Focal Plane Array Optical Focusing and Alignment

Aaron Greenwood
amg@nmt.edu or amgreenw@google.com

Introduction

A need for active adjustment of a focal plane array (FPA) exists for space based telescopes at Sandia National Laboratories. Legacy systems relied upon manual alignment of the FPA to the optics train. The alignment was done by checking the image then adjusting the position of the FPA with shims, repeated until the desired alignment and focus were achieved. This approach has several drawbacks, including the cost of labor and clean room time, potential for damage of sensitive equipment, and potential for loss of alignment due to launch, thermal expansion, etc. An active system will allow for quicker and more robust alignment. The system will utilize three actuators that will be able to adjust the FPA in tip (θ_x), tilt (θ_y), and despace (z). [Fig. 1] The FPA will be used as the sensor that will detect when proper alignment is achieved. [Fig. 2] To do this an algorithm is needed to process the image to generate numerical vales of alignment and focus that can be compared.

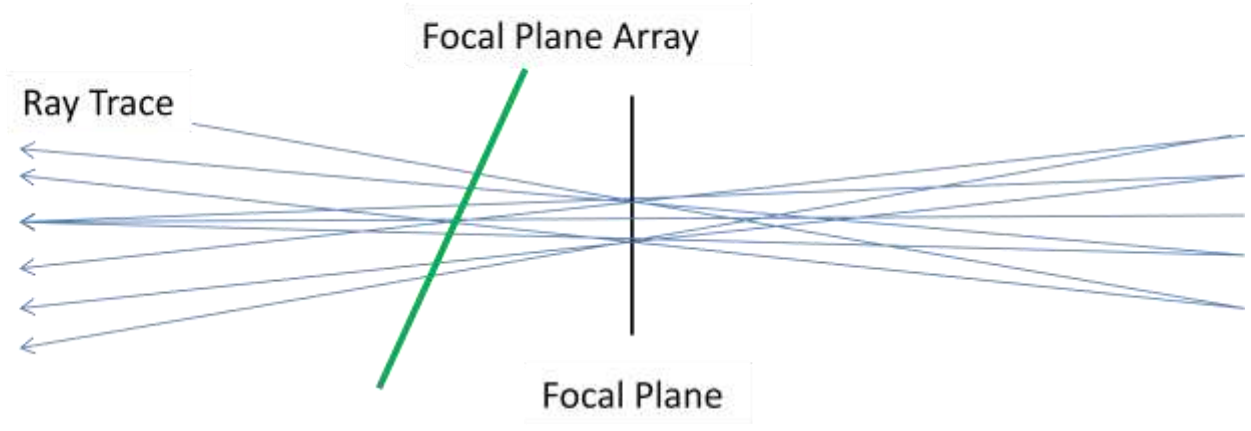


Figure 1 Diagram of light path and FPA placement

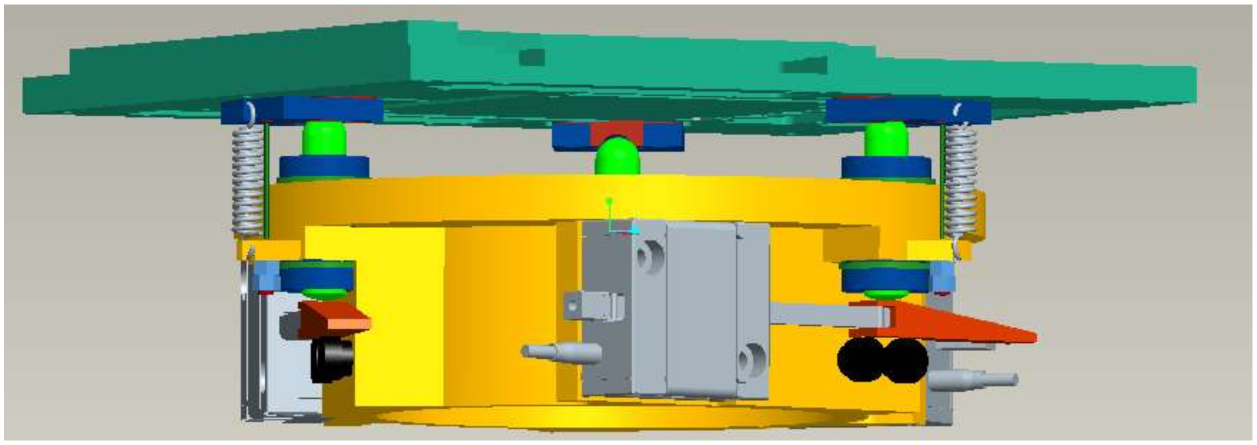


Figure 2 Proposed active alignment and focusing mechanism

Approach

The first step in developing the algorithm is developing a model to represent the images generated by the FPA when it is correctly aligned and focused as well as when the focus or alignment is off. The image will be projected as a circle onto the FPA when alignment (tip and tilt) are correct. When it is incorrect the spot will change in shape and develop a projection. When focus (despace) is changed, the image will change in area. To represent these images, eight images were generated; four for the alignment and focusing algorithms respectively.

The alignment images have decreasing amounts of eccentricity. The first three images are ovals with the final image as a circle, representing correct alignment. The focusing images are circles of decreasing diameter, with final image (smallest diameter) representing correct focus. The intensity of the spot is not constant, with the intensity a maximum in the center, decreasing as the radius increases. Additionally the images have been corrupted by additive Gaussian noise.

