# INSTRUMENT SCIENCE REPORT

FOC-088

**TITLE:** A Description of the Software used to Derive the Geometric Correction

**AUTHOR:** P. Greenfield **DATE:** 12 September 1995

## ABSTRACT

This report gives a description of the IDL software developed to generate the latest geometric correction models and describes the process of using the specific routines to generate geometric correction models.

---

**DISTRIBUTION:**

SSD: W. Hack, R. Jedrzejewski, A. Nota
SSG: P. Greenfield, P. Hodge. R. White

## 1. Introduction

The Instrument Science reports FOC-086 and FOC-087 describe the derivation of the new geometric correction models for the FOC. These latest models use an entirely new method to determine the geometric correction model as well as using an entirely new representation for the distortion model. These changes necessitated developing new software to derive and implement the new distortion models. This report describes the software used to generate the new distortion models and outlines the process in using the software to generate the models.

Since the previous reports describe in detail the theory behind the new models, the reader is referred to those reports for an explanation of the theory. Only the barest outline will be given here. The distortion models are now represented by the use of two-dimensional spline functions. The coefficients that define the spline functions are determined by fitting these coefficients to the observed data. The data consists of multiple, offset observations of a crowded star field. Since the offsets between the observations are well known a priori and the stars have fixed spatial relationships, it is possible to determine the relative positions of the stars and the distortion model simultaneously from the observations. Once the star positions have been fit, it is possible to use them to determine the distortion model coefficients for other formats. Hence, there are two basic fitting procedures, one that fits both the star positions and distortion coefficients and one that just fits distortion coefficients.

The two-dimensional spline representation uses two-dimensional third order *B*-splines (see ISR FOC-086 and Dierckx, 1993 for the details on the computation of *B*-splines). The *B*-splines are defined by the choice of the knot positions in the x and y coordinates. Implicit in my representation is that there are 3 coincident knots at the boundaries of the format. Each *B*-spline is associated with (and defined by) a set of 4 adjacent knot positions for each direction. Thus the number of coefficients necessary to specify all the *B*-spline amplitude is = (# interior *x* knots+3)×(# interior *y* knots+3) corresponding to the grid of *B*-splines.

The fit star positions are calculated in units of undistorted pixels. Since the size and location of the geometrically corrected pixels is entirely arbitrary, it is necessary to define the characteristics of these pixels to adequately constrain the solution. The size (i.e., plate scale) of the pixels is arbitrary, it has been defined to match the previous definition of undistorted pixels, i.e., 0.01435 arcsec. The way this is manifested in the solution is that the offsets between the exposures are specified in units of the undistorted pixels. Hence the plate scale will be set by specifying the offset in units of 0.01435 arcseconds. The orientation of the star field in the undistorted coordinates is completely arbitrary. It is set by specifying the direction of the exposure offsets in *x* and *y* in undistorted pixels. We have chosen to presume the orientation (position angle) specified previously by the ORIENTAT keyword for the *y*-axis of the undistorted image is what should result from the new distortion correction so that no change to the PDB will be necessary. As a result, one calculates the offsets in pixels in an coordinate system with the y axis with the given position angle. The final free parameters relate to the absolute position of the stars. Since the star positions

are being determined in pixel units on the sky, their absolute position is arbitrary. The first star in the list of stars used for the fit is presumed to have coordinates 0,0. This requires adding the appropriate offset to the undistorted coordinates generated by the spline model to properly place them in the output image. The offset for the full format was chosen to make it align as closely as possible with the previous distortion correction. The offsets for all the other formats were chosen to align them with the full format result after accounting for the relative offsets of the formats.

## 2. The IDL Routines

A number of IDL routines were written for use in generating the distortion models. The most important ones are listed in the Appendix along with a brief explanation and the calling sequence. These routines are not particularly easy to use or user friendly. Several of them have been written with less than the ideal programming style and encompass some poor programming practices such as dependencies on hard-coded constants such as image sizes or zoom factors. There is noticeable ad hoc nature to many of them. The process of generating a distortion model is very tightly bound to IDL; all the intermediate results are kept as variables in IDL, and usually saved to disk using the IDL save facility. It is only at the very end that the results are manifested in a calibration file ready for use by NEWGEOM or CALFOC. Using these routines successfully requires fully understanding the meaning of the intermediate products (variables), keeping proper track of them, and being able to inspect them for validity, frequently using standard IDL facilities.

Ideally, these routines would be rewritten to be easier to use and using better programming practices. Nevertheless, the lack of resources and the relatively low level of FOC use preclude such a rewrite.

None of the widget capabilities of IDL was used in the software, only the simple image windows were used for some of the routines. A couple of the routines presume the availability of a display capable of displaying 1150 pixel wide windows. Some of the fitting routines make use of a external sparse matrix inversion library described and used in ISR FOC-060 and FOC-061. The files generated for this implemented this external packed using the linkimage facility under VMS IDL. Since IDL now supplies a sparse matrix routine, it would be most sensible to make the minor changes to the fitting routine to use the native IDL routine for use on other platforms.

## 3. The Model Fitting Process

The following sections will outline the process of actually using the software to generate models. Descriptions of the routines referred to are given in the Appendix.

### 3.1. The General Fit

This describes the process of fitting both the star positions and the distortion model coefficients. As mentioned in FOC-086, the star positions in each exposure are determined by using DAOFIND. DAOFIND generates a text file containing a header followed by columns containing a star number, x, y position, relative magnitude, sharpness, and roundness. The last two are mea-

sures of how well the image size and shape correspond to that expected. I allowed very broad ranges for both in generating the star list. A full format exposure usually generates between a 1000 and 1200 stars using DAOFIND. These star positions are read into IDL using the astro procedure `readcol`. For example, for the northeast image of a set

```
readcol,'find_ne.txt',format='(f,f,f,f,f)',xne,yne,mne,sne,rne
```

to read the *x*, *y* positions, magnitudes, sharpness, and roundness in to the respective variables `xne, yne, mne, sne, rne`. Generally speaking, we will have no use for the sharpness and roundness values. Most of the routines deal with the position as a 2-dimensional array and so the next step is to combine *x* and *y* into one variable

```
pos_ne = [[xne],[yne]]
```

This is done for each of the images in the set of overlapping exposures. Before going any further it is necessary to compute the offsets of the exposure to the reference image (for which I always took the one with offset 0,0). As mentioned, offsets must be computed in pixels in the coordinate system having its *y*-axis with a position angle equal to the value specified by the header keyword `ORIENTAT` (since the value may vary slightly between the different image, adopt the one from the reference image). For images with zoomed pixels it is possible to determine the distortion model either using the dezoomed or zoomed image. I always chose to use the zoomed image. In this case one must pay close attention to the coordinate values for *x* here and later in order to get everything right. This point cannot be emphasized enough! In this case the *x* offsets must be in zoomed pixels.

