M. Aurand
4.21.2010
EE552

FFT-Based Astronomical Image Registration and Stacking using GPU

The productive imaging of faint astronomical targets mandates vanishingly low noise due to the small amount of signal amplitude available. The readout noise of the camera is gaussian in nature and dominant in presence; hence a popular method of reducing its influence is image stacking. Several images can be summed to minimize this white noise as a function of $1/\sqrt{N}$. This project explores the use of MATLAB in combination with GPUMAT to create a functional program to register and stack a series of images to reduce total noise contributions.

Introduction

The images generated by the telescopes I use are typically a few megapixels in size, at 16-bit intensity depth. Each image in a series is dithered by a small amount to reduce the influence of pixel defects on the final, stacked image. Any program to process the image series must precisely identify the offsets and apply spatial transformations necessary to produce a high-resolution, low-noise final photograph.

My program performs several steps to ensure adequate and reliable operation under real-world circumstances. The FFT cross-correlation is used to measure the offset of any image with respect to a user-selected "reference" image. FFT correlation is not very sensitive to sub-pixel movements, but works well as a coarse centering tool. Once the image has been roughly centered (within an integer pixel value), several stars are measured and compared to the "reference", quantifying sub-pixel residual offsets. The rotation angle of the image is also measured to determine whether or not the image should be rotated. Scale from image to image is invariant, and needs no correction.

Below is a sequential list of operations performed by my software.

- **A. Read in reference image and image series to be registered/stacked**
- **B. Pre-process reference image**
  1. Automatically generate valid list of reference stars
  2. Quantify position and angular moment of each reference star
- **C. FFT-register (to integer pixel values)**
- **D. Centroid-register**
  1. Quantify residual offset/angular moments relative to B.2
- **E. Image transformation**
  1. Generate transformation matrix
  2. Apply/interpolate
- **F. Image summation**

A. Image opening

The images are opened using the MATLAB `fitsread( )` function, which supports the 16-bit data format of scientific instruments. Alternatively, more popular image formats such as .jpg and .bmp can be imported by the `imread( )` function. These images are allocated within GPU memory as single-precision floating points.

B. Reference Image Pre-Processing

After coarse FFT registration, star images will be used as tie points for fine adjustment of position and rotation. In order to maximize solution robustness, only the center 50% of the reference image area is utilized to generate tie stars. If one was to use stars near the edges of the reference frame, it is potentially unlikely that they will appear in the off-center images. This center 50% of the image implies that even with an input image shifted by M/2 or N/2, at least 25% of the centroid points should still be usable during fine adjustment.

In order to automate the selection of reference stars, the usable center 50% is thresholded to generate a binary image. This binary image is sorted to identify and label the points. Next, some criteria are applied to select the best stars. In order to preserve the accuracy of a center-of-mass centroiding algorithm, the stars should not be saturated nor should they be too spatially close to an adjacent star. The stars identified in the binary image are sequentially examined and the ones that are saturated or within the exclusion radius (user-adjustable) of another star are discarded. This leaves a list of only the favorable tie-point stars.

Once the list of valid tie stars has been generated, the center-of-mass centroiding algorithm is applied, and a reference feature list created. The reference feature list parametrizes each star's position and angle relative to the center of the reference image.

C. FFT Registration

FFT registration provides an exceedingly convenient way of obtaining a test image's Cartesian X and Y displacement relative to the reference image by means of correlation.

Suppose we have two identical images (f_1 & f_2), one with a spatial offset of x_0 and y_0. The spatially offset image (f_2) can be represented in terms of the other image:

$$f_2(x, y) = f_1(x - x_0, y - y_0)$$

Upon conversion to the frequency domain by Fourier transformation, a separable term A is created.

$$F_2(u, v) = F_1(u, v) * e^{-2j\pi (ux_0 + vy_0)}$$

$$A = e^{-2j\pi (ux_0 + vy_0)} \quad : \quad F_2(u, v) = F_1(u, v) * A$$

By taking the cross-spectrum of the two transforms, F_1 and F_2, A is readily solved.

