Instrument Science Report WFC3 2014-24

Instrument Science Report ACS 2014-03

Local Pixel Bundles: Bringing the Pixels to the People

Jay Anderson
Dec 9, 2014

ABSTRACT

The automated galaxy-based alignment software packaged developed for the Frontier Fields program (`hst2galign`, see Anderson & Ogaz 2014 and http://www.stsci.edu/hst/campaigns/frontier-fields/) produces a direct mapping from the pixels of the `flt` frame of each science exposure into a common master frame. We can use these mappings to extract the `flt`-pixels in the vicinity of a source of interest and package them into a convenient "bundle". In addition to the pixels, this data bundle can also contain "meta" information that will allow users to transform positions from the `flt` pixels to the reference frame and vice-versa. Since the un-resampled pixels in the `flt` frames are the only true constraints we have on the astronomical scene, the ability to inter-relate these pixels will enable many high-precision studies, such as: point-source-fitting and deconvolution with accurate PSFs, easy exploration of different image-combining algorithms, and accurate faint-source finding and photometry.

The data products introduced in this ISR are a very early attempt to provide the `flt`-level pixel constraints in a package that is accessible to more than the handful of experts in HST astrometry. The hope is that users in the community might begin using them and will provide feedback as to what information they might want to see in the bundles and what general analysis packages they might find useful. For that reason, this document is somewhat informally written, since I know that it will be modified and updated as the products and tools are optimized.

1. INTRODUCTION

HST data is not easy to analyze. Most of its detectors are undersampled, some of them significantly. In addition, there are a non-trivial number of bad pixels and cosmic rays (CRs) in each exposure, such that we cannot take a single image of an object and be sure to know everything that HST can tell us about that object. In general, several exposures must be taken at a variety of dither positions in order to remove the detector artifacts and arrive at an unbiased understanding of the astronomical scene.

Combining these multiple observations is far from trivial. The `Drizzle` software package, which includes the `Multidrizzle` and `Astrodrizzle` improvements, was designed to construct from multiple exposures a single image that was as faithful as possible to the true astrometric scene, even though there is no single way of doing this, and there are drawbacks with every approach. `Drizzle`'s main priority is to preserve flux, and it does this all while removing artifacts and getting the flux as close as possible to its true location. Compromises often must be struck between improving the sampling of the output image and making the output image as regular as possible coverage-wise. Even after striking the best compromise for each particular data set, the user is often left with correlated output pixels that irregularly sample the scene (in that the effective light-center of each pixel is offset somewhat from the center of that pixel). Fruchter has developed some strategies for mitigating the sampling issue (`iDrizzle`, see Fruchter 2011), but such approaches are not widely available and furthermore they do not solve all problems.

Since the only true constraints we have on the scene come from the pixels in the flat fielded "`flt`" images, it would be best if we didn't have to distill these "hard" constraints into the "soft" constraints of a compromised image. It should be possible to respect the pixels while we analyze the scene. One way to do this is to construct parameterized models of the scene and push them into the data domain for comparison with the `flt` pixels. Such an approach requires a detailed knowledge of the point-spread function (PSF), but fortunately accurate PSFs can be constructed and are readily available in the `flt` domain. (This is in contrast to the `Drizzle` domain where each source has different sampling and signal-to-noise properties and therefore a different PSF, depending on the particulars of which input pixels contribute to which output pixels.)

The ideal of working in the `flt` domain has been universally recognized for a long time, but it has been hard to realize because few people have access to an accurate two-way mapping between the greatly distorted `flt` frame of each exposure and an undistorted master frame. The distortion solutions that `Drizzle` uses are very complicated. There is a polynomial solution from the IDCTAB file, along with several supplementary look-up-table based corrections to account for detector defects, filter dependence, and other non-linear issues. The multi-step nature of the correction makes it extremely difficult to smoothly work out the inverse solution. In practice, this mapping has been successfully done within the `Drizzle` software environment, but the fact that it has worked out the forward and reverse solution internally does not mean that a typical user can do it.

In hopes of making these transformations easier, I have chosen to put high-precision distortion solutions for the three main HST imagers (ACS/WFC, WFC3/UVIS, and WFC3/IR) into a simple pixel-based format, which serves as a simple look-up table solution. There are two images for each detector/filter combination: one for the x coordinate and one for the y coordinate, and each image has the dimensions of the detector itself. The value of the x and y images for each pixel provide directly the location in the

distortion-corrected frame to which the pixel maps. The distortion-corrected frame here differs slightly from that of the official solution, since it was constructed for differential photometry rather than absolute mapping into the V2-V3 plane. One could also construct a similar image for the reverse mapping, but we have not done that here; we simply iterate to solve for the inverse mapping of the pixel-based solution when we need it.

The accurate mapping from exposure to master frame that is accomplished by the `hst2galign` procedure allows us to create a new kind of data product for the community. This data product will make it possible for astronomers to make full use of the pixel constraints in the multiple `flt` images when modeling sources. This product will "bundle" together the pixels in multiple flt images, along with simple look-up-tables that will make trivial the transformations from exposure frame to master frame and back again. We will also include in the bundle accurate PSFs to enable the detector signature to be taken out of the model-to-sky fitting.

This report is organized as follows. Section 2 will describe usage of the `hst2galign` procedure, and how it generates the files that allow us to transform from individual exposures into the reference frame. The next section will describe the first of the two new routines, `hst2bundle`, which "bundles" together all the data for a specified source in the stacked image. Section 4 describes the format of the `bundle` files themselves. These files should be amenable to a wide variety of analyses. One can imagine treating these bundles as "objects" in Python and constructing a variety of "methods" that can operate on bundles. I am not yet versed in Python, so I have done the next best thing. I provide a software package written in FORTRAN, `bundle2process`, which provides a variety of tools for analysis of `bundles`. While I realize that most of the community does not use FORTRAN, this routine will show interested users how to make use of the transformations and PSFs to do a variety of analyses, such as: screen-dumping of pixel and mapping data, point-source photometry and astrometry, simple image stacking with various kernels, using deconvolution and iterative forward-modeling to construct an image of the scene, with or without a point source, etc.

2. SETTING UP THE ALIGNMENT

A previous ISR by Anderson & Ogaz (2014) describes the use of `hst2galign` and how to download and run it. This document does not have much to add to that ISR, but for convenience, I will describe the basics of what the routine does to refresh the reader in regards to the files generated.

The alignment routine operates from the command line. A typical call looks something like this:

```
./hst2galign.2014.12.09.e \
   HMIN=3       \
   FMIN=25      \
   PMAX=99999999\
   CRLIM=0.05   \
   INST=WFC3IR  \
   "FILEGCX=/grp/webpages/jayander/GCLIB/WFC3IR/wfc3ir_gcx.fits" \
   "FILEGCY=/grp/webpages/jayander/GCLIB/WFC3IR/wfc3ir_gcy.fits" \
   NPKMIN=0.50    \
   MKSTACK+       \
   DOWCS+         \
   ../00.data/*_flt.fits
```

Note there can be no spaces after the "\" line breaks. Here, the first four parameters describe the finding algorithm that is used to identify sources in the individual exposures. Stars are identified if their brightest pixel is less than `PMAX` but brighter than any others within `HMIN` pixels, and if they have at least `FMIN` pixels over sky and each of the eight surrounding pixels has at least `CRLIM` times the central pixel (after a local sky subtraction). Sources are initially measured with a crude centroid algorithm.

The HST instrument is identified as a 6-character code (`WFC3IR`, `WFC3UV`, and `ACSWFC`). Since the `flt` images for WFC3/IR are in electrons/s, we multiply these images by 1000 in order to deal with integer pixel values. The direct distortion solution is available as a fits file. When collating sources into a master list, each source must be found in at least `NMIN` fraction of the total number of images.

The flag `MKSTACK+` ensures that a stack will be generated (this is important for identification of sources). For WFC3/IR images the units of this stack are returned to e^-/s. The `DOWCS+` flag will ensure that this stack will be constructed in a RA/Dec-oriented frame with a pixel scale of 100, 40, and 50 mas/pixel for the three detectors given above.

The stack is not as rigorously constructed as a product that `Drizzle` would create, in that the routine has not taken special care to rigorously preserve flux. It is designed to simply be *representative* of the scene so that one can identify objects of interest in the master frame and generate bundles of them to analyze later.

The routine goes through each `flt` exposure and identifies all qualifying sources. It then finds the transformation from each exposure into the reference frame defined by the first exposure — this could either be in the direct distortion-corrected frame or in a distortion-corrected frame that has been oriented with North up and East to the left. The sources from all exposures are then collated, and a master list is

made of those sources that could be found in a minimum number of images. The transformations are then used to find an average position for each star in the reference frame, which in turn allow us to improve the transformations themselves. This virtuous cycle is repeated five times in order to arrive at a consistent set of transformations and average positions.