Before continuing it is necessary to construct byte scaled images for each exposure so that IDL can rapidly display the images for the next step. The byte-scaled image should be constructed so that it is relatively easy to judge the center and quality of faint and bright images. This is probably best done by constructing a log image and byte scaling it appropriately, e.g.,

```
lim = bytscl(alog(im+1) < cutvalue)
```

where cutvalue give the most appropriate display (check by using `tv, lim`).

The next task is a somewhat tedious one and must be planned carefully. In many respects it is the most confusing and takes the most work than all of the rest of the steps combined. It is necessary to make correspondences between identical stars in different exposures manually so that the software can identify all multiply observed stars. I chose to match a fairly high number of stars for full format exposures for each pair of exposures, typically between 50 and 100 depending on the amount of overlap. I attempted to mark all stars that formed the convex hull of the stars common to a pair of exposures as well as a moderately uniform distribution of stars in the interior. There was also a tendency to match more stars towards the format edges where distortion was presumably higher, and especially near the scan line beginning. These manually identified pairs of stars are used later by a routine that triangulates the overlapping areas based on the manually matched

stars to determine a piecewise linear transformation function to match the rest of the stars. The routine that aids in the manual matching of stars is `pairstars`. It displays two image windows, one above the other, that should show zoomed sections of the two images that correspond to each other. Using the mouse, one can mark a star in the top window and then mark the corresponding star in the bottom window. The position of all the DAOFIND stars is marked with crosses in the top image. Marking any one of those (by clicking the left button) results in the next number in the sequence displayed for that star. When the corresponding star is marked in the bottom image a cross is plotted at the clicked position (it does not check to see if there is a DAOFIND star there or nearby) labeled with the same number. Since only a small section of large formats can be shown in the window, it requires many offsets of the images to cover the overlapped region. These offsets can be accomplished by using a crude drag feature. This is accomplished by clicking the middle button in the top window. The next right button click marks a point in the field, the second indicates where you want that point to end up. After the second click, both images are shifted in their respective windows by the same amount. The window where the mouse is active (i.e., the one the program expects a mouse click) is highlighted with a white border. The right button is used to quit the program.

The perceptive reader will wonder how the program knows to display the images with the proper offset relative to each other. This is done by specifying an offset in calling the program that results in the appropriate matching regions being displayed. As a start one can base the offset on the relative offsets computed for each image. But because of distortion, it is usually possible to tweak the predicted offset to get better matching displays. This is simply done by executing the program with refined values until you are satisfied with the alignment of the two views (the brute force method). The program starts by placing the top image with its lower leftmost pixel at the lower leftmost point in the window. The bottom image is displayed with its lower leftmost pixel at the specified offset, that is, a –100,–100 offset results in placing pixel 100,100 of the image at the lower left. The way that offsets are handled require certain contortions in the way the data are reduced, which will be described shortly.

The program is invoked by typing

```
pairstars, im1, im2, pos1, offsets, pospair
```

where `im1` (top) and `im2` (bottom) are byte scaled versions of the exposures described previously, `pos1` is the position array for `im1`, offsets is a 2-element array specifying the *x* and *y* offset of `im2` relative to `im1` (e.g., [–100,–230]), pospair is a three dimensional array specifying corresponding pairs of positions which is returned by the program (dimension = #pairs×2×2; first index: pair ID; second index:0=x coord, 1=y coord;.0=im1 coord,1=im2 coord).

Because pairstars initially displays the top image with no offset, it is necessary to order pairs of image such that the second image is always below (relative to the pixel offset coordinate system) the first. For a similar reason (although not as important since the windows are much wider then

they are high) it is preferable to have the second image always to the right of the first image. These requirements coupled with the requirement of the next program, which deals with all pairs of images forces a certain ordering of images and pairs. The next step is a synthesis of all unique, multiply observed stars as determined from the manually matched pairs of stars and the DAOFIND list of stars. The program which carries out this synthesis systematically goes through all possible image combinations in a certain order which is based on a order of images given to it. This order of images is used to construct a 2 dimensional matrix of all image pairs e.g.,

| | **im1** | **im2** | **im3** | **...** | **imN** |
|---|---|---|---|---|---|
| im1 | | im1_im2 | im1_im3 | | im1_imN |
| im2 | | | im2_im3 | | im2_imN |
| im3 | | | | | im3_imN |
| | | | | | |
| imN | | | | | |

**Table 1.**

The resulting order of pairs implied by this construction becomes im1_im2, im1_im3, ...im1_imN, im2_im3,..im2_imN, and so on. The order of the images for each pair must correspond to the order that stars for each pair were matched. That is, if the list of images presented to the program is im1, im2, ..., imN, then the order of pairs given to the program must be im1_im2, im1_im3, and so on (as above) and the list of star pairs obtained from pairstars must match the order of the images listed for each pair; in particular for pair im1_im2, im1 is the first image, and im2 is the second, for pair im3_imN, im3 is the first image and imN is the second. So that these pairs are constructed in the right order, it is necessary to order the images from highest to lowest (in the pixel coordinate system) and within that ordering (given images at effectively the same relative y position) order the images from right to left. To take as an example the last set of observations, there were observations at the center position (which I'll denote by `im_cc`) and the 4 positions equally spaced from then center position to the northeast, northwest, southeast, and southwest (denoted by `im_ne`, `im_nw`, `im_se`, `im_sw`). Since these images by chance happened to be oriented almost exactly with north, north corresponds to the "highest" images and east with the rightmost. The order implied by the above requirements would be for the images: `im_ne, im_nw, im_cc, im_se, im_sw`. The corresponding pairs are then `ne_nw, ne_cc, ne_se, nw_cc, nw_sw, cc_se, cc_sw, se_sw`. Thus for the pair `ne_nw` `im_ne`, `im_nw` would be the first and second images for `pairstars`, respectively.