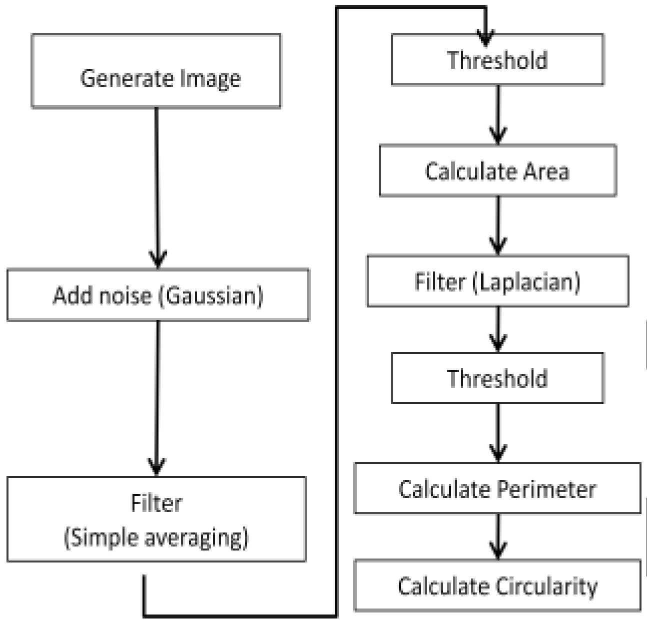
The approach is to minimize the area, for both the alignment and focus. Also the total intensity of the focusing images will be calculated so the intensity per area can be calculated. For the alignment, the circularity ratio [Eqn. 1] will be used to check the shape of the image. The circularity compares the area to the perimeter. The circularity ratio equals one when the image is a circle, and is less for all other shapes. [1]

Equation 1 $$R_c = \frac{4\pi A}{P^2}$$

To prepare the image it is first blurred using a 7x7 averaging filter. This will effectively reduce the Gaussian noise. It will blur the image extensively but the relative area size should be preserved. The blurred focus images are then summed too determine the overall intensity. All images are then thresholded too max and zero intensities. The area is calculated simply, by counting the number of pixels mapped to the higher value, giving the area of the image in pixels. This is done for both alignment and focusing. At this point the focusing processing is complete. The alignment algorithm has one more step; calculating the perimeter. This is done utilizing a Laplacian mask shown below.[Eqn. 2] This generates an image that is the second derivative of the original. This results in only the area of changing intensity being preserved. The image is thresholded to isolate the perimeter. In this case the number of high intensity pixels are the perimeter of the image's spot. [1]

Equation 2. $$Laplacian\ Mask = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

The program was divided into four separate files; two each for the alignment and focusing. The first files generate and blur the images used for the focusing and alignment. Then the result is processed in the second set to determine area of the image for focusing and the area and perimeter for the alignment. Figures 3 and 4 show the outline of the algorithms.