Once the initial transformations have been generated, the routine makes a stack of the input images in the reference frame. As mentioned above, this stack is not rigorous, but it is good enough to allow us to identify sources.

After the stack has been made, the routine goes through five iterations wherein the current-best transformations are used to construct a template for each of the identified common sources. This template is then used to re-measure the source's position in each exposure, re-compute the transformations and then re-compute the average positions. At the end of each iteration, the position residuals for each galaxy are evaluated and galaxies with astrometric errors less than 0.05 pixel are given weights of 1.0 and galaxies with larger errors are given weights that go as $1/(\text{sigma}/0.05)^2$. This virtuous cycle generally results in transformations with errors of 0.02 pixel or better, based on the residual.

The best ways to make sure that the hst2galign routine completed successfully are to (1) examine the stack file, (2) load the region file and make sure that there are a good number of green galaxies, and finally (3) examine the LOGC.XYMEEE.05 file to ensure that there is a reasonable trend of astrometric quality with flux/magnitude. The ISR by Anderson & Ogaz (2014) can give more details on what to look for.

It is worth noting that this routine was primarily designed to stack images with moderately small dithers. It was not designed to deal well with mosaics that do not have large overlap among all exposures. At some point, I do plan to generalize the routine to deal with larger fields, but that involves a very different approach to iterating on the reference frame, and it is not within the current scope.

The `hst2galign` files that are most useful to the bundling are the `LOGD.stack_bar.fits` file (which contains the stack of the scene) and the `image_flt.mat` files (which implicitly define the mapping from each exposure into the reference frame).

3. EXTRACTING THE BUNDLES

Once the `hst2galign` routine has been run, we can start extracting bundles. We will describe the format and content of the bundles in the next section. This section simply gives a prescription for generating them.

The routine that generates them is named: hst2bundle. Below is an example of how to run it:

```
./hst2bundle.e  \
  INST=WFC3IR  \
  "FILEGCX=/grp/webpages/jayander/GCLIB/WFC3IR/wfc3ir_gcx.fits" \
  "FILEGCY=/grp/webpages/jayander/GCLIB/WFC3IR/wfc3ir_gcy.fits" \
  "FILEPSF=/grp/webpages/jayander/WFC3/WFC3IR_PSFs/STDPSF_WFC3IR_F105W.fits"\
  "REFIMAGE=../01.hst2galign/LOGD.stack_bar.fits" \
  UREF=0630 \
  VREF=0979 \
  21x21 \
  ../00.data/*_flt.fits \
  ../01.hst2galign/*_mat.fits
```

Much of this command can be traced back to the hst2galign call, since many of the same reference files are needed. For instance, the instrument is specified in the same way, as is the distortion solution for that instrument.

Here, we also specify a PSF, in what is my current "standard" format. This "standard" PSF format is a way of specifying a spatially variable PSF in an image. There are NX×NY PSFs in the image, with their X and Y centers specified with keywords in the header. The PSFs are then given in a data cube that is 101x101xN_{PSFs}, where N_{PSFs} = NX×NY. There are FORTRAN subroutines provided that take a position in the detector (i, j) and extract the appropriate PSF for that location. The PSF for the target location in each exposure is one of the products provided in the bundle.

The `REFIMAGE` file is the stack generated from the run of `hst2galign`. It is needed so that we can associate a WCS header to the extracted bundle.

The UREF and VREF parameters tell the program the location of the source in the reference frame. It should correspond to the pixel closest to center of the source and will be rounded to the nearest integer, since the extraction is designed to work with whole pixels.

The "`21x21`" parameter tells the routine to extract a 21×21-pixel raster about the center of the source. Additional options are 31, 41, 51, and 101 pixels on a side, for when the object of interest does not fit in a small raster. Note these are always odd, since it is convenient to have a specific pixel at the center.

The `*_flt.fits` wildcard is simply the list of images in the data set. The second wildcard `*_flt.mat` are the mat files generated by `hst2galign`; the routine will match these up one to one, and if there are any `mat` files that don't match any corresponding `flt` files, or vice versa, it will stop. I sometimes find it useful to not keep all the files in the same directory.

The result of running the `hst2bundle` routine is the generation of a file that has a name similar to `bundle_21x21_00667_01348.fits`. The next section will describe the contents of the file.

4. CONTENT OF THE BUNDLES

Each bundle is a fits file with standard extensions. The primary header contains keywords that describe various aspects of the bundle. First, it gives the bundle size in terms of the dimensions of the extracted raster (NP) and the number of contributing images (Ms). Then it lists the images that went into the bundle: the image stem names, exposure times, observation dates, filters, exposure times, the location of the central pixel in the detector frame, the mapped location of that pixel in the reference frame, and finally the central value of the PSF (which tells you what fraction of a star's light will land in a pixel if the star is centered on a pixel).

The bundle contains ten extensions. The first 7 extensions are three-dimensional image cube with dimensions NP×NP×Ms. The final extension is 101×101×Ms in size. The extensions are listed below:

```
1     'SCI'      real*4     pixel values from the flt image
2     'ERR'      real*4     error-array values from the flt image
3     'DQ'       integer*2  data-quality array from the flt image
4     'UPIX'     real*8     mapping from raster into ref frame (x)
5     'VPIX'     real*8     mapping from raster into ref frame (y)
6     'UV2I'     real*4     mapping from ref frame into raster (i)
7     'UV2J'     real*4     mapping from ref frame into raster (j)
8     'PSF'      real*4     the PSF for each flt image
```

We will describe the extensions one by one here. The first three extensions are simply the pixel array, error array, and data-quality-flag array directly from the corresponding `flt` image.

The fourth and fifth extensions (`UPIX` and `VPIX`) provide the mapping from each pixel in the data cube into the master frame. These two images can be used to place each pixel into its proper location relative to all the other pixels in all the other exposures in the coordinate system of the master frame. These arrays can be bi-linearly interpolated to convert a non-integer position measured in one of the rasters (say, from PSF fitting) into its corresponding location in the master frame.

The sixth and seventh extensions (`UV2J` and `UV2I`) provide the reverse mapping: from the pixels in the reference frame into each individual raster. If we have a position (U,V) in the reference frame, then we can place it within the local master-frame raster at

u_{LOC} = HP + (U−UREF) and
v_{LOC} = HP + (V−VREF),

where `HP` is simply the center of the raster `(NP+1)/2`. To determine the exact location within the image-N raster for image N, we simply take `x=UV2I(`u_{LOC}`,`v_{LOC}`,N)` and `y=UV2J(`u_{LOC}`,`v_{LOC}`,N)`, where the 2-D arrays `UV2I(*:*,N)` and `UV2J(*,*,N)` are bi-linearly interpolated at the location (u_{LOC},v_{LOC}).

The final eighth extension corresponds to the PSF that should be applicable within each raster. It has been interpolated from my standard PSF models at the location of the center of the tiles. The PSFs are in

the “effective” format, meaning that they represent the integral over the face of the pixel of the instrumental PSF (the raw PSF as it impacts the detector). As such, the value of the PSF at an offset (Δx,Δy) tells us the fraction of a star’s flux that should land in a pixel (i, j) offset by (Δx,Δy) = (i-$x_\star$, j-$y_\star$) with respect to the star’s center. The PSF is supersampled by a factor of 4, such that the above offset would involve interpolating the array at $x_{PSF} = 51 + 4\times\Delta x$ and $y_{PSF} = 51 + 4\times\Delta y$. The PSF has considerable gridpoint-to-gridpoint structure, so it is *not* safe to simply linearly interpolate this array. One should use a bi-cubic scheme, particularly near the center. There is a subroutine `rpsf_phot( )` that carries out this interpolation properly.

5. ANALYSIS WITH THE BUNDLES

These bundles clearly contain a lot of useful information. In this section we will provide several examples of how to do various analyses on them.

I have created a single software program to do this: `bundle2process`. The user gives the name of the bundle file as the first command-line argument, then a list of tools that perform operations on the bundle. The full list of commands and their syntax can be obtained by simply executing the program without any arguments. We provide this output for reference as the Appendix.

The routine has several running variables that are used in the operation, such as `UTARG` and `VTARG`. These are initially set to be the center of the bundle, but they can be changed with a command-line argument, and this new location will then be the target location. Similarly, a `RADIUS` can be specified and operations that require a radius will use that parameter. If a point source has been fit for position or flux, then that position and flux are loaded into the `UTARG`, `VTARG`, and `ZTARG` variables.

5.1 DUMP(FILENAME)

The dump tool simply takes the current bundle and prints it out to a file named `FILENAME`, given in parentheses. If a radius is specified, then it outputs only those pixels within a certain radius of the target master-frame location (`UTARG`, `VTARG`). Note that this command must be in quotes on the command line to prevent the command-line parser from interpreting the parentheses.