Since not all pairs of stars yield useful (or any) overlaps, it is necessary to construct a matrix identifying which pairs of images have lists of matched stars. For the example just used, the matrix would be

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (1)$$

where elements with value 0 indicate where no list of paired stars exists and elements with value 1 indicates a paired list does exist. In IDL the matrix would be constructed using the statement

```
pairmask = [[0,1,1,1,0],[0,0,1,0,1],[0,0,0,1,1],[0,0,0,0,1],[0,0,0,0,0]]
```

This variable will be needed for the next step.

The program pairstars is run all the useful overlaps using the order indicated by the above prescription. Each use results in a list of pairs; if we continue with the above example we would have the pair variables: `pospair_ne_nw, pospair_ne_cc,` and so on. The routine that synthesizes the list of multiply observed stars is called `multistar`. Since it must deal with a collection of position lists and pair lists of variable number and size, some flexible form of packaging these variables is needed. It probably would have been best do arrange them as an array of structures, each of which contained the size information for each. I took a different approach by packaging each array into one array, which contains imbedded information about the dimensions and size of the arrays contained within. There are two routines developed to package and unpackage these arrays called `group` and `ungroup`. For the given example, the input arrays would be grouped by typing

```
pos_group = group(pos_ne, pos_nw, pos_cc, pos_se, pos_sw)
```

```
pospair_group = group(pospair_ne_nw, pospair_ne_cc, ... , pospair_se_sw)
```

Note that the order of arguments is important and must correspond the order determined above.

Then multistar would be run as follows

```
multistar, pos_group, pospair_group, offsets, pos, q, o
```

The input variable offsets contain the offsets of each of the input images relative to the reference image (`im_cc` in this case) which is not necessarily the same as the first image (a 2-d array, dimensions `nstars`, 2). The program must do a great number of comparisons and transformations so it may take 5 to 15 minutes to finish depending on the number of images and stars. The program returns three arrays; `pos` contains the position coordinate pairs for each star that have been determined to be multiply observed as measured in a given image (a given star will appear at least twice in this list) and has dimensions `n_meas_star_pos,2`; `q` is a 1-d array of length `n_meas_star_pos`, it contains the star ID corresponding to each element of pos—one can find other observations of the same star by searching `q` for other occurrences of the same star ID (note these "star IDs have no relation to those used for the original list of star positions for the individual images); o is a 2-d array of dimensions `n_meas_star_pos,2` that contains the image off-

set corresponding to the star position given in the corresponding array `pos`. These three arrays contain all the information needed to carry out the fit (aside from the definition of the spline knots). The procedure `multistar` (or `multiuvstar`, described later) use a procedure called `pmatch`. This procedure should be edited before running `multistar` to match the type of pixel being used (zoomed or normal). The line that must be changed is noted in the file.

The last step before fitting is the definition of the spline knots. A structure is used to contain the definition of both the *x* and *y* knots for the spline fit as well as the fitted coefficients. This structure can be generated using the routine `pickpanel`. This routine can be used to generate a spline structure from scratch or to use an existing one to generate a new one. It is invoked as

```
splinemodel=pickpanel(bytim) ; to make from scratch

splinemodel=pickpanel(bytim, old_splinemodel) ; to modify a pre-existing model
```

The `bytim` is an image of the same size as those used for the model. You may use the star field byte scaled image or a byte scaled flat field to determine where the knots may be placed (the star field is useful as an indication of star density relative to panel size whereas a flat field provides clues about where distortion might be higher, i.e., scanning effects). Whatever you use, it must be byte scaled. The routine allows interactive placement of *x* and *y* knots as well as deleting knots by means of mouse clicks and keystrokes (it use should be fairly self explanatory).

Once the spline structure is defined, a model fit can be attempted. The program is `twodfit` and would be run as follows

```
@lsparse
twodfit, pos, o, q, splinemodel, uvpos, resid
```

The returned array `uvpos` contains a 2-d array of the fit position of each of the stars. The index of a position in the list corresponds to the star ID that is referred to in the input array `q`. The first postion has coordinates 0,0 by definition. It is expected that many stars will have negative coordinates for this reason. As mention, only the relative positions have any meaning in this result. The array `resid` contains the *x* and *y* residuals corresponding to the measured positions appearing in pos, i.e., it is the difference between the model predicted positions and the measured positions. The residual number that the program prints out is simply the rms of this array. The spline model coefficients fit are placed into the structure `splinemodel`. It is this variable that contains the end result of the fit. The variables `uvpos` and `resid` are of great value in analyzing the quality of the fit, however. And it is `uvpos` which may be used as input to determine the splinemodel for

other formats. The `@lsparse` is necessary before the first use of `twodfit` (or `twodfitw`, described later) in order to link the sparse matrix routines on a VMS machine.

### 3.2. Weights

Weighting can be incorporated into the fitting process. Presuming an array `w` corresponding to each measured position in the original position lists (e.g., `w_cc, w_ne,` and so on) that contained the weighting factor for each measured position. Then the variable `w_group` would be constructed in the same way `pos_group` was. That is

```
w_group = group(w_ne, w_nw, w_cc, w_se, w_se)
```

Then `multistar` would be called with two additional arguments

```
multistar, pos_group, pospair_group, offsets, pos, q, o,w, w_group
```

where `w` is the set of weights corresponding to `pos`. To fit, `twodfitw` is used instead

```
twodfitw, pos, o, q, w, splinemodel, uvpos, resid
```

The difficult part is determining the weights. Currently they are based on the dependence measured between total counts and position errors using Monte Carlo techniques (see ISR FOC-086). First it is necessary to transform the relative magnitudes measured by DAOFIND into the equivalent of total counts. This can be crudely done by using `magcal` to determine an average ratio of DAOFIND flux to total counts. For each image (the central one is used as an example here)

```
magcal, im_cc, x_cc, y_cc, m_cc, ratio_cc
```

the array `ratio_cc` contains the ratio of DAOFIND flux (e.g., $10^{-mag/2.5}$) to total counts for each star. A plot of the histogram will indicate the rough scaling factor to convert the flux to total counts. Then

```
tcounts_cc = scale_factor * 10^(-mag/2.5)

w_cc = calcweights(tcounts_cc)
```

The routine `calcweights` is valid only for normal pixel images. It is unlikely that weighting would be useful for zoomed mode images.

