window of the application GUI with the camera view tab selected. User can choose whether an IR or visible system is used, and then choose the cameras and set relevant camera settings in this view, as well as see returned images.

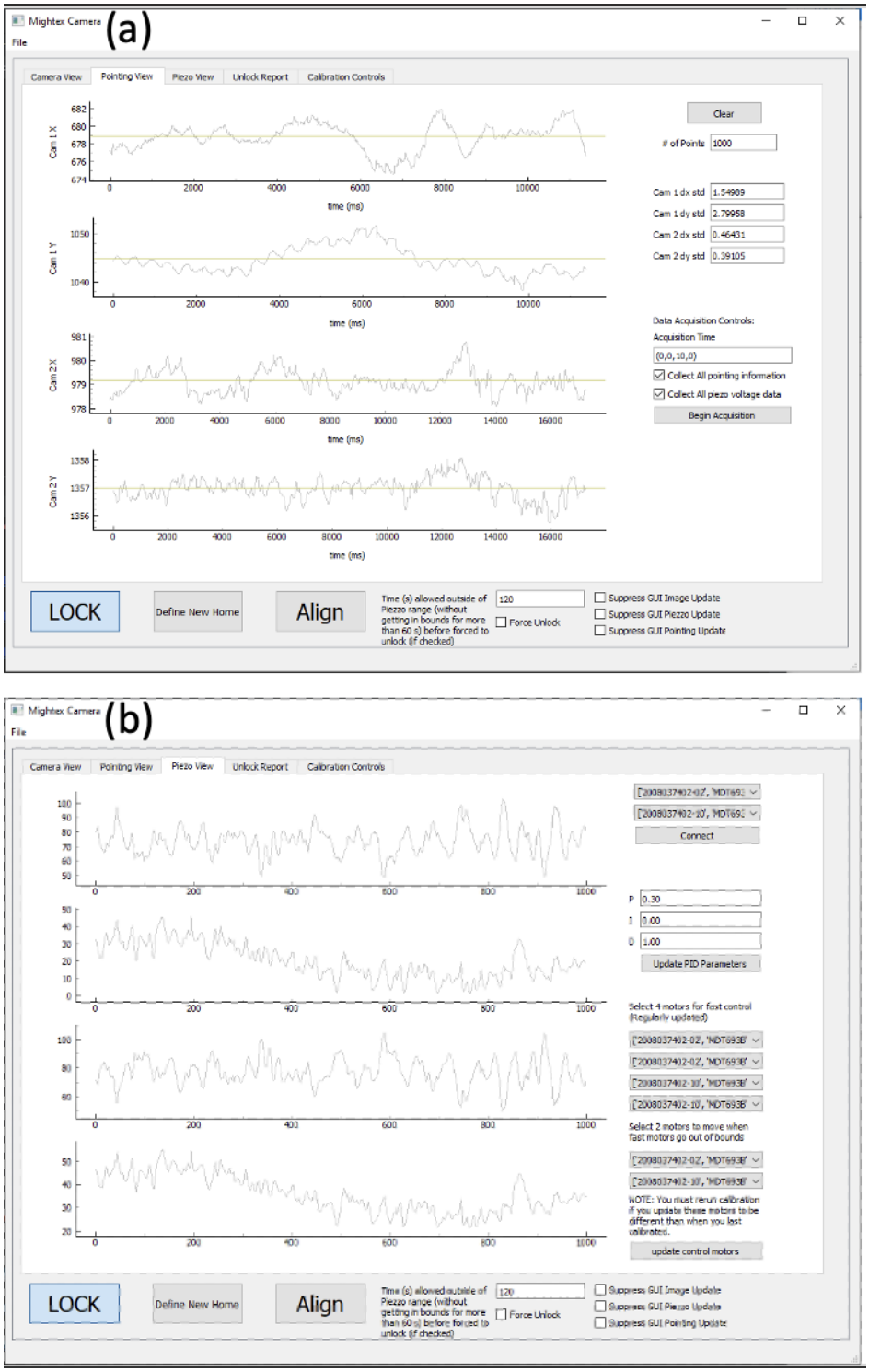

Figure 6.20: Point lock GUI line plots: a) Shows the GUI view of the pointing view tab. In this tab, the user can see line plots of the center of mass coordinates, controlling the number of points plotted. User can also initiate data acquisition. b) Shows the GUI view of the piezo view tab. This shows plots of the Voltages (Volts) vs time plotted as index of voltage applied (right current, left past). The user can also select the motors and what channels to use for control (fast) vs. slow adjustments.