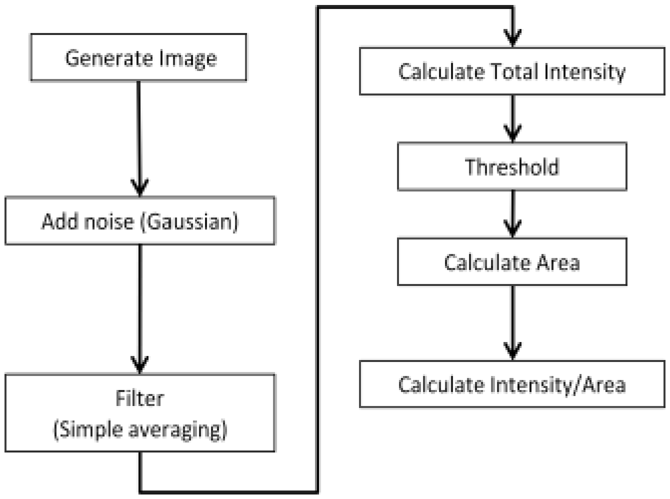


Figure 3 Alignment algorithm **Figure 4 Focusing algorithm**

The values for generated for the images would then be used to for the minimum value using hill climbing techniques. [2][3] If the above techniques are not sensitive enough to allow for differentiation, other of focus value detection can be used. Some of these methods are illuminant gradient [2] and threshold gradient [3] methods for determining focus value, as well as filtering techniques such as frequency selected weighted median (FSWM) filtering. [4]

Results

The results were satisfactory. The algorithms' were able to distinguish between the different images. As shown in Table 1, the alignment algorithm, worked well. The images were generated and processed, in under one second even using the CPU. This should be plenty fast enough for this application. There was a problem in the perimeter calculation though. The laplacian did not quite generate a perfect perimeter value, leaving the circularity ratio lower than expected. This was because the laplacian leaves more pixels than necessary, especially along the diagonal parts of the circle's curve. Adjusting the threshold values did not entirely correct the problem, but setting the threshold to set the positive part of the laplacian to max intensity and the negative part to zero gave the best result. The remaining error was corrected by using a scaling factor based on the circular image. Unfortunately, this will not be the same for exactly different size circles. This means that the general applicability is less than originally hoped, and that it may not be possible to end the alignment procedure when circularity is achieved.

Implementation in the GPU was not very successful. The programs ran considerably slower on the GPU. For despace adjust and the speedup was 0.012, and the tip adjust speed up was 0.023. The computers in the DSP lab were also much slower than my computer, which isn't extremely powerful. A program that ran in 0.4s on my computer took 36s on a DSP lab computer. The difference could be the processors, but the Matlab version could have played a role. My computer runs 2009a, while the DSP computers have 2010a.