### 3.3. Coefficient-only fits

When the star positions have been fit using the above techniques for one format, it is possible to use the fit positions for the stars to fit the spline model coefficients for other format. Presuming that there is a suitable array `uvpos` obtained from another full fit and that `uvpos` covers a wide enough field for the format being fit, then it is fairly straightforward to fit the coefficients. If a weighted fit is desired then the weights are determined in exactly the same way as previously. If the observations that will be fit were taken at the same telescope roll angle as those used to derive `uvpos`, then no rotation of the `uvpos` coordinates will be necessary. On the other hand if they

were not obtained at the same roll angle, then a rotation must be applied to the uvpos positions to bring the star field into the same orientation as the observations to be fit. The rotation angle will be the difference in the telescope roll angles for the two sets of observations. The function uvtran can be used to apply a rotation as well as offsets.

For the purposes of the next step, it is necessary to apply an offset to the rotated uvpos positions to roughly align them with the image coordinates of the same stars. This is most simply done by identifying the same star in the uvpos list and one of the images that will be used for the fit. This can be accomplished by using one of the original images for the first fit and matching a star with the new format image, and then tracking the star through the pos_cc, pos, and uvsol arrays. Once a correspondence is identified, determining the offset for each image (which will be referred to as uvoffsets in a following routine) is trivial. The analogous procedure to pairstars for this situation is pairuv.

```
pairuv, im, uvpos, uvmag, pospair
```

where im is a byte-scaled image suitable for display, uvpos is a list of *u*,*v* star positions, uvmag is the corresponding relative magnitude of the star images, and pospair is the returned list of paired stars. If there are more than one image where pairuv must be run, it is most convenient to call pairuv using the same uvpos with uvoffsets added (i.e.,

```
[[uvpos(*,0)+uvoffsets(i,0)],[uvpos(*,1)+uvoffsets(i,1)]]
```

A constant can be added to the magnitudes to give a more usable display (the program assumes the range of relative magnitudes is similar to those returned by DAOFIND, i.e. between 0 and –3; magnitudes outside that range result in no difference in appearance from those magnitude values).

Ideally one would like to use the suitably transformed uvsol obtained as the fit *u*,*v* positions as the input uvpos for pairuv. The problem is that it is not a simple matter to match magnitudes to the fit star positions. Generally speaking, each fit position corresponds to two or more star images, each with a different measured magnitude. The bookkeeping involved in tracking down the source images and magnitudes and appropriately combining or averaging them is sufficiently messy to avoid if possible. A set of routines was written to do this (as well as include all stars only once observed into the list) but they are not well enough developed and their use is not recommended. An easier, and almost always usable, method is to simply use the DAOFIND positions for one of the images—presumably the central one—and apply the geometric correction model to those positions to obtain the corresponding *u*,*v* positions. This is done by using the routine spline_ev_2d (found in twodfit.pro or twodfitw.pro )to evaluate the spline model as follows:

```
uvpos = spline_ev_2d(splinemodel, pos)
```

Remember to apply the necessary coordinate transformation to this uvpos to align it with the image. The advantage of this approach is that the magnitudes from DAOFIND can be input

directly into pairuv. This approach will not work if one is trying to use *u*,*v* positions obtained for the same format but different roll angle since there will be areas of the image not covered by the previous central image. In that case some means of associating magnitudes with the uvsol will be necessary or trying to use pairuv without magnitude information (which is tricky without magnitude cues).

Since there may be more than one image to provide a higher density of points, it is necessary to group the pos and pospair arrays (even if there is only one) to use the corresponding matching program which is called as follows

```
multiuvstar, pos_group, pair_group, uvpos, uvsol, uvoffsets, offsets,$
                        posset, q, o, w, w_group
```

where pos_group, pair_group, offsets, q, o, w, and w_group have the same meanings as for multistar. The array posset contains the pairs of image positions and undistorted positions for each observed star image generated by multiuvstar. The remaining three—uvpos, uvsol, and uvoffsets—can easily confuse. The variable uvpos corresponds to the positions used to pair the DAOFIND positions, i.e., those used for pairuv. The variable uvsol corresponds to those *u*,*v* positions obtained from the previous fit (appropriately transformed). Both are used so that the visual pairing can be done with one (because of the easier association with magnitudes) and the fit positions (uvsol) can then be tied to each of the measured positions after star images are matched. The variable uvoffsets contains the offsets needed to be added to uvpos that were used to register the uvpos positions with each of the input images, i.e., uvoffsets(image number, 0=x coord; 1=y coord).

The model fit is performed by calling either uvfit or uvfitw for the unweighted or weighted version respectively.

```
uvfit, posset, splinemodel, resid
uvfitw, posset, splinemodel, w, resid
```

where splinemodel's knots are defined in the same way as for the full fit.

The 128×128 format used a polynomial fit instead because of the relatively small number of data points. The routine uvpolyfit performs the $2_{nd}$ order 2-d polynomial fit (weighted). It is called as follows

```
uvpolyfit, posset, w, ucoeff, vcoeff
```

The model is evaluated using the function in the same file called polytwod. It would be used as follows to obtain the undistorted coordinates from the distorted coordinates.

```
u = polytwod(ucoeff,posset)
v = polytwod(vcoeff,posset)
```

Here u and v are simply one-dimensional arrays.