We generated a stack of the 28 exposures of the WFC3/IR F105W Frontier Field observations of the MACS0717 parallel field. We will provide this data set on a website (see the last section) so that users can start making bundles without having to learn the `hst2galign` procedure first. We extracted a bundle about a sharp galaxy at (717,1166) in the image. Then we "dumped" the pixel values using the following command:

```
bundle2process.e bundle_21x21_00717_01166.fits "DUMP(dump.out)"
```

Each line in the dump output is one pixel in the 3-D raster. The output columns give: (1) the image number, (2 and 3) the (i, j) location within the raster, (3 and 4) the (I_{FLT},J_{FLT}) location within the `flt` frame, (5) the pixel value, (6) the sky value, (7) the error-array value, (8) the DQ-array value, (9 and 10) the mapping of this pixel into the master frame, (11 and 12) the offset between the global target location and this pixel, and (13) the corresponding PSF value. If individual-exposure positions have been solved for, then an additional three columns provide the offset and PSF for the position solved for in each individual exposure.

We will use the `DUMP` tool to illustrate the outputs of various other tools, but here we will simply show where the pixels in a raster map to in the master frame by plotting columns 9 and 10 onto the stack of the scene for an extracted 21×21 raster. The small yellow points in **Figure 1** correspond to the mapped locations of every pixel in the bundle. The larger blue points show where the pixels from the first exposure map to.

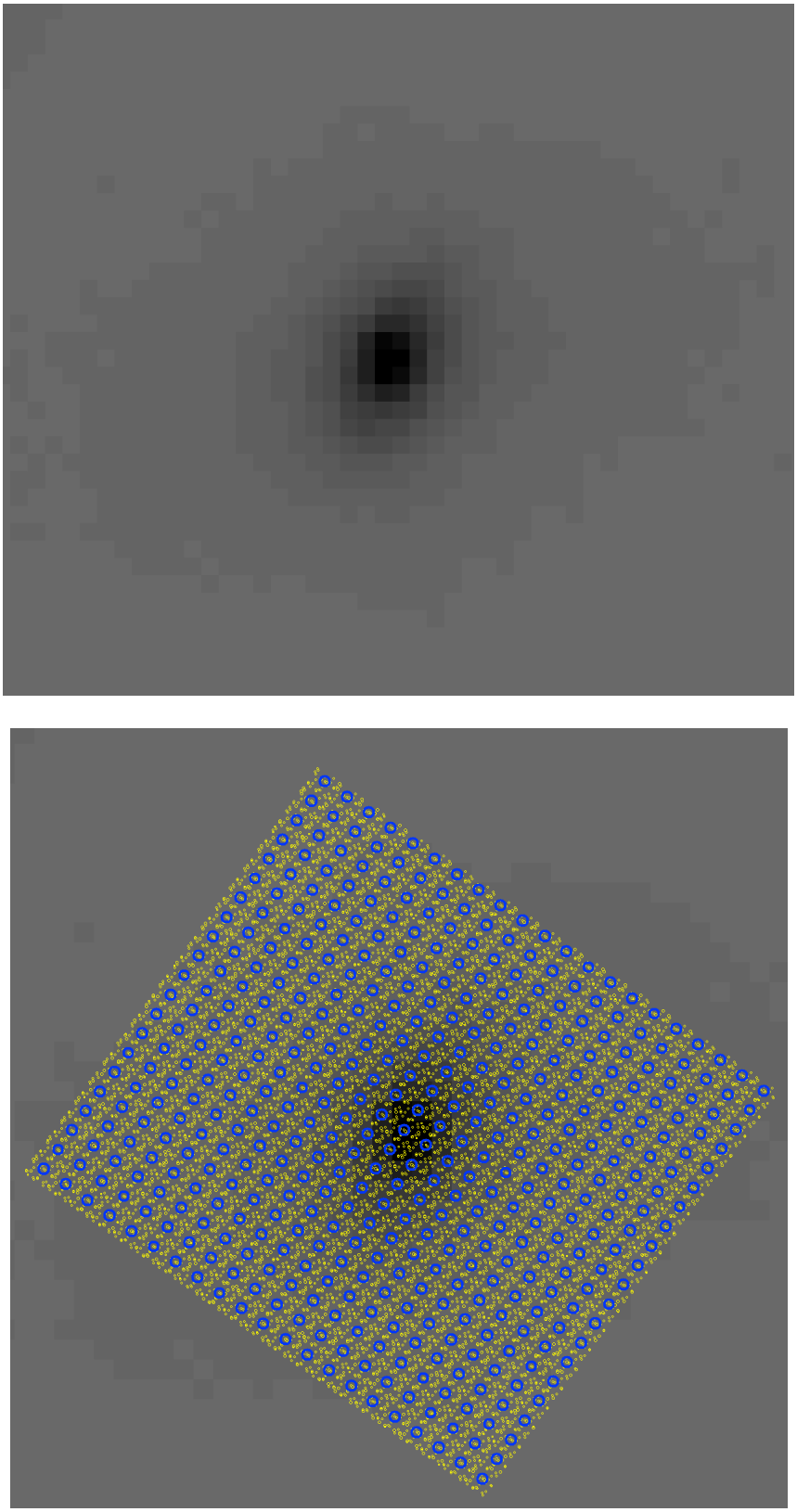

Figure 1: (Top) 1×-supersampled stack of a galaxy in the reference frame. (Bottom) the locations where the 28 images sample the scene (ie, where each pixel maps to). The bigger blue circles show where the first exposure samples scene.

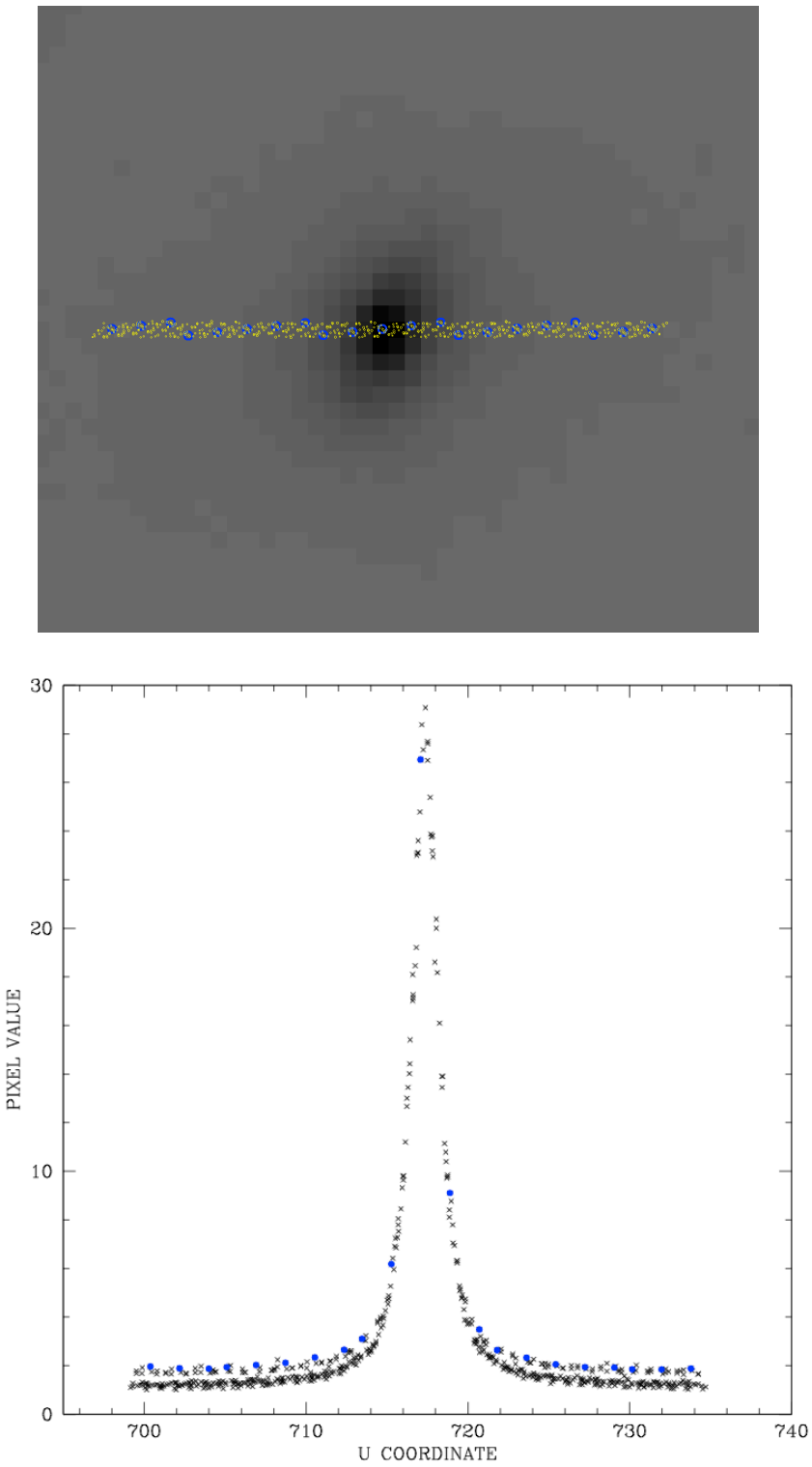

Figure 2: (Top) the same galaxy, but showing on the samples within ±0.5 pixel of the center horizontal line. (Bottom) the pixel values for each of these samplings. This is a slice through the galaxy. The smooth nature of these samples shows that the scene truly is continuous and not intrinsically pixelated. The blue dots show the points from the first exposure, and the black dots from all the other 27 exposures.