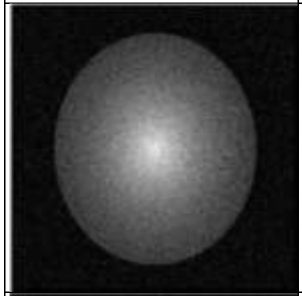
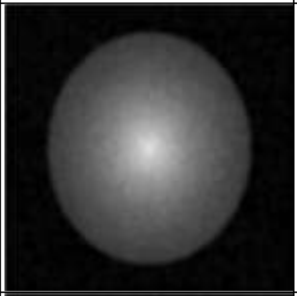
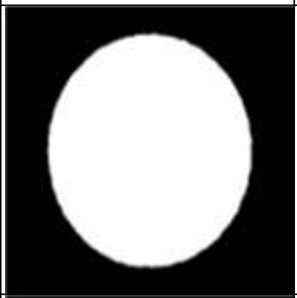
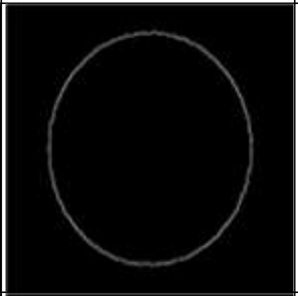
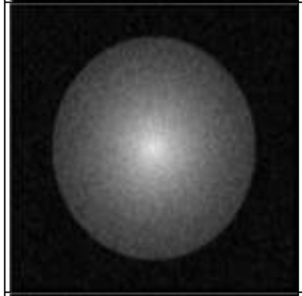
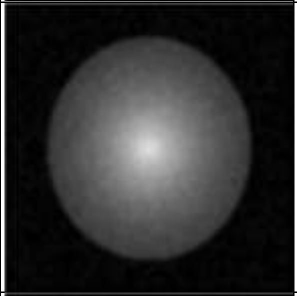
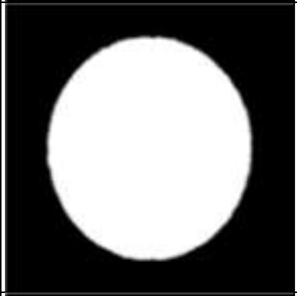
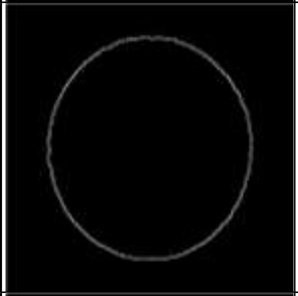
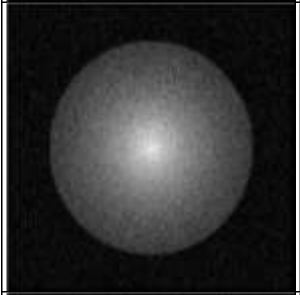
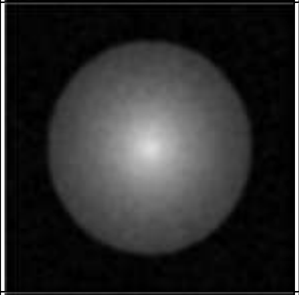
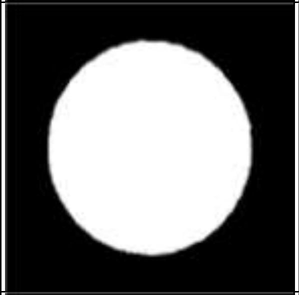
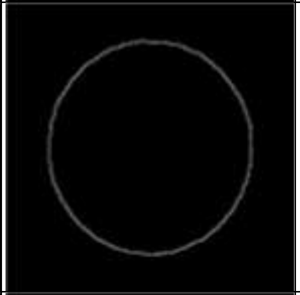
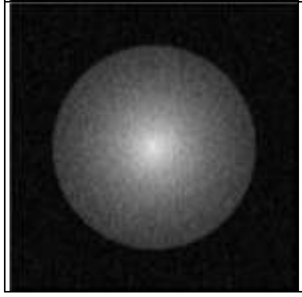
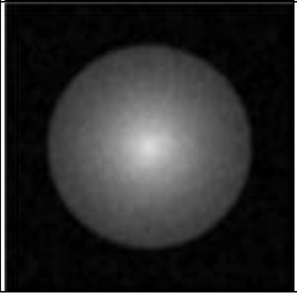
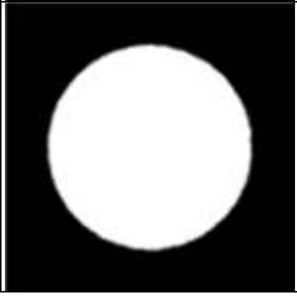
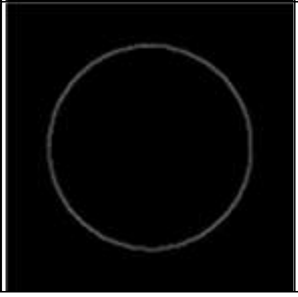
Noisy	Filtered	Threshold	Perimeter	Circularity
				0.9719
				0.9822
				0.9956
				1.0001