## 4. Generation of Calibration Files

At present, constructing the geometric correction file is a highly manual process (though nothing prevents a more automatic method in principle). Before geometric calibration files can be constructed the *u*,*v* solutions for all the formats need to be properly registered. I have done this by choosing the large format as the master format and determining the appropriate offsets to the *u*,*v* solution that best register the geometrically corrected location of the reseaux with those assumed for the previous reseaux-based geometric correction. This can be done using the iraf tasks in the foc package to do so or the `getres` and `fndres` procedures in the `fndres.pro` file. The procedure `getres` simply allows the user to mark the approximate position of the reseaux using the display, the resulting positions are passed along to `fndres` which obtains better positions for the reseau given the initial positions. These *x* and *y* positions are then transformed to undistorted *u*,*v* coordinates using `spline_ev_2d` and dezoomed if necessary (for the full format, for example). The reference positions for the reseaux must be obtained from the appropriate STSDAS table and entered by some means into array variables for the *x* and *y* positions (one can use the `rprintx` task in `foc.focgeom` to print the coordinates into a file and then use the UIT astron IDL library task readcol to read the *x*,*y* coordinates into arrays). One must now associate the corresponding reseaux between the two sets of coordinates and reorder one or both of the lists (as well as select only those reseau that appear in both) to construct corresponding lists. The easiest way to do this is to chose one of the central reseau common to both input lists and determine the approximate offset to apply to the *u*,*v* coordinates to match coordinates. Apply this offset to all of the *u*,*v* coordinates of the geometrically corrected reseau. Then use the pmatch procedure to determine which pairs of reseau coordinates match and construct the corresponding reseau lists from the result. The adjustment of the offset is obtained by computing the average difference in the *u* and *v* differences.

The offset for other formats is determined by identifying a common star and then computing the undistorted *u*,*v* position for the star in both formats. For the star to properly register in geometrically corrected images, the relative offset of the new format must equal the difference in *u*,*v* carinated adjusted for differences in the format offsets. For example, suppose the difference in the *u*,*v* coordinates of a star in the 512×512 and 256×256 formats is (100,200). The relative offset is (–100–256+384,–200–256+384) = (28,–72) must be added to the 512×512 coordinates to match the absolute coordinates implied by the 256×256 position. The relative offset determined for each format is combined with the absolute offset determined for the master format to obtain the net offset for the format.

The geometric correction file is constructed beginning with the procedure `eval2d` which evaluates the spline model on a grid

```
eval2d,splinemodel,x,y,u,v
```

x and y are 1-d arrays which define the *x* and *y* coordinates corresponding to the two 2-d output arrays u and v. In particular what is generated are two arrays u and v which represent the undistorted *u*,*v* coordinates corresponding to the input coordinates represented by x and y. For example, u(i,j) is the undistorted *u* coordinate corresponding to distorted image coordinate x(i),y(i) and likewise for v. The geometric correction task actually requires just the opposite mapping. What is needed is two 2-d arrays xd, yd whose values represent the corresponding distorted coordinates. That is, xd and yd specify the mapping from a location in the undistorted image to a location in the distorted image. The value xd(i,j) is the *x* value of the distorted image that corresponds to the position *i*,*j* in the undistorted image (and likewise for yd). The procedure geoinvert constructs the inverse mapping from the u,v arrays.

```
geoinvert,u,v,xd,yd,mask
```

where mask is used to indicate where values were extrapolated. The geometric correction program expects a single array which is a concatenation of the xd and yd arrays. Furthermore, for a format nx × ny in size it expect to see a 2*(nx+1) × (ny+1) array since it evaluates positions at the corners of the pixels.

The following illustrates the generation of the master format correction file

```
; run rprintx in iraf to generate reference reseau file
readcol,'optref.txt',opt_u,opt_v ; read into IDL
getres,meas_init_x,meas_init_y ; get inital reseau positions
fndres,im,meas_init_x,meas_init_y,meas_x,meas_y
; apply distortion correction
gc_res = spline_ev_2d( splinemodel, [[meas_x],[meas_y]])
gc_meas_u = 2*gc_res(*,0) ; dezoom
gc_meas_v = gc_res(*,1)
; determine approximate offset manually
; apply approximate offset
tmeas_u = gc_meas_u + approx_offset_u
tmeas_v = gc_meas_v + approx_offset_v
; find reseau pairs
pmatch,[[tmeas_u],[tmeas_v]],[[opt_u],[opt_v]],4,pair
; construct matched reseau set
ptmeas_u = tmeas_u(pair(*,0))
ptmeas_v = tmeas_v(pair(*,0))
popt_u = opt_u(pair(*,1))
popt_v = opt_v(pair(*,1))
offset_u = approx_offset_u - total(ptmeas_u-popt_u)/n_elements(popt_u)
offset_v = approx_offset_v - total(ptmeas_v-popt_v)/n_elements(popt_v)
;
; now begin making geometric correction file
;
eval2d, findgen(1025)/2, findgen(1025), u, v
; dezoom u values and apply absolute offset
u = 2*u
```

```
u = u + offset_u
v = v + offset_v
geoinvert,u,v,xd,yd
; form combined array for geo correction file
gc_im = fltarr(2050,1025)
gc_im(0,0) = xd
gc_im(1025,0) = yd
save, file='geocorr.sav',gc_im ; save in a save file
; read in a template header and open new file for writing
sxopen,1,'template.hhh',hdr
sxaddpar,hdr,'NAXIS1',2050
sxaddpar,hdr,'NAXIS2',1025
sxopen,1,'geocorr.hhh',hdr,'','W'
sxwrite,1,gc_im
close,1
;
; Done!
```

In reality, the process shown above would include inspection of the products after major steps to make sure that all results were sensible as well as saving intermediate results to disk as IDL save files.

The following illustrates the generation of a geometric correction file for a format which must be registered with the master format.

```
;
; determine the relative offset
rel_offset_u = 2*star_pos_u_full - star_pos_u_256 - 384
rel_offset_v =   star_pos_v_full - star_pos_v_256 - 384
eval2d, findgen(257), findgen(257), u, v
; apply net (relative + absolute for master format) offset
u = u + offset_u + rel_offset_u
v = v + offset_v + rel_offset_v
geoinvert,u,v,xd,yd
; form combined array for geo correction file
gc_im = fltarr(514,257)
gc_im(0,0) = xd
gc_im(257,0) = yd
save, file='geocorr256.sav',gc_im ; save in a save file
; read in a template header and open new file for writing
sxopen,1,'template.hhh',hdr
sxaddpar,hdr,'NAXIS1',514
sxaddpar,hdr,'NAXIS2',257
sxopen,1,'geocorr256.hhh',hdr,'','W'
```

One can apply the geometric correction to the starfield image taken in both formats and blink the pair (with the appropriate format offset) to determine if there is any residual offset that needs to be

applied. If so, the above process can be repeated with the adjusted offset. Any such offset should be much less than a pixel (offsets as small as 0.2 are noticeable when blinking different formats).