5.2 PIXCOMP-RELATED TOOLS

The pixels in the above profile clearly show some systematic issues. Some of the exposures have a much higher sky than the others. We provide a tool in the bundle2process package that re-normalizes the background. To do this, it iteratively compares each exposure against all the others to determine whether all the pixels in one exposure are systematically high or low relative to the others. In addition to looking for zeropoint-type offsets, this comparison also allows us to reject bad data. The routine flags all points that are 7-sigma discrepant from the others as "primary" defects. It then examines the pixels adjacent to each of these pixels and identifies a "secondary" defect as being more then 3.5-sigma discrepant. The routine runs for a default of 10 iterations, or until there are no new pixels to reject. This tool is called `PIXCOMP`.

We ran `PIXCOMP` on the above bundle and then ran the `BKGD_MATCH` tool and dumped the result. We used the following command:

```
bundle2process.e bundle_21x21_00717_01166.fits \
      PIXCOMP PIXREJ BKGD_MATCH "DUMP(dump_norm.out)"
```

The result was the following for the fifth iteration:

NIT	NIM	SBAR_M	ADJ	SIG	Ns	Nu	REJ1	REJ2	NEW1	NEW2
05/05	1	0.5067	-0.0068	0.0612	440	414	1	0	0	0
05/05	2	0.4266	-0.0068	0.0708	439	419	2	0	0	0
05/05	3	0.3052	-0.0068	0.0490	441	412	0	0	0	0
05/05	4	0.3234	-0.0068	0.0512	441	416	0	0	0	0
05/05	5	0.3539	-0.0068	0.0567	440	415	1	0	0	0
05/05	6	0.3454	-0.0068	0.0672	441	419	0	0	0	0
05/05	7	0.3552	-0.0068	0.0607	441	416	0	0	0	0
05/05	8	0.4871	-0.0068	0.0916	440	421	1	0	0	0
05/05	9	-0.0378	-0.0068	0.0391	438	407	1	2	0	0
05/05	10	-0.1395	-0.0068	0.0359	440	414	1	0	0	0
05/05	11	-0.1265	-0.0068	0.0378	441	411	0	0	0	0
05/05	12	-0.1223	-0.0068	0.0340	441	405	0	0	0	0
05/05	13	-0.0125	-0.0068	0.0362	441	411	0	0	0	0
05/05	14	-0.0814	-0.0068	0.0381	439	410	2	0	0	0
05/05	15	-0.0655	-0.0068	0.0405	441	416	0	0	0	0
05/05	16	-0.0694	-0.0068	0.0365	441	410	0	0	0	0
05/05	17	-0.0887	-0.0068	0.0469	439	411	2	0	0	0
05/05	18	-0.1364	-0.0068	0.0347	439	408	2	0	0	0
05/05	19	-0.1270	-0.0068	0.0385	441	412	0	0	0	0
05/05	20	-0.1387	-0.0068	0.0381	441	408	0	0	0	0
05/05	21	-0.1595	-0.0068	0.0488	441	417	0	0	0	0
05/05	22	-0.2204	-0.0068	0.0338	439	406	2	0	0	0
05/05	23	-0.2067	-0.0068	0.0387	441	412	0	0	0	0
05/05	24	-0.2011	-0.0068	0.0356	441	411	0	0	0	0
05/05	25	-0.2649	-0.0068	0.0377	441	410	0	0	0	0
05/05	26	-0.3102	-0.0068	0.0385	441	408	0	0	0	0
05/05	27	-0.2997	-0.0068	0.0339	441	409	0	0	0	0
05/05	28	-0.2951	-0.0068	0.0349	441	410	0	0	0	0
05/05	28	0.0000	-0.0068	0.2635	28	28	15	2	0	0

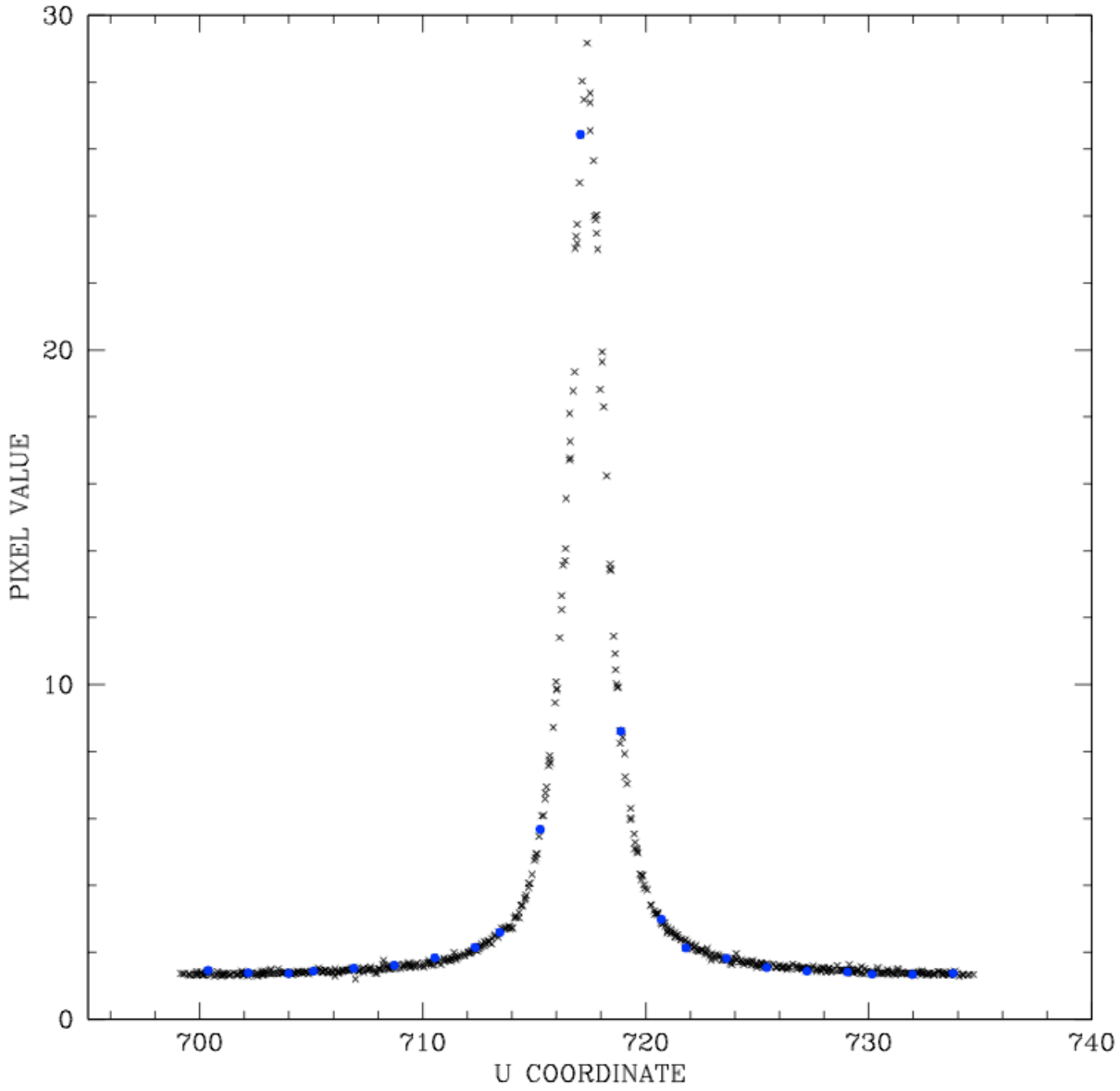

Figure 3: The same as the bottom panel in Figure 2, except now the pixels in each exposure have been zeropointed up or down to better agree with the others. Again, the blue points are those from the first exposure, and the black ones from the other 27 exposures.