$$A = F_1(u, v) F_2^{*}(u, v) / | F_1(u, v) F_2(u, v) |$$

Where A is visualized as an impulse in the cross-spectrum. It is located at the offset:

$$u, v = (x_0, y_0)$$

In order to obtain this correlation numerically, the GPU functions `fft2( )` and `ifft2( )` are employed to produce a 2D result array. This result array features an impulse centered about the measured X and Y offset of the test image, which is recorded as a rough offset. This is a coarse result (integer pixel value) because the FFT is critically sampled. Better resolution could be obtained by padding the input images to larger size, at the expense of memory and computation time. In this case, however, any extra resolution is of no utility because the centroiding method is next employed. The centroiding algorithm is significantly less computationally intensive than the FFT method, but requires some degree of preregistration to ensure trustworthy output.

D. Centroid Registration

Taking into account the FFT-derived offsets, a test image is shifted and the areas where the reference stars should lie are input to a center-of-mass (COM) centroiding routine. Each reference star location is expanded by the measurement radius to form a region of interest for each star point. Providing the rotation is within reasonable limits (±5º), the stars will now be within the measurement radius specified during the reference star selection routine (measurement radius < exclusion radius). Each measured test star is compared in X and Y to the parameters found during reference pre-processing, and angular rotation/offset residuals are averaged to generate final rotation and fine offset figures.

E. Image Transformation

Using the total linear offset and rotation figures, a transformation can be generated to center and derotate the test image. In order to maximize throughput without image degradation, images are only rotated if the angular error exceeds 0.1º. This also serves to reduce the negative impact of noise in small rotation terms.

The offset is a simple correction, applied first with bilinear interpolation. Once centered, the image is conditionally rotated by Affine transformation and bicubic interpolation.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

The affine transformation is computationally expensive, so neglecting its use when unnecessary for overall image quality is prudent.

F. Summation

Summing the images is trivial once they have been shifted and rotated. They are summed immediately after transformation, as each image in the series is sequentially manipulated.

Results

The program works nicely, and performs the expected operations to generate an output that was compared to a commercial astronomical image processing suite (CCDStack). The images were visibly identical with the test image set I utilized.

The test image set was from my recent observation of NGC3628, a distant galaxy. Each image subframe consists of a 10-minute exposure period, of which ten frames are summed to generate a 100 minute aggregate exposure. The test computer had an E5300 CPU (2.6GHz) running MATLAB and GPUMAT in conjunction with a GeForce 9500GT video card.

With this setup, the computational advantage of the GPU could really only be applied to the FFT coarse centering operation. Below is a table representing the various major chunks of code and the end computational advantage of each section.

Operation	CPU Time	GPU Time	GPU Advantage
FFT Centering	7.819s	4.007s	1.95
COM Fine Registration	0.182s	0.429s	0.42
Resampling	6.14	6.14	1

The FFT operations ran about twice as fast on the GPU than on the CPU. The COM operations ended up taking longer on the GPU despite attempts to improve speed. It is suspected that frequent transfers from CPU to GPU and vice versa occupy most of the extra time.

With a different GPU jacket (such as the one from Accelereyes), other GPU functions such as `conv2( )` become available. This function in particular would allow the acceleration of the re-sampling operations. Unfortunately, I did not have access for testing.