## 5. Utility Routines

There are a number of utility routines, mainly for displaying or plotting the results. They are described in the Appendix.

## 6. References


1. Dierckx, P., *Curve and Surface Fitting with Splines*, Clarendon Press, 1993.

2. Greenfield, P., Instrument Science Report FOC-060, *FOC UV Flat Field Response*, May 1992.

3. Greenfield, P., Instrument Science Report FOC-061, *The Algorithms and Software Used to Process the Orion Flat Field Data*, May 1992.

4. Greenfield, P., Instrument Science Report FOC-086, *Deriving the Geometric Correction from Crowded Fields*, February 1995.

5. Greenfield, P., Instrument Science Report FOC-087, *The New f/96 Geometric Correction Models*, September 1995.


## Appendix

The following is a list of the relevant functions and procedures, the files they are contained in, and a brief description of their parameter list. These programs can be found on FOCA in the directory SYS$LOCAL:[geo_distortion]. For most the filename is the same as the program name (i.e., `multistar` is in `multistar.pro`) and as long as the procedure (or function) has the same name as the filename, it is loaded automatically when you invoke it. Those whose names are different from the filename may need to be "`.run`" in order to use them. Procedures and functions can be distinguished in the following by whether or not the argument list is enclosed in parentheses.

**Setup and Fitting routines:**

---

**`calcweights(tot_counts) [in magcal.pro]`**

```
Calculates the appropriate weights to be used for fitting purposes given the
total counts measured or derived for each star. This routine is only valid for
normal pixel image fits.
```

```
input:
      tot_counts     array of total counts for each star.
output:
      return value   array of weights for each star.
```

---

**eval2d, splinemodel, x, y, u, v**

Evaluates the both the u and v spline models on a grid defined by the x and y input arrays. The x and y input arrays are linear arrays which define the input grid coordinates and the size of the output array. For example, the ouput undistorted x coordinats (u) is a n_elements(x) x n_elements(y) array whose undistorted x position, u(i,j) corresponds to the position x(i),y(j) in the distorted image.

```
input:
      x              linear array specifying input distorted x positions.
      y              linear array specifying input distorted y positions.
output:
      u              the output undistorted x postions corresponding to the
                     input distorted position.
      v              the ouput undistorted y position corresponding to the
                     input distorted position.
```

---

**geoinvert, u, v, xd, yd, mask**

This procedure takes two input 2d arrays (u,v) whcich represent the mapping from distorted image coordinates to undistorted image coordinates and produces the inverse mapping as xd and yd. The same sized 2d array mask shows where the inversion had to be extrapolated. This procedure can take quite a while for full format images and has the potential for consuming all of virtual memory.

```
input:
      u              2d array of undistorted x positions corresponding to the
                     position in the array representing the distorted image
                     coordinates.
      v              likewise for undistorted y positions.
output:
      xd             2d array of distorted x positions corresponding to the
                     position in the array representing the undistorted image
                     coordinates.
      yd             likewise for distorted y positions.
      mask           2d array with nonzero values showing extrapolated values
                     for xd and yd.
```

---

**group(v1, v2, ... , vn)**

This function takes all the input arrays (which must be of the same type and same number of dimensions) and creates a new, single 2-d array where the 2nd dimension size is the number of input arrays and the 1st dimension size is tied to the maximum size of the input arrays. The output array has the same type as the input arrays. Use ungroup to extract the individual input arrays.

---

**magcal, im, x, y, mag, ratio**

Calculates the ratio of counts in a 5x5 box around each star (defined in the x,y list) to the flux measured by DAOFIND (input as a magnitude). The ratio is used to determine an effective zero point for the DAOFIND magnitudes.

input:

| | |
|---|---|
| im | raw image used for DAOFIND. |
| x | array of x coordinates for the stars. |
| y | array of y coordinates for the stars. |
| mag | magnitudes as measured by DAOFIND. |

output:

| | |
|---|---|
| ratio | array of ratios. |

---

**multistar, pos_group, pair_group, pairmask, offsets, pos, q, o, w, w_group**

This procedure takes all the DAOFIND derived positions for each of the images and the manually identified pairs of stars for each possible pair of images, and does the work to generate the sets of data for use with the fitting program. In particular, the following sequence is carried out:

1) identify the stars from DAOFIND that correspond to the "matched positions".
2) eliminate pairs that don't yield ID's for both images.
3) transform the one set of positions for the first image in the pair to the expected coordinates in the second image of the pair using the matched pairs to carry out a piecewise linear transform.
4) use the transformed positions to identify the same stars in the second image of the pair.
5) eliminate all of those pairs where the transformed position isn't matched to a star in the DAOFIND list for the second image.
6) construct a table of ID stars and images.
7) determine all stars that are observed in more than 1 image and concatenate into one list of positions with the corresponding image offset and a "q" array which identifies which star that position belongs to.

input:

| | |
|---|---|
| pos_group | the "grouped" set of DAOFIND position lists (see the documentation on the group function). See the report text for an explanation of the expected order of the |

position lists.

| | |
|---|---|
| pair_group | the "grouped" set of manually paired star images obtained from pairstars. See the report text for an explanation of the expected order of the pair lists. |
| pairmask | a 2d array (nimage x nimage). nonzero entries indicate those pairs of images for which a set of manually paired list of stars has been obtained (and grouped into pair_group) |
| offsets | a nimage x 2 array containing the offset of the respective image from the reference image (in general, the reference image-0,0 offset-is not the first). |
| w_group (opt.) | the "grouped" set of weights appropriate for the corresponding position arrays contained in pos_group. |

output:

| | |
|---|---|
| pos | an array containing the list of all star positions (from any image) for which the star has been observed in two or more images. (dimensions: nstar x 2) |
| q | an array corresponding to pos which gives a star "ID" for each element in the pos array. Used to identify which positions in pos correspond to the same star. (dimensions: nstar) |
| o | an array corresponding to pos which gives the image offset for the image the position was measured in (dimensions: nstar x 2). |
| w (optional) | an array corresponding to pos which give the appropriate weighting factor for the pos position (dimension: nstar) |