The SBAR_M column shows the DC-offset for each exposure, relative to the average of the others. Note that the first exposure is indeed high relative to the average, as we saw in **Figure 2**. The REJ1 and REJ2 columns show the number of primary and secondary rejections determined for each exposure. **Figure 3** shows the new slice across the object; it is clear that the images are consistently background-registered now.

The DUMP facility can also be good for looking at faint objects. In **Figure 4**, we isolate a faint object at (at i=692, j=1248 in my frame or 07:17:19.04, +37:50:22.22 in the WCS frame). The image and

"dumped" profile are shown. It is clear that this will be easier to fit in the raw pixel space than in the stacked image.

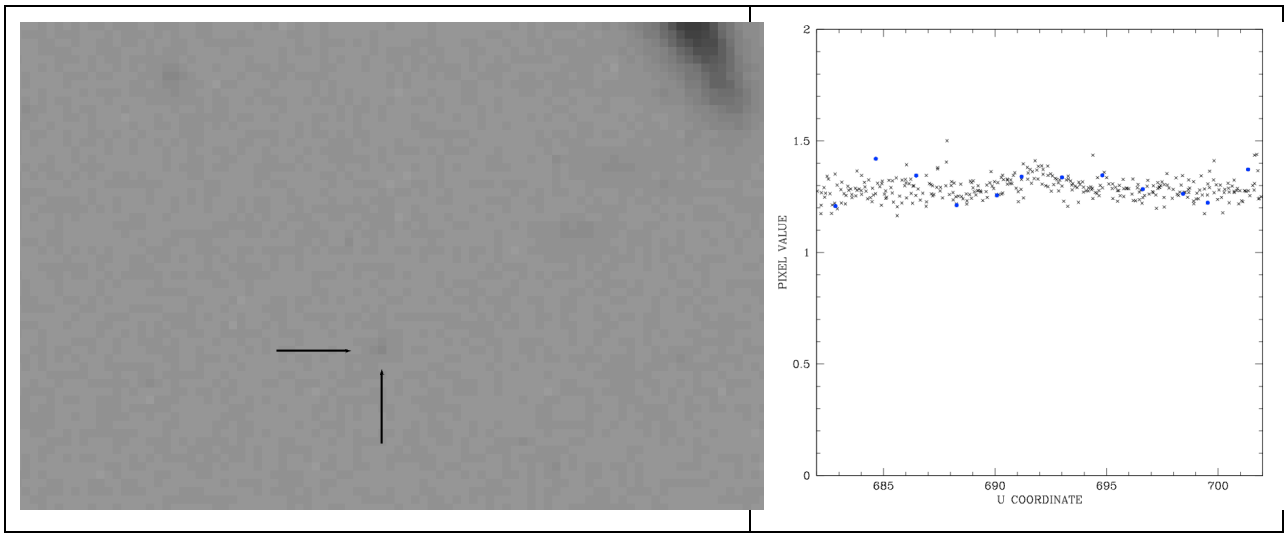

Figure 4: (Left panel) Image of a very faint source. (Right panel) The "dumped" mapped pixels in the vicinity of the source. It is clear that the source shows up as a ~1-sigma detection in most of the 28 exposures. The arrows are 9 pixels in length, so the region shown in the plot is about twice the length of an arrow. Again, the first exposure's pixels are shown in blue.

Here are all the other tasks that work with the output of `PIXCOMP`, or do similar things. These are:

`BKGD_MATCH` — this removes the DC offset identified by the routine from each exposure

`PIXREJ` — this sets the pixels that were flagged by PIXCOMP to the LOBAD value.

`PIXREJ_BY_DQ(FLAG)` — this uses FLAG & the DQ array to set pixels to LOBAD

`PIXREJ_TO_DQ` — this pushes the bad-pixel flag set by `PIXCOMP` into the DQ array

`SKYSPEC(r1,r2)` — this specifies what radius to use to measure sky

`SKYFIND` — this finds a sky value for each exposure

`SKYSUB` — this removes the found sky value from each exposure

`WRITE_BUNDLE(file)` — this outputs a new bundle, with the pixels adjusted/flagged

5.3 THE PSF-RELATED TOOLS

In the eighth extension, the bundle contains a "library" PSF. This PSF was constructed from dithered sets of images of star fields, often at or near the center of Omega Cen, since it has a nice dense, flat distribution of stars. In fact, in support of the Frontier Fields program we took observations of the center of Omega Cen through the ACS filters F435W, F606W and F814W (the FF filters in that detector) and simultaneously of an outer field with WFC3/IR through F105W, F120W, F140W, and F160W (the FF filters for that detector). So, there should be good PSF models for many of the data sets of interest. (It is

worth noting that the ACS PSF and distortion solution appear to have changed somewhat from pre-SM4 to post-SM4, but that is beyond the scope of this effort. I do provide separate PSFs and distortion solutions for pre- and post-SM4.)

As described above, the PSFs are stored in a $101\times101\times N_{PSFs}$ data cube, with each PSF applying at a particular fiducial location within the image. A PSF for a specific, non-fiducial location is interpolated between the neighboring fiducial PSFs. The PSFs are supersampled ×4 relative to the flt pixels, so they must be interpolated to be used. There is so much structure at the center of the PSFs that they should not be linearly interpolated, rather one should use bi-cubic interpolation, as is done in the `rpsf_phot( )` routine. This is mentioned above, but I repeat it here since it is important.

The PSFs are naturally oriented in the detector frame, as are the rasters in the pixel bundles. The master frame is generally oriented with North up and East to the left. The routine `hst2galign` is able to align images that are taken at a variety of orientations, and the PSFs will not necessarily be aligned in the output frame, but the software should naturally be able to take this into account when fitting stars or doing deconvolution.

We extracted a bundle centered on a star in the frame at (638,981). We then ran the following command, which fits a PSF to each individual exposure.

```
bundle2process.e bundle_21x21_00628_00981.fits PIXCOMP PIXREJ /
    BKGD_MATCH XYZFIT_IND "DUMP(starfit_628_981.dump)"
```

The output of the fit is here:

```
#--  ------- ------- ------- ------ -------- -------- ------ --------- ------  ---------
# M    U_M     V_M     M_M    Q_M       X_M      Y_M     S_M      Z_M    C_M     RDATE_M
#--  ------- ------- ------- ------ -------- -------- ------ --------- ------  ---------
001  628.392 980.783  -5.482 0.0443  716.778  612.157  1.351    155.84  7.671  2014.7329
002  628.427 980.721  -5.480 0.0316  715.249  614.709  1.347    155.63  3.153  2014.7332
003  628.433 980.722  -5.496 0.0271  741.047  638.906  1.374    157.94  2.900  2014.7333
004  628.422 980.735  -5.483 0.0192  739.563  641.368  1.342    156.02  2.001  2014.7335
005  628.407 980.725  -5.483 0.0378  718.271  611.717  1.371    156.05  7.668  2014.7336
006  628.415 980.758  -5.489 0.0257  719.803  609.203  1.362    156.95  2.705  2014.7339
007  628.425 980.699  -5.480 0.0205  744.051  635.932  1.366    155.58  2.382  2014.7340
008  628.402 980.751  -5.523 0.0383  742.602  638.371  1.318    161.83  3.570  2014.7343
009  628.359 980.783  -5.433 0.0842  717.575  610.178  1.386    149.00  7.486  2014.7548
010  628.447 980.722  -5.481 0.0687  716.000  612.675  1.351    155.70  6.332  2014.7550
011  628.405 980.726  -5.484 0.0516  741.832  636.895  1.341    156.21  6.136  2014.7551
012  628.412 980.751  -5.482 0.0564  740.323  639.351  1.365    155.82  5.867  2014.7554
013  628.361 980.716  -5.468 0.0602  717.432  616.985  1.337    153.88  5.682  2014.7555
014  628.479 980.625  -5.366 0.0888  718.847  614.498  1.367    140.05 13.795  2014.7557
015  628.424 980.705  -5.490 0.0501  743.187  641.112  1.352    157.04  4.888  2014.7559
016  628.419 980.758  -5.485 0.0625  741.696  643.600  1.371    156.32  6.002  2014.7560
017  628.423 980.763  -5.474 0.0292  721.918  611.267  1.373    154.67  3.196  2014.7582
018  628.469 980.727  -5.385 0.0954  720.359  613.759  1.366    142.58 11.138  2014.7584
019  628.416 980.724  -5.494 0.0296  746.192  637.966  1.356    157.66  3.233  2014.7585
020  628.423 980.759  -5.480 0.0352  744.703  640.430  1.349    155.54  3.508  2014.7588
021  628.389 980.770  -5.466 0.0574  721.749  612.931  1.339    153.65  7.676  2014.7635
022  628.418 980.751  -5.429 0.1061  720.227  615.403  1.352    148.42 14.144  2014.7637
023  628.461 980.725  -5.473 0.0579  745.990  639.553  1.358    154.58  5.813  2014.7639
024  628.385 980.710  -5.485 0.0399  744.525  642.057  1.361    156.38  4.121  2014.7640
025  628.442 980.731  -5.473 0.0364  717.205  612.258  1.353    154.62 11.809  2014.7745
026  628.409 980.747  -5.494 0.0317  718.737  609.745  1.354    157.60  3.067  2014.7748
027  628.439 980.730  -5.488 0.0283  743.006  636.378  1.355    156.68  3.100  2014.7749
028  628.405 980.738  -5.495 0.0310  741.527  638.873  1.359    157.75  3.395  2014.7751
#--  ------- ------- ------- ------ -------- -------- ------ --------- ------  ---------
# M    U_M     V_M     M_M    Q_M       X_M      Y_M     S_M      Z_M    C_M     RDATE_M
#--  ------- ------- ------- ------ -------- -------- ------ --------- ------  ---------
028  628.422 980.735  -5.479 0.0445  AVG
024    0.020   0.019   0.025 0.0257  RMS
```