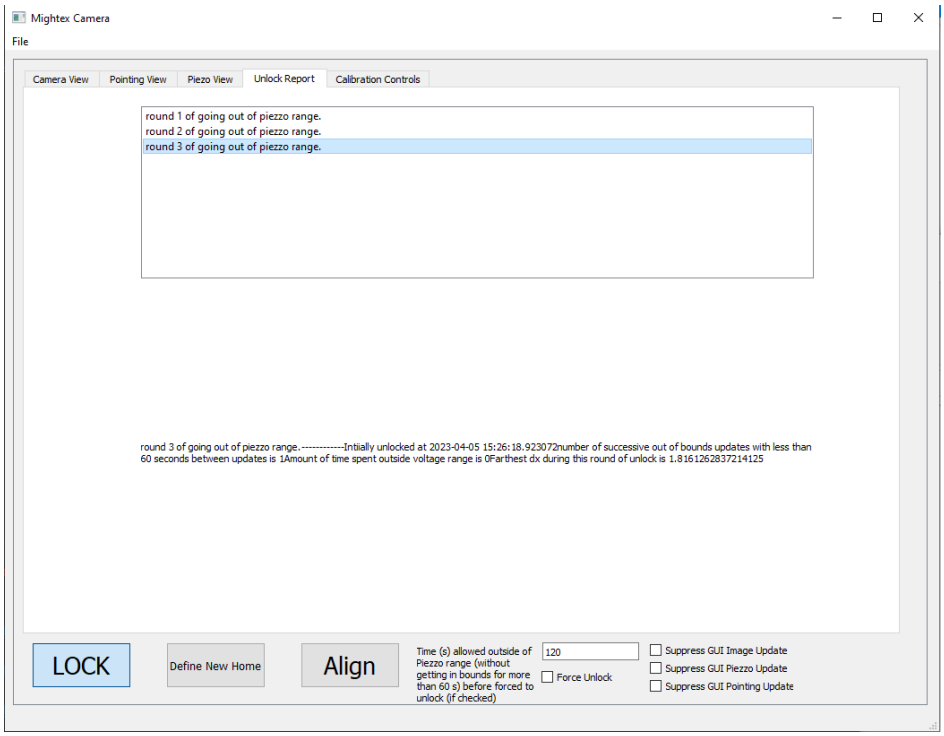

Figure 6.21: Unlock Reporting: Figure shows the GUI view of the unlock report tab. This tab displays a log of instances where the update voltages went out of bounds, forbidding a complete restoration of the pointing. Logs include information about when this occurred, for how long, and how far out of bounds the pointing drifted.

reduced. However, it is also apparent that the system really does not improve noise above about 125 mHz, which is because the data shown is from when the system was forced to average over many frames, reducing the bandwidth of the stabilization. This was to reduce the introduction of unwanted high frequency noise, while continuing to stabilize low frequencies. This is an effective quick and dirty trick; however, a better approach would be to tune the PID parameters more carefully, which will be implemented in the future.

6.3.6 Limitations

While the system performance is excellent, there are limitations that should be kept in mind. First and foremost, the piezos have very little throw! This means that they can only correct for a narrow range of pointing drift, limiting how long the system can stabilize the pointing. Therefore, longer throw piezos are more desirable; however, there is a limit to what is commercially available. Unfortunately, the pointing drift depends on many many factors and can exceed this limit. To an extent, this can be overcome by daisy chaining together multiple mirror actuators with integrated piezo stacks (see Fig. 6.23). We have done this, and it works great, but our laser is so unstable that even 2x the throw is not enough for multiday scans. An ideal solution would be a slow but long range motor with an integrated piezo motor, but these are not available; so, I came up with my slow motor algorithm described above, but I have not implemented it with a different type of long-range motor yet, allowing only additional piezos to be used thus far, which still does not overcome the pointing drift of my laser. Second, it is important to note that what we really care about is the true laser pointing, and our system uses the center of mass of the beams as a proxy for this. In fact, most systems employ a quadrant detector that is also using the center of mass as a proxy for the pointing information. However, the laser beam mode also changes with time! This can be caused by natural changes in the laser and also by the beam clipping on an optic upstream of the stabilization system as it drifts. It can also be caused by the beam drifting onto a defect on the optic. All of these will change the calculated center of mass coordinates even if the beam pointing does not change at all. Therefore, correcting for this “pointing drift” will actually induce