---

**multiuvstar, pos_group, pair_group, uvpos, uvsol, uvoffsets, offsets, $**
**posset, q, o, w, w_group**

This procedure has a similar purpose to multistar except instead of pairing stars between different image, it is intended to pair stars between images and a previous u,v solution.

input:

| | |
|---|---|
| pos_group | same as with multistar |
| pair_group | same as with multistar except pairs are between images and a u,v list of star positions. |
| uvpos | a list of undistorted positions used in matching with star positions in images (used to generate pair_group). |
| uvsol | a list of undistorted star positions generated by a previous fit. Not necessarily the same as uvpos (see report text). |
| uvoffsets | the offsets that needed to be added to uvpos for use with pairstar for each image. (dimension: nimage x 2) |
| offsets | same as with multistar. |
| w_group | same as with multistar (except required this time) |

output:

| | |
|---|---|
| posset | a list of all image position, undistorted position pairs |

generated by multiuvstar (dimension: nstar x 2 x 2).

q — same as with multistar.

o — same as with multistar.

w — same as with mulitstar.

---

**pairstars, im1, im2, pos1, offset, pospair**

This procedure is used to interactively pair a subset of stars between a pair of images for use with multistar. The program repeatedly cycles between two windows, each of which displays a region of on of the input images. The active window (the one expecting a mouse click) is highlighted by a white border (presuming white is the color of the maximum value in your color table!). The first window show all of the DAOFIND stars marked with a cross. The user marks a star in the first window (left mouse click) and marks the corresponding star in the second window (to within a few pixels, the software attempts to match the appropriate star). The displayed region can be moved relative to the images by doing a "crude" drag. This is accomplished by a middle mouse click in the top window followed by two left mouse clicks in the top window indicating that the first click position is to be moved to the second click position. A middle right mouse click stops and returns from the procedure.

input:

- im1 — a byte scaled image corresponding to the first image.
- im2 — a byte scaled image corresponding to the second image.
- pos1 — the DAOFIND list of positions corresponding to the first image.
- offset — the relative offset of the second image relative to the first in pixels (a 2 element array, [x,y]) so that the displayed regions will be approximately the same region of sky.

output:

- pospair — an array containing the marked pairs of positions (dimension: nstar x 2 x 2).

---

**pairuv, im, uvpos, uvmag, pospair**

This procedure is similar to pairstar except only one image is used and the star images are paired with u,v positions contained in uvpos.

input:

- im — a byte scale image
- uvpos — a list of u,v positions of stars in the same field as determined from another format.
- uvmag — the corresponding list of relative magnitudes for stars in uvpos (the faintest magnitudes should be around 0).

output:

- pospair — an array containing the the pairs consisting of a marked star image and a marked u,v position (dimension:

nstar x 2 x 2).

---

**pickpanel(byte_image, splinemodel)**

This function constructs a spline model structure corresponding to the selected knot positions. A window the same size as the input image is displayed. The user is then prompted to select x or y knot positions, which are seleted by marking the position with a left mouse click. Keystrokes select x or y mode and a delete knot mode as well. The function is terminated with a right mouse click. If a splinemodel is included in the argument list, it is used as the starting basis for creating a new splinemodel.

input:

| | |
|---|---|
| byte_image | a byte scaled image useful for selecting knot positions. It must be of the same size as the format being fit. |
| splinemodel | a spline model structure containing knot positions to be used a starting point. (optional parameter) |

output:

| | |
|---|---|
| return value | a spline model containing the knot positions but 0 values for the spline coefficients. |

---

**pmatch, pos1, pos2, rthresh, pair**

This procedure is used to find which items in the two input position arrays correspond to each other within the specified radius tolerance (rthresh). The result is returned as a array of pairs of indicies into the two position arrays identifying matching positions. NOTE THAT THE SOURCE FILE NEEDS A MODIFICATION TO HANDLE ZOOMED OR NORMAL PIXEL DISTANCE EVALUATION (indicated by a comment in the source file). Even though this routine is not used by the user directly it is used by the multi*star routine and needs to be modified to match the appriate pixel format (yes I know, lousy design!)

input:

| | |
|---|---|
| pos1 | an array of x,y pixel positions. |
| pos2 | an array of x,y pixel positions. |
| rthresh | the maximum distance allowed between two positions in pixels for them to be considered a match. |

output:

| | |
|---|---|
| pair | an array of pairs of indicies for pos1 and pos2 indicating matched positions. |

---

**polytwod(coeff, posset)**

This function evaluates the distortion model generated by uvpolyfit. It is applied to the u component and v component distortion models separately (i.e., use 2 separate call to evaluate.

```
input:
      coeff          the 2-d 2nd order distortion model coeffients.
      posset         the set of distorted, undistorted positions pairs. Only
                     the distorted positions are used in this evaluation.
output:
      return value   the undistorted coordinate value corresponding to posset.
```

---

**twodfit, pos, offsets, q, splinemodel, uvsol, resid**

The 2d spline fitting procedure (unweighted version).Must type @link (on VMS systems before executing to link in sparse matrix routines, will not work on Unix systems without replacing sparse matrix calls with native IDL sparse matrix calls.

```
input:
      pos            the list of multiply observed star positions as generated
                     by multistar.
      offsets        the list of image offsets correponding to each position
                     in pos (the o array produced by multistar).
      q              the list of star ID's associated with pos as produced by
                     multistar.
input & output:
      splinemodel    the structure containing knot locations which is used to
                     return the fit spline coefficients.
output:
      uvsol          the fit undistorted positions for each star.
      resid          the x,y residuals for each observed position.
```