The routine fits the PSF to each raster, determines a position and flux for the star by fitting the PSF to the inner 5×5 pixels. It then transforms the positions into the master frame. The first column gives the image number and the second two columns the position in the reference frame, which corresponds to the position measured in the raster. The third column is the instrumental magnitude, the fourth the quality of fit (this is the absolute value of the fractional residuals relative to the model). Then it gives the position for the star in the detector frame, along with the determined sky, flux, and chi-squared. Finally, we give the time of the observation in fractional years. For observations taken over a long period of time, proper motions can sometimes be seen.

The final two lines (below the footer) report the average position and flux and the RMSs about the average. Generally, we can do astrometry of bright stars to better than 0.01 pixel, so this indicates some minor problem with the transformations. Even so, the astrometry is quite good.

Note that at the end of the `bundle2process` call, we "dumped" the pixels to a file. This new file contains the PSF expectation value for each pixel (i.e., the fraction of light from the point source that should land in each pixel). Below, we plot the pixel value (minus sky) against the predicted PSF value. The pixels in the first exposure are shown as filled blue circles. The line corresponds to a slope of 155.45 counts/second total. The plot on the left shows the PSF value as inferred from the position measured in each individual exposure. The plot on the right uses the average position in the UV frame mapped back into the individual frame. The correlation on the left is slightly tighter, but not very. There are some discrepancies in the PSF in between 0.2 and 0.3 on the horizontal axis, but the errors are less than 10% RMS. Overall the library PSF appears to provide a pretty good model to the data. This gives us confidence that if we use the PSF to fit deconvolved profiles of galaxies, our errors will be relatively small.

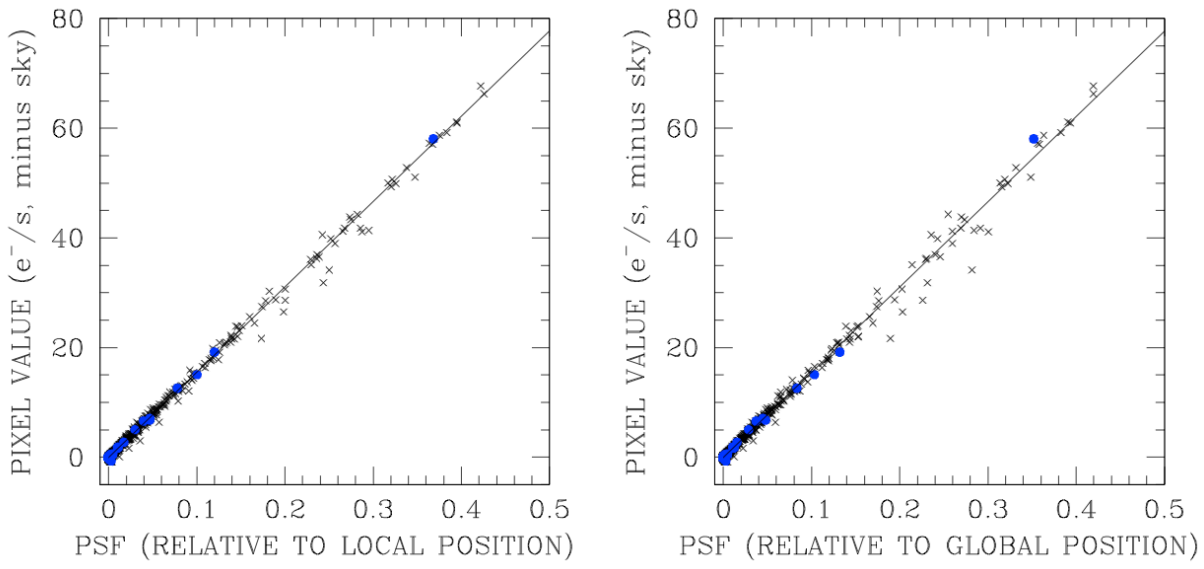

Figure 5: (Left) the plot of PSF model value against observed pixel value for the case where the PSF was fit independently in each exposure. (Right) the same, but where the PSF model was determined with respect to a single master-frame position for all exposures.

All the PSF-related tools are:

`XYZFIT_IND` — this does independent PSF fits, then averages and resets target location/flux

`XYZFIT_TOG` — this does a simultaneous PSF fit, then resets target location/flux

`TARGET(u,v[,z])` — resets target location (u,v) and possibly flux (z)

`STARSUB[(u,v,z])` — subtracts the target from each individual exposure

`SKYSUB` — the above fitting also re-find sky; this can be subtracted from each raster

`WRITE_BUNDLE(file)` — this outputs a new bundle, with the pixels adjusted/flagged

One tool I have not yet written is something to do two-source fitting. That would be a good tool to write, but exactly how to do it might well depend on the signal to noise and other issues. It may be hard to converge on the right answer without a good starting point. This is one of the possible future tools I mention in the last section.

5.4 STACKING-RELATED TOOLS

The final main collection of tools that the `bundle2process` routine contains relate to the generation of various kinds of stacks. The goal of the package is not to reproduce the things that Drizzle does, nor is it to be carefully rigorous. For example, I have not given careful thought as to whether or not the pixel-area correction has been done or needs to be done[1]. Rather, the goal is to allow a qualitatively different kind of analysis.

The stacking procedure is called by the "`MAKE_STACK`" command. Parameters for the stack creation are preset, they are:

`MAKE_STACK` — generates the stack using the current set parameters

`EFF` — this flag generates an "effective" stack; one that is forward modeled with no PSF

`INS` — this flag generates an "instrumental" stack; one that is forward modeled with the PSF

`STACKNIT=20` — this tells the stacking routine how many iterations to go

`TARGET(u,v)` — resets target location; stack is centered on `(UTARG,VTARG)`

`FITPS-` — tells the routine not to seek a point-source at the center of the stack

`FITP+` — tells the routine not to seek a point source at the center

`STACKR(file)` — this outputs the stack (like a pixel pix-frac; the pixels are corelated)

`STACKI(file)` — this outputs the stack (like a point pix-frac; pixels are independent)

The stacking is preformed in an iterative forward-modeling procedure. The stacks we construct here are 2× supersampled relative to the image pixels. This means that a 21×21 pixel raster will generate a stack that covers the inner 11×11 true pixels. The reason for this is that the rasters can be at any orientation relative to the stack, but since they are always the same size, the stack has to be smaller to ensure that all of the rasters cover the entire stack.

The "scene" is initially zero everywhere. At the beginning of each iteration, the current scene is "blotted" into each exposure. If we are trying to construct a deconvolved stack, then the (instrumental) scene is convolved with the PSF then resampled into the frame of each exposure. If we are constructing a direct (effective) stack, then the scene is just resampled into the frame of each exposure. If we are solving for a point source, then the point source is subtracted from each exposure at this stage. Even if the scene is treated in an effective way (ie, not deconvolved), the point source is treated as being a point.

---

[1] A pixel-area correction can be necessary if one wants to measure absolute fluxes rather than surface brightnesses. When there is distortion present, one must decide whether to flat-field the data in a way that preserves total flux or surface brightness. The convention adopted by the pipeline for the `_flt` pixels has been to preserve surface brightness. Thus, if one wants to measure an absolute flux, one needs to correct the measured flux with the PAM (pixel-area map).