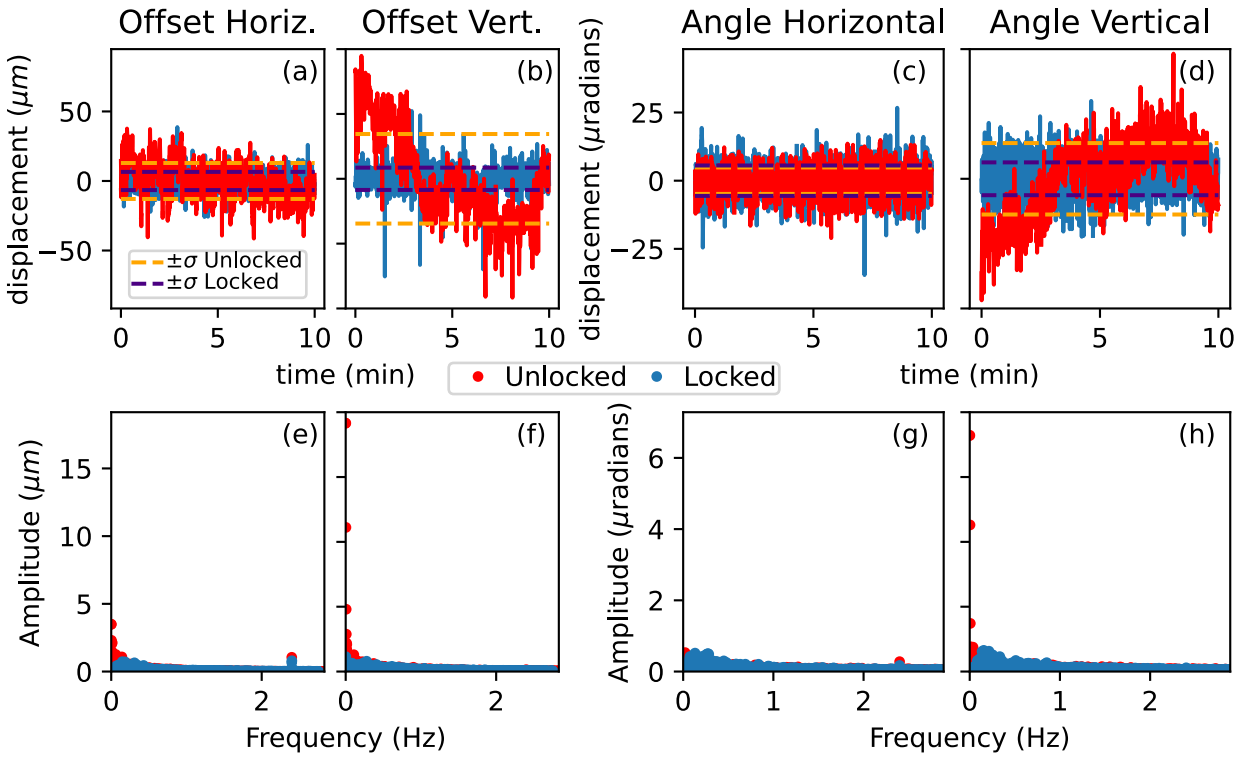

Figure 6.22: Point locking characterization: a-d) time series of the displacement of the center of mass on each camera, and the dashed lines represent the standard deviation of the data. e-f) Magnitude of the Fourier transforms of a-d, respectively. a,b,e,&f) Come from the camera outside of the focus, and thus, the displacement is converted from pixels to microns using the pixel size of the camera. c,d,g,&h) Come from the camera in the focus, which can be converted to a change in angle, $\Delta\theta = \arctan(\Delta x/2f)$ [113], where $f = 200$ mm is the focal length of the lens and Δx is the displacement of the center of mass on the camera.

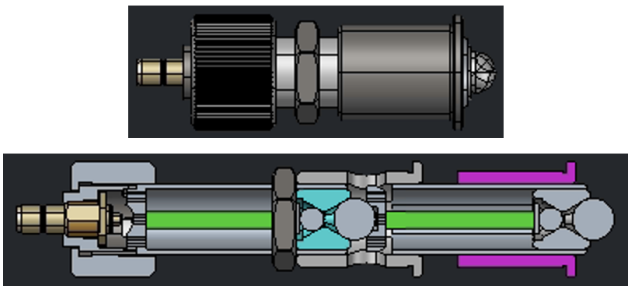

Figure 6.23: Daisy chain of multiple piezo stacks to double effective throw. Top is a single Thorlabs Polaris-P20 mirror actuator with an integrated piezo stack. These can be screwed into a Polaris mirror mount to make a piezo adjusted mirror. Bottom shows a cross section of two Polaris-P20s in series, doubling the effective piezo throw. Piezos are the green rectangles. Cad drawings of Polaris-P20 is downloaded from Thorlabs website.

unwanted pointing changes, and therefore what looks more stable to your measured system can be less stable at the systems that really matter. This is an unfortunate limitation of the system design; however, this is where a camera based system can shine, because it is possible to account for mode changes. For example, a Gaussian mixture model might be employed to detect different modes of operation that the laser is moving between, applying a different pointing target and/or calibration matrix A^{-1} for each operating mode of the laser. Regardless of the details, the point is that more information is available to account for this limitation using more advanced algorithms, possibly including machine learning algorithms. However, time limitations have rendered these ambitions projects for a future date.

6.4 Conclusion

Our time-resolved ARPES system provides a unique opportunity to test the purported light induced superconductivity seen in many systems by Cavalleri et al. My upgrades to the system make these experiments practical and improve the data quality attainable. The new window in a window and point locking systems allow a collinear geometry and precision alignment and stabilization, enabling easier initial alignment and long-time, higher quality data acquisition. The new purge system will yield higher pump intensities with less distorted pulses over a much broader frequency range.