Figure 1 Tip image progression and circularity results

The focusing algorithm performed better, with the differences being readily apparent based on the area of the spot, even for small diameter changes, as shown in graph 2. However, it became apparent that calculating the intensity was not necessary, only serving to obscure the correct result. This is because the intensity should actually be the same for all images as the spot is entirely contained within the image, while the total area expands as focusing is lost. So all change is dependent on the area. My model was inexact though as the max intensity increased with the area, so dividing total intensity by area balanced out obscuring the result. If the spot was larger than the FPA, as in most cameras, the result would be opposite; total intensity would decrease with loss of focus, while area would be constant. The focusing was even faster than the alignment, and could be reduced further by not calculating the total intensity.

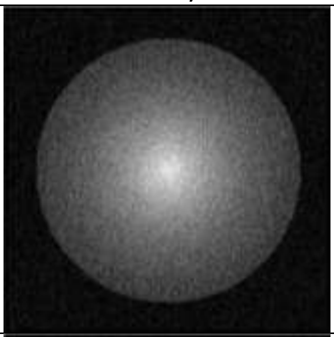
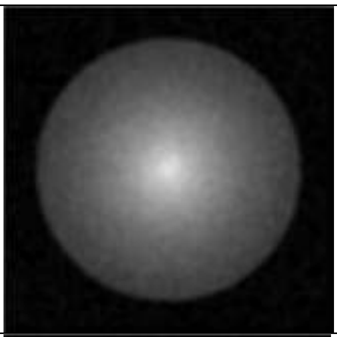
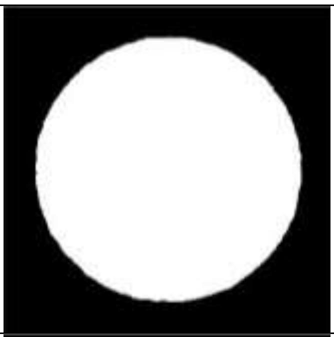
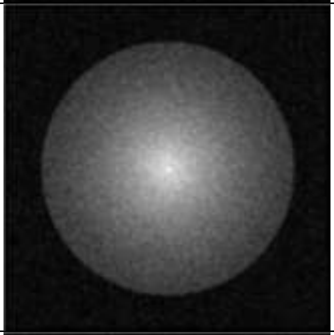
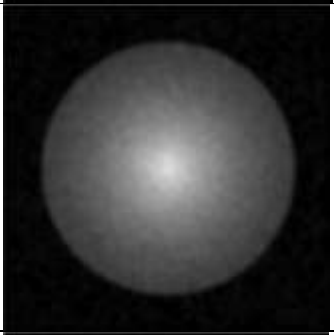
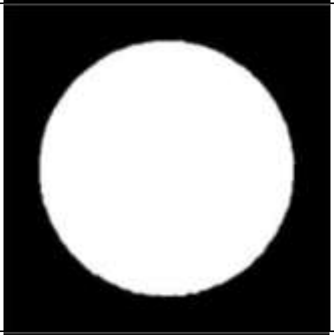
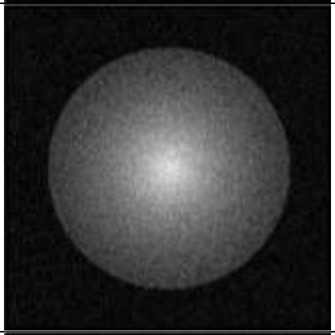
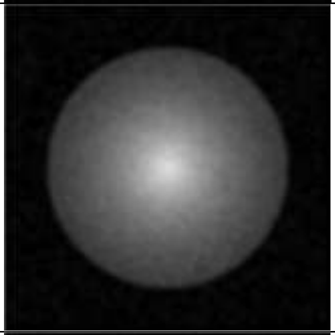
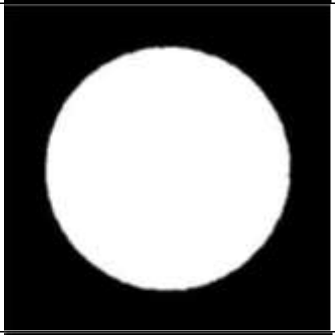
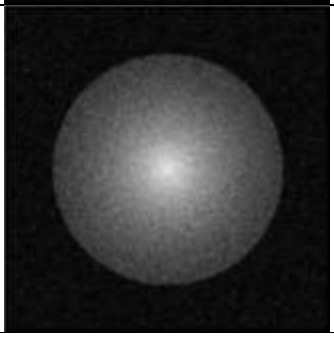
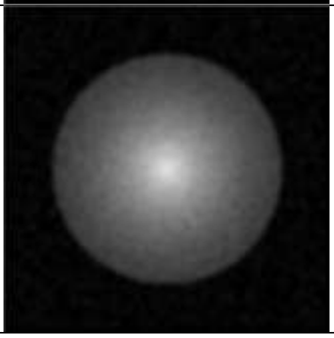
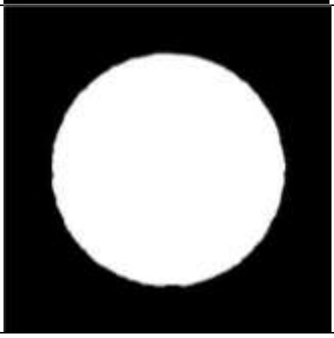
Noisy	Filtered	Threshold	Intensity	Area
			96.472	45774
			97.355	42046
			98.388	38450
			99.438	35057