Once we have an estimate of what the current scene should look like in each exposure, we subtract it from each exposure. We then go through the stack, pixel by pixel, and determine how it should vary in order to reduce the residuals between the observations and the estimates. Once we have a new estimate of the stack, we smooth it with a smoothing kernel. Currently, this is hardcoded to be a 7×7 planar kernel for the first two iterations, a 5×5 planar kernel in the next two, a 3×3 planar kernel in the fifth and sixth iterations, and beyond that it uses a 7×7 quartic kernel. This last kernel has 15 degrees of freedom for 49 points, so essentially every 3+ "pixels" in the stack are independent, but quartic variations over 7 pixels are preserved. There are four stack pixels for every `_flt` pixel (since the stack is ×2 supersampled), so this corresponds to each pixel having more than one degree of freedom. This 7×7 quartic kernel is equivalent to fitting a 2-d quartic function to every 49 pixels and replacing the pixel value with the value of the quartic. This smoothing approach is my way of ensuring that the pixels in the stack do not vary chaotically from point to point. The reason we begin with more smoothing than we end up with is that we want to converge on the smoothest possible scene, and by starting smooth it is often the case that unreal fluctuations are damped out before they get "set in".

The iterative process allows each pixel to influence the stack indirectly. With the one-pass `Drizzle` procedure, each input pixel must contribute directly to one or several output pixels, but there is no way to let its contribution depend on the slope of the scene, or the residual between the scene and the pixel value at the location of the pixel. As such, the light-center of drizzled pixels are often off from the center of the pixels themselves, since it is rare to have the average of the contributing pixels map precisely to the center of the pixel. This iterative procedure allows each pixel to contribute to the scene at the location where it truly is, since the scene is not just represented by the gridpoints, but rather is a continuous function constrained by the gridpoints.

If you turn on the `SHOWWORK` flag, then the stacking routine will output a summary of the changes in the scene for each iteration in the image file `LOG_STACK_REPORT.fits`. The iterations are shown for the bundle extracted from (821,1194). Each row (from bottom to top) shows the results of a single iteration. The first column gives the starting out scene, the second the raw adjustments that the residual images tell the stack to make. The third is the difference between the requested adjustments and what the smoothing allowed. The fourth column gives the actual changes. The fifth column gives the new stack. Finally, the sixth column gives the stack with independent pixels (the adjustment for each stack pixel came only from pixels in the rasters that map most closely to that stack pixel).

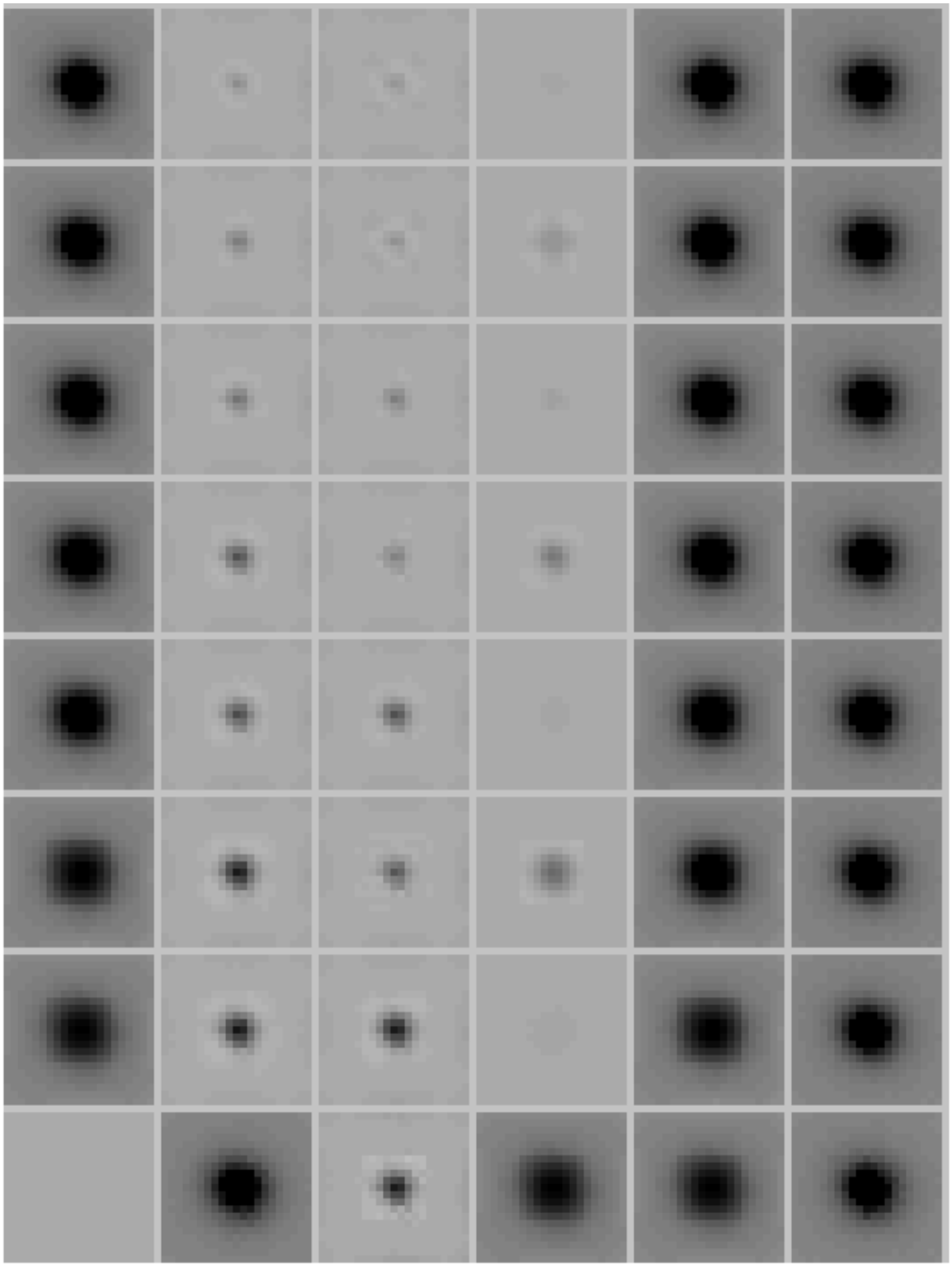

Figure 6: This is the `SHOWWORK` output of the first eight iterations of the `MAKE_STACK` routine. Each row (from bottom to top) is the result of one iteration. The columns from left to right are (1) the previous model, (2) the recommended adjustment, (3) the adjustment that was disallowed because of the smoothness constraint, (4) the net adjustment, (5) the new model, and (6) the new model with independent pixels.

The stack shown in **Figure 6** clearly is sharper at the center than is allowed by the smoothing kernel. We can try to see whether or not putting a point source at the center will help. We may want to run `XYZFIT_TOG` to get a good position for it first. Then it will center the stack on this location before solving for the scene with a point source. The point-source solving iterations begin in the same way as the regular stacking. But after the eighth iteration, if there is any excess flux (above that allowed by the smoothed image) in the central 3×3 pixels of the stack, then it puts some of that excess flux into a point source. This source then gets added to the scene for each of the raster estimates, and then the composite blot is subtracted from each raster and the residual stacked to see how the stack should change. Eventually, if there is something sharp at the center, this excess flux goes into the point source as the smooth part of the scene adjusts to reflect that. The reason we do it this way is that the non-point source part of the scene is encouraged to be as significant as possible, and only if it gets too sharp do we associate that with a point source[2].

The stacks get output into multiple-extension fits files. The first extension (`'BAR'`) contains the smooth part of the scene. The second (`'SIG'`) contains the RMS of the adjustment-estimates for each pixel. The third (`'SRC'`) contains the source, which will have just one pixel. The fourth extension (`'TOT'`) contains the total scene (the first plus the third). The fifth extension (`'NNN'`) gives the number of contributing exposures to each pixel in the stack. The sixth (`'UUU'`) gives the number used in adjusting each pixel (some rejection is done at the stacking stage, even if there is rejection done earlier). Finally, the seventh and final extension (`'PKM'`) gives the peak-map, which just tells you how many local maxima there were in the individual exposures that mapped to each stack pixel. This can help tell when there are faint sources present, since the statistics of counting peaks can be easier than dealing with pixel values themselves, which can have variable signal to noise (see Anderson et al. 2008b).

Note that the stack is reported in terms of surface brightness, rather than flux. By this I mean that if the background is 10.0 in each exposure, then the background of the ×2 supersampled stack will be 10, and not 2.5. Similarly, if the flux in the point source is 100, the central pixel of the third extension will be 400, since that is the surface brightness in the central 0.5×0.5 pixel. In the above example, it turned out that about half of the flux in the central pixel of the no-point-source stack ended up going into the point source.