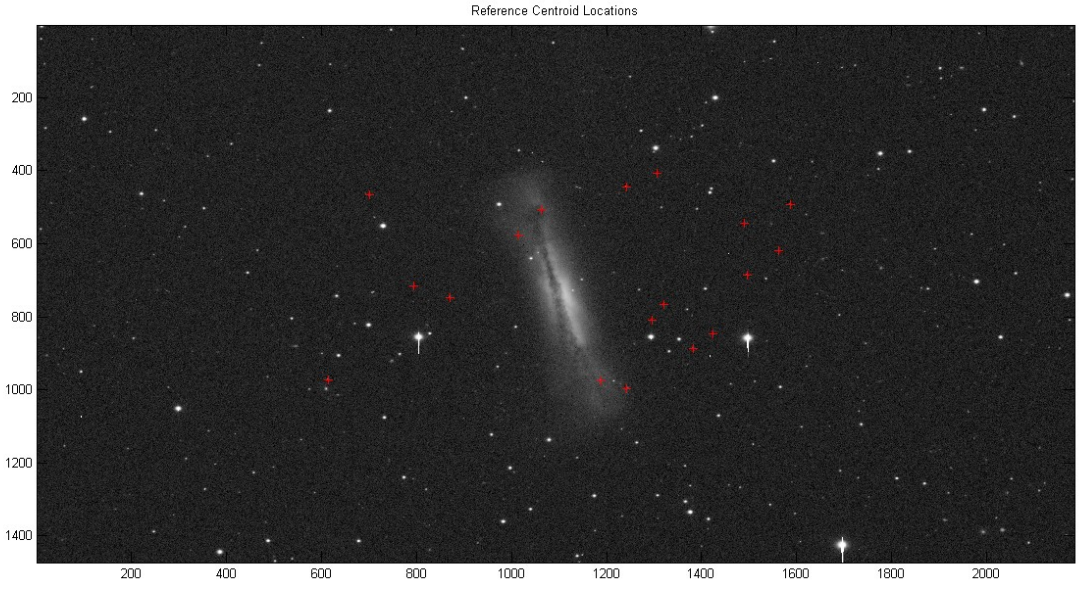

Figure 1: Automatically selected reference stars, overlayed on reference image.

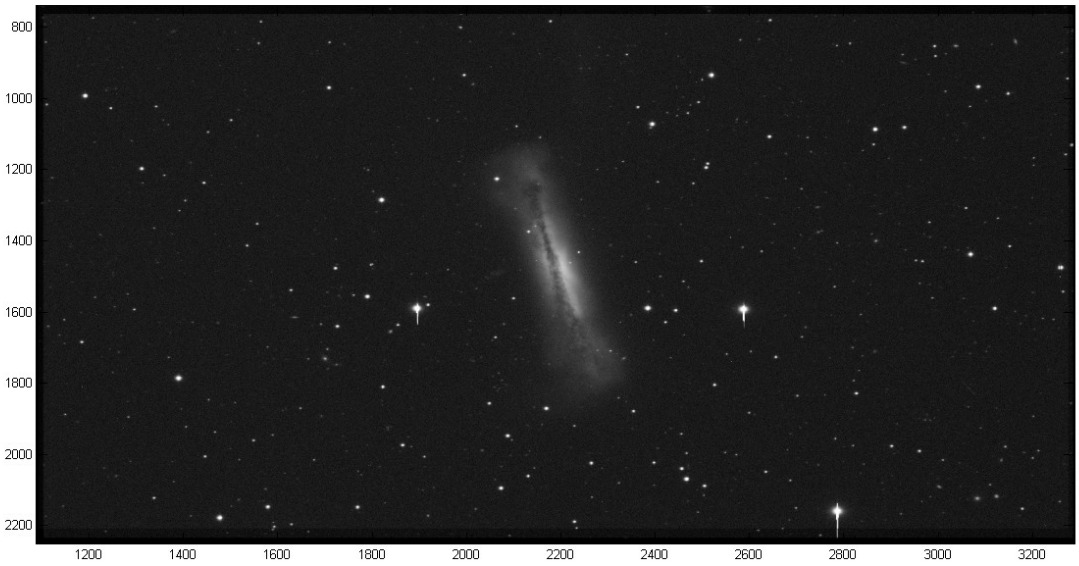

Figure 2: Result of stacking 10 images upon the reference image. Note reduced noise content and improved SNR.

Summary

The program developed for this project will be used frequently by me, as it runs almost twice as fast as the software I have been using in the past. This will be especially meaningful when I must run large datasets.

The GPU offers a powerful solution for large image manipulation, especially when employing complex functions such as FFTs.

I did learn a critical lesson with my FFT registration code. I was using the max( ) commands to find the correlation peak for each image, which ended up being very slow. Using the Nvidea linear algebra command cublasIsaMax( ), I saw a dramatic decrease in computation time.