Table 2 Despace image progression, intensity flux and area in pixels

Conclusion

Overall the performance was sufficient. However there is room for improvement in calculating the perimeter of the images. The GPU implementation was very slow; this is likely due to the programs which rely on nested for loops and the time taken to transfer information to the GPU and Back to the CPU. Improvements could be made by optimizing the programs for parallel processing. Implementing the full program in CUDA would also improve performance by eliminating the data transfers. Unfortunately, time did not permit this work. Future work includes developing programs for turning the data created by the processing into commands to the actuators. Further development of the image processing aspect depends on the optics train to which it will be attached. As this is designed to demonstrate increased capability for future designs, this information is not available.

References

[1] R.C. Gonzalez, R.E. Woods, *Digital Image Processing*, Third Edition, Pearson Prentice Hall, Upper Saddle River, 2008, pp 797-823.

[2] Chih-Ming Chen, Chin-Ming Hong, Han-Chun Chuang, “Efficient Auto-focus Algorithm Utilizing Discrete Difference Equation Prediction Model for Digital Still Cameras”, *IEEE Trans. on Consumer Electronics*, vol. 52 , no. 4, pp.1135 – 1143, Aug. 2006.

[3] Jie He, Rongzhen Zhou, Zhiliang Hong. “Modified Fast Climbing Search Auto-focus Algorithm with Adaptive Step Size Searching Technique for Digital Camera”, *IEEE Trans. on Consumer Electronics*, vol. 49, pp.257-262, May 2003.

[4] K.S. Choi, J.S. Lee, S.J. Ko., “New Autofocus Technique Using the Frequency Selective Weighted Median Filter for Video Cameras”, *IEEE Trans. on Consumer Electronics*, vol. 45, no. 3, pp.820-827, Aug. 1999.