For completeness, the command used to generate the stack described above was:

```
bundle2process.e bundle_21x21_00821_01194.fits \
      PIXCOMP PIXREJ BKGD_MATCH INS SHOWWORK XYZFIT_TOG \
      FITPS+ STACKNIT=25 MAKE_STACK "WRITE_STACKR(stack.fits)"
```

When the stack is generated from the `MAKE_STACK` command, both the "pixel-correlated" version and the "pixel-independent" versions are constructed. The only difference between them comes in the very last iteration. In the former version, the last-iteration adjustment is made by using the closest pixel in each raster to each stack pixel. In the latter version, the last-iteration adjustment is made by using only pixels in rasters that map most closely to the particular stack pixel. As such, the "independent" stack has much more noise than the "correlated" one. Since both are generated, then user simply decides which one

[2] This is of course only one way of deconstructing a scene into a point-source component and a smooth component; I just wanted to give a simple example of how it might be done.

to output (`WRITE_STACKR` for the regular stack, and `WRITE_STACKI` for the independent stack). One must specify before running `MAKE_STACK` whether the stack is to made with or without deconvolution, or with or without point-source fitting. The images are made with a valid WCS header, as well as "physical" coordinates that map directly to the reference frame. I will leave it as an exercise for the interested user to see what happens when one tries to stack point sources, with and without the FITPS option turned on.

5.5 Other Command-Line Options

There are a few command-line commands that we have not touched on yet. Many of them simply allow the user to specify parameters. They are:

`VERBOSE` — maximal output; for diagnostic purposes

`TERSE` — turn off verbose (default)

`SHOWWORK` — this flag makes the operations generate some table-type output

`LABEL` — unused thus far

`HIFLAG=60000` — sets the HIBAD flag

`LOFLAG=-25` — sets the LOBAD flag (things flagged as bad are set to this)

`STOP` — halts operation of the program

`PAUSE` — pauses the program and waits for keyboard input to continue

`SKIP` — goes on to the next command

6. AS-IS DISCLAIMER

The goal of all these tools is give users a sense of how one might make use of the pixels and the mappings that are provided in the bundles. I don't expect that many of these tools can be used directly for science, but they should give people a sense of what can be done with the pixels and the mappings, and how to do it. My aim in this is just to give people access to the pixels in a way that they can interrelate them with the scene and each other, and also use PSFs to do more accurate modeling of what's really out there.

It is worth noting that I don't expect that this software package is in its final form. Indeed, I have been rushing and cutting corners in an effort to get it out the door sooner, so that people can kick the tires and see how they might want to tweak it to make it better. I can't promise to implement every possible desire-ment people may have, but if there are some easy things I can do, I will be happy to do them. And if there's anything that isn't working and should be, I'll definitely fix that as quickly as possible. You should not trust that the program does everything it says — you should test it and make sure that all the outputs make sense.

On a personal note, I would like to see this approach to analysis take off. It would be nice if one day we could say to the archive: "Give me every pixel that HST has observed within 0.5 arc-second of a particular RA/Dec. Tell me when each pixel was observed, what area on the sky it covers, what filter was used, etc." And a tool should just be able to ingest all of this and come up with a consistent model for the source, be it a time-series of flux, a 2-d profile, a color, a point-source extraction, etc. This document and package is one limited initial step in that direction.

I am not an object-oriented programmer, but I could imagine that if this "bundle" format catches on (perhaps with some modifications), one could easily treat each bundle as an "object" in Python and could write code that would provide some "methods" to operate on it and produce results. Essentially, the commands in my `bundle2process` environment are very close to that. I don't think in writing these commands I have made too much use of FORTRAN's speed, so perhaps much of it could be rewritten in a more convenient language. I leave that up to others, but I think that the "bundle" format might need to attain its final form before people will start writing involved packages for it.

One issue that I do see as potentially problematic is that the current version of the `hst2galign` software treats different datasets completely independently. By that I mean that if there are sets of observations of a field in F606W and in F814W, it will reduce the F606W data separately from the F814W data. They won't even have the same reference frame, since the reference frame for each comes from the first exposure of each, and even if they are at the same POS-TARG, the fact that each filter has a different offset relative to the others and a different distortion solution (on account of filter-dependent fingerprints) means that the frames will not be lined up perfectly.

I can imagine that it would be nice to be able to have a F814W bundle and a F606W bundle that are constructed in the same reference frame. One might even want to have a bundle that contained observations from multiple filters, but that is going even farther than I'm ready to go now.

It shouldn't be hard to add the flexibility to the `hst2galign` to allow the user to provide "input reference frame" for the images to align with. If people would like to see this happen, let me know. It would ensure that the bundles extracted from data sets through different filters would be lined up in the (U,V) frame.

The bundles that I have worked with in this document have all been 21×21-pixel extractions. The `hst2bundle` routine is currently able to extract regions up to 101×101 pixels in size. This starts to become interesting in terms of a systematic analysis of the field. One can imagine dividing the field up into many such regions, extracting bundles, then running finding and fitting operations on these to create source catalogs, perhaps even extracting local bundles for each object of interest.

On a related note, I have been working on a program (called "KS2") that reads in all the images for a star field and reduces the entire data set simultaneously in local patches. It does iterative waves of source-finding and source-subtraction in multiple filters for multiple detectors, with all the detailed analysis being done on the `flt` images where the PSF is well characterized and reliable models exist. The routine also does artificial-star tests. The KS2 routine is an improvement on the code I wrote to reduce the GO-10775 (PI-Sarajedini) for the ACS Globular Cluster Treasury Program (see Anderson et al 2008a). One could imagine such a "method" for the sizeable bundles. At the moment, KS2 is not publicly available, but someday it (or a modification of it) might be.

7. REFERENCES

Anderson & Ogaz "The Software Package hst2galign" ACS/ISR-2014-XX and WFC3/ISR-2014-YY.

Anderson et al. 2008a **AJ** 135 2055. "The ACS Survey of Globular Clusters. V. Generating a Comprehensive Star Catalog for each Cluster."

Anderson et al. 2008b AJ 135 2114. "Deep ACS Imaging in the Globular Cluster NGC6397: Reduction Methods"

Fruchter, Andrew S. 2011 PASP 123 497. "A New Method for Band-Limited Imaging with Undersampled Detectors"

8. DOWNLOADS

Below are a few interesting places to find the programs and reference files for downloading. If you are not able to find a needed PSF or distortion correction for a particular filter/detector combination, you can use a close one or ask me if I can construct it quickly.

http://www.stsci.edu/~jayander/FF_SOFTWARE

http://www.stsci.edu/~jayander/STDPSFs

http://www.stsci.edu/~jayander/GCLIB

A. APPENDIX

This is what is printed out when the bundle2process routine is executed without arguments. It is a nice reference for what kinds of tools it provides.

```
> ./bundle2process.e <cr>

 Program operation...

 bundle2process.e bundle.fits COMMAND1 COMMAND2...

 LIST OF COMMANDS
       all are optional (except the first)
       executed in order
       no spaces within an argument
          without surrounding quotes
       commands with symbols often must be enclosed
          in surrounding double quotes in order
          to prevent the command-line from doing
          the wrong thing

    VERBOSE/TERSE
       maximal/minimal output to the screen
    SHOWWORK/HIDEWORK
       output to output files
    LABEL(bundle_101x101_0900_1204)
       define a label to be used for outp files
       (does not do anything yet)
    SKYSPEC(r1,r2)
       define where sky is extracted from
       defaults are 8 and 12 (must be integers)
    SKYFIND[(r1,r2)]
       can include r1,r2 or will use default
       find sky for each raster
       centered on target
    SKYSUB
       subtract sky from each raster
    PIXCOMP[(SIG_PRIM,SIG_SEC,NIT)]
       does the inter-raster comparison
          fills rejection array
          finds the background match val for each
    PIXREJ
       set rejected pixels to LOFLAG
    PIXREJ_BY_DQ(DQBADFLAGS)
       flag pixels as bad (LOFLAG) according to
       the DQ array value
    PIXREJ_TO_DQ(BADFLAG)
       turn on flags in DQ array according to
       whether pixels were PIXCOP rejected
    BKGD_MATCH
       use output of PIXCOMP and match backgrounds

    TARGET(u,v[,z])
       specify a location in the image to start
       point-source ops or stack-centering
```

RADIUS=3
   Specify a radius to do various fitting things
XYZFIT_TOG[(utarg,vtarg,RFIT)]
   also resets target to the output
XYZFIT_IND[(utarg,vtarg)]
   also resets target to the output
STARSUB[(u,v,z)]
   subtracts a star either from the current locn,
   or from the one specified

EFF
  treate the stack in an effective manner
INS
  treate the stack in an instrumental manner
     ie, do a forward-modeling deconvolution
STACKNIT=20
  number of forward-model stack iterations to do
FITPS-
  turn off(-) or on(+) the PS fitting during the
     stacking procedure
MAKE_STACK
  generates the stack using current parameters
STACKR[(NAME)]
  output the regular stack
     (correlated pixels)
STACKI[(NAME)]
  output the independent stack
     (ucorrelated pixels)

HIFLAG=60000
LOFLAG=-25
STOP
   halt program
PAUSE
   pause program
SKIP
   go to next arg