---

**twodfitw, pos, offsets, q, w, splinemodel, uvsol, resid**

This prodedure is identical to twodfit except that it performs a weighted fit (and requires @link in the same way).

```
input:
      pos            same as for twodfit.
      offsets        same as for twodfit.
      q              same as for twodfit.
      w              array of weighting factors associated with pos.
input & output:
      splinemodel    same as for twodfit.
```

```
output:
      uvsol          same as for twodfit.
      resid          same as for twodfit.
```

---

**spline_ev_2d(splinemodel, pos)**

Evaluates the splinemodel at the distorted positions contained in pos.

```
input:
      splinemodel    the spline structure as produced by one of the fitting
                     routines (twodfit, twodfitw, uvfit, or uvfitw).
      pos            an array of positions containing the distorted positons.
output:
      return value   an array of undistorted positons corresponding to pos.
```

---

**uvfit, posset, splinemodel, resid**

The fitting program to fit only spline coefficients given the undistorted positions of all star images (unweighted fit). (Does not need the sparse matrix routines.)

```
input:
      posset         the list of undistorted, distorted position pairs as
                     produced by multiuvstar.
input & output:
      splinemodel    same as for twodfit.
output:
      resid          same as for twodfit.
```

---

**uvfitw, posset, splinemodel, w, resid**

The weighted version of uvfit.

---

**uvpolyfit, posset, w, ucoeff, vcoeff**

This procedure fits a 2-dimensional, 2nd order polynomial to the u and v components of the distortion model. It is intended to be used in place of the spline model fit for the 128x128 format.

```
input:
      posset         an array of pairs of distorted and undistorted star
                     positions as generated by multiuvstar.
      w              an array of weights corresponding to posset.
output:
      ucoeff         the 2-d 2nd order polynomial coefficients for the u
                     component of the distortion model.
```

```
      vcoeff          the 2-d 2nd order polynomial coefficients for the v
                      component of the distortion model.
```

## Utility Routines:

---

**fitlt(xin, yin, xout, yout)**

Fit a linear, 4 parameter transformation to transform from the xin,yin list of coordinates to the xout,yout list of coordinates (offset, scale, and rotation).

```
input:
      xin             array of x positions to transform.
      yin             array of y positions to transform.
      xout            array of x positions to fit to.
      yout            array of y positions to fit to.
output:
      return value    transformation coefficients.
```

---

**fitst(xin, yin, xout, yout)**

Fit a linear, 5 parameter transformation to transform from the xin,yin list of coordinates to the xout,yout list of coordinates (offset, rotation, xscale, yscale) (allows skew).

```
input:
      xin             array of x positions to transform.
      yin             array of y positions to transform.
      xout            array of x positions to fit to.
      yout            array of y positions to fit to.
output:
      return value    transformation coefficients.
```

---

**lintran, xin, yin, coeff, xout, yout**

Applies the transformation fit by fitlt to the list of points xin, yin.

```
input:
      xin             array of x positions to transform.
```

```
      yin            array of y positions to transform.
      coeff          transformation coefficients.
ouput:
      xout           array of transformed x positions.
      yout           array of transformed y positions.
```

---

**panelplot, pos, q, o, splinemodel**

plot the location of all the measured points on the format area along with the locations of the spline knots.

```
input:
      pos            position array as obtained from multistar.
      q              star ID as obtained from multistar.
      o              image offsets as obtained from multistar.
      splinemodel    the spline structure as constructed from pickpanel.
```

---

**plotresid, pos, resid, nx, ny, o, splinemodel, residrange, $**
**[/tadpole, /noknots]**

This procedure is intended to plot the residuals of the fit on the screen. the default mode displays the residuals as small boxes of different intensities on a gray background. The /tadpole option plots the residuals as tadpoles. By default the location of the knots is overplotted. The /noknots option stops the overplotting of the knots.

```
input:
      pos            position array of fit points as input to twodfit(w).
      resid          residual array as returned from twodfit(w).
      nx             x size of the format.
      ny             y size of the format.
      o              offset array as input to twodfit (offsets corresponding
                     pos).
      splinemodel    the spline model structure (used to plot knot locations).
      residrange     a scalar indicating the maximum range of the radial
                     residual value to be displayed in the default mode. For
                     the tadpole mode it is the magnification factor applied
                     to the tadpole vectors displayed.
      tadpole        when nonzero (or when specified as /tadpole) it plots the
                     residuals as vectors with a box indicating the tail of
                     the vector. (optional keyword argument).
      noknots        when nonzero (or when specified as /noknots) it prevents
                     the overplotting of the knot positions. (optional
                     keyword argument).
```

---

**psplotresid, pos, resid, splinemodel, mag, [filename=<filename>, $**
**label=<label>,/encap]**

This is similar to plotresid except is intended to produce paper plots (post-script). All the plots are produced in tadpole mode and knot positions are always overplotted. The postscript file is automatically printed (but the file is not deleted) if encapsulation is not set. The plot includes a scale bar showing the size of a one pixel residual.

input:

| | |
|---|---|
| pos | same as for plotresid. |
| resid | same as for plotresid. |
| splinemodel | same as for plotresid. |
| mag | magnification factor for residual vectors. The same as residrange in plotresid for tadpole mode. |
| filename | optional keyword argument used to specify the name of the postscript file to plot to. |
| label | optional keyword argument to specify a label for the plot. |
| encap | optional keyword argument to specify that the postscript file should be encapsulated for inclusion into another document. The file is not printed in this case. |

---

**skewtran, xin, yin, coeff, xout, yout**

Applies the transformation fit by fitst to the list of points xin, yin.

input:

| | |
|---|---|
| xin | array of x positions to transform. |
| yin | array of y positions to transform. |
| coeff | transformation coefficients. |

ouput:

| | |
|---|---|
| xout | array of transformed x positions. |
| yout | array of transformed y positions. |

---

**tadpole, x, y, dx, dy, factor**

A simple tadpole plotting routine.

input:

| | |
|---|---|
| x | array of x locations of tadpoles (vectors). |
| y | array of y locations of tadpoles. |
| dx | array of the x component of the vectors. |
| dy | array of the y component of the vectors. |
| factor | magnification factor for the vectors on the plot. |