The Centroid Algorithm

The centroid algorithm is implemented in C, parallelized with threads, and wrapped in a cython wrapper to be called from C.

To identify the fiber spot images, a thresholding algorithm is used. The unfiltered image is searched to find a pixel above the threshold. The region of contiguous pixels above the threshold neighbouring the first pixels is flagged using a standard “flood-fill” algorithm (not considering diagonal pixels), and intensity-weighted first and second moments are calculated on the fly, as well as the peak pixel value. Candidate spots were filtered by the number of pixels included, to filter out noise and hot pixels. In addition, spots too close to the edge of the image are filtered out.

The first moments are then used as initial input for the calculation of an iterative windowed centroid via the following equations (derived from those used in the SeXtractor photometry package). The position is

$$x_w^{t+1} = x_w^t + 2\frac{\Sigma_{r<r_{max}} w_i^t (x - x_w^t)}{\Sigma_{r<r_{max}} w_i^t I_i}$$

where t the iteration number, I the intensity at pixel i, r_{max} is the radius over which the centroid is calculated, and the window function w is

$$w_i^t = exp(-\frac{{r_i^t}^2}{2s_w^2})$$

where r is the distance from the current estimate of the centre (x,y) and s is a constant based on the typical spot size. s was determined empirically from the data, and tested for robustness.

The algorithm requires two external parameters to be set; the threshold (which is given in terms of RMS and calculated from the data), and the kernel size in the x and y directions, used for the calculation of the constant s. The same kernel size is used for all images, to avoid biases in the results.

The algorithm will not return duplicate spots. In unusual lighting circumstances (e.g., the dome lights are turned on and there are bright patches on the image), the algorithm will fail by returning one very large spot in the bright area.

Changes from the previous version

The previous version of the code did not check for contiguity of the pixels in a spot, and used a box size to determine the region in which to search. For compact PSFs this worked well;

however for the more extended PSFs currently seen in MCS data, this could result in a spot being larger than the box, which produced two candidate spots that could produce similar but not identical results from the initial guesses, particularly for oddly shaped PSFs. Increasing the box size also increased the probability of edge cases involving hot pixels, particularly with lower thresholds.

In addition to the changes in the spot detection algorithm, the previous version used a dynamically calculated kernel size, which could lead to small (< 0.02 pixel) variations in calculated centroid. This was changed to a constant kernel size for all images, of (4,9) in (x,y) Note that this differs from the the windowed algorithms used in SeXtractor, which operate on an image-by-image basis.

Tests

The code was checked for memory leaks using the routine leaks. Memory checking, etc. was done via a C routine to call the centroiding code directly (without the python interface); this code is included in the windowedCentroid directory. The code was tested on the real system in simulator mode for a variety of data with different exposure times (0.8, 4.8 and 8 s) to check repeatability, detection efficiency, and to screen for duplicate points at the C level. In particular, pairs of consecutive images with the same exposure time were used to check for differences inconsistent with seeing effects, as this was how we detected issues in the first place. No issues were found.

The code was also tested with sequences of 4.8 s plots with different MCS foci, and varying the input kernel size, to check the robustness of the choice of kernel size.

The updated code has a processing time of 0.3-0.4 seconds for the centroiding, well within specifications, and comparable to the pre-update code.

A Makefile is included in the repository for compiling the C code for testing without running via the cython interface. It can be edited for FITS file input and compiled using make, which produces an executable test_centroid which can be run from the command line.