Argus PIXel Toolbox

John Stanley, Rob Holman
Coastal Imaging Lab, Oregon State University

03/06/2017

1 Concept

The Argus PIXel toolbox is based on the same concept that any field experiment is. Packages of instruments are deployed, where each instrument may have one or more sensors. The same ideas appear in this toolbox. Individual pixel locations (sensors) are grouped into instruments, which are further grouped into packages. It is this package which is then deployed into the field for collection of data.

Functions are also included to provide the basis of an instrument database, although final details of this database have not been implemented.

2 Usage

The functions will be described in the same sequence as they might be used to create a package. Unlike normal Matlab help, the functions will be presented with correct capitalization.

2.1 PIXForget

PIXForget is used to clear any uncommitted instruments or packages from the workspace. This is accomplished by reloading the data for the station being worked on. If there is no stored data, this effectively removes all information about packages and instruments previously created.

```
PIXForget;
```

2.2 PIXSetStation

PIXSetStation indicates which station these packages will be used at. This is the station name as found in the data files. This information is important so that the correct camera information can be retrieved later.

```
PIXSetStation('tinmuth');
```

2.3 PIXCreateInstrument

While packages can be built from the top down (i.e, create the package, add instruments, and then add pixels to the instruments), it is easier to create them from the "middle out". By creating the instruments first you can then make one call to create a package which uses them.

The function to create an instrument is PIXCreateInstrument. Each instrument is given a name (which can be used when processing the produced stack data to retrieve the included pixels by name), a type, and optional flags that control the conversion of x,y,z real-world coordinates into U,V image coordinates. These flags will be described later. The function returns an id, which is used with other functions to refer to this instrument. E.g.,

```
iid1 = PIXCreateInstrument('mBW', ...
                           'matrix', ...
                           PIXInterpUV );
```

2.3.1 Instrument Naming and Typing Convention

The toolbox puts no limit on the names or types of instruments you may create. These strings are for your use. However, in the interests of global harmony and understanding, we will share the convention that is being adopted by the CIL for names and types.

Type	Use	Name prefix
'matrix'	cBathy	mBW
'vbar'	surface currents	vbar
'runup'	wave runup transect	r

Names are based on the type of instrument, and include an indication of location. The names are created by appending the location to the prefix for the type. For example, a 'vbar' instrument that samples a line near x=150m would be named 'vbar150'.

These names are being used in the CIL file naming system when results from stack processing are stored. The results from a vbar instrument at x=304 at the Scripps site would be stored in a file named:

2.4 Adding coordinates to an instrument

There are three ways of adding real-world coordinates to an instrument. You can add individual x,y,z triples (or arrays of them), you can define a line in x,y,z space, or you can define a regularly-spaced two-dimensional set of x,y,z points (called a 'Matrix'). Notice that these are real-world coordinates, in meters, in the station coordinate system.

2.4.1 PIXAddPoints

To add individual points, or arrays of points, to an instrument, use the function PIXAddPoints. The parameters to this function are the instrument id, an array of x, an array of y, and an array of z, and an optional array of names for each point. The arrays must be of the same length. The function returns an id for this set of coordinates, which you may typically ignore.

For example,

```
    PIXAddPoints( iid1, [1 2 3], [1 2 3], [-1 0 1] );
```

adds the three points (1,1,-1), (2,2,0), and (3,3,1) to the instrument created above.

2.4.2 PIXAddLine

To add a line of regularly spaced coordinates to an instrument, use the function PIXAddLine. This function requires the instrument id, the x and y coordinates of the ends of the line, the z value for the line, the spacing, and an optional name for the line. E.g.,

```
    PIXAddLine( iid1, 0, 0, 100, 0, -1, 2 );
```

creates a line of coordinates from (0,0) to (100,0) at an elevation of -1, every 2 meters and adds it to the instrument created above.

2.4.3 PIXAddMatrix

A 'matrix' is a regularly-spaced two-dimensional set of coordinates. To add a matrix to an instrument, you need to provide the instrument id, the corners of the real-world rectangle, the elevation, and the x and y spacings. E.g.,

```
    PIXAddMatrix( iid1, 0, 0, 100, 100, -1, 1, 2 );
```

creates a set of points from (0,0) to (100,100) with a point every 1m in x and every 2m in y. The elevation of these points is set to -1.

2.5 PIXCreatePackage

Once you have instruments with coordinates of pixel sensors, you can easily create a package. The function PIXCreatePackage takes a string which names the package, and an array of previously created instrument ids. You can also provide an array of names for the instruments if you wish to override the names previously given to them in the PIXCreateInstrument call. This function returns an id which is used in following functions to build and schedule the package. E.g.,

```
    pid = PIXCreatePackage( 'tinmuth', [iid1 iid2 iid3] );
```

creates a package consisting of the three instruments with the ids iid1, iid2, and iid3.

2.6 Instrument Flags

Now that we've described what the coordinates are, it is time to explain how they are used in creating the UV pixel coordinates when the data are actually collected. There are three flags that can be used in the PIXCreateInstrument call. These flags apply to all coordinates that make up the instrument. Note that you can create a package with instruments that have different flag values.

2.6.1 PIXFixedZ

This flag instructs the toolbox to use the z value initially stored with the instrument instead of the tidal z value that will be provided when the package is scheduled. This is useful when you are defining an instrument that will sample a non-water-suface point. For example, if you are sampling the visual intensity of one of the ground-based GCPs, you do not want the z value to fluctuate with the incoming or outgoing tides.

2.6.2 PIXFixedXY

When an x,y,z coordinate is converted to an image UV coordinate, normally the UV value is determined from the x, y, and z, then truncated to an integer, and the x, y, and z back-calculated for that UV integer value. (This is because the sample program at the remote end can only sample integer pixel locations.) This simplifies the process greatly if you don't care exactly where the pixels come from, as long as they come from somewhere close to the specified location. E.g., a wave directional array where the processing routine can use arbitrary x,y locations.

When you do not want this conversion to take place, you can specify PIX-FixedXY as one of the flags to the PIXCreateInstrument function. The collection routine will still convert the UV coordinate to integer prior to collection, but the back-calculation will not take place. This flag is not commonly used.

2.6.3 PIXInterpUV

As was previously noted, the collection routine that actually samples the video data works only with integer UV locations. This causes artifacts to appear in, e.g., linear arrays of pixels, as the line steps from one U to the next (in a nearly-vertical line), or one V to the next in a nearly-horizontal line. This effect can be minimized by using the four surrounding pixels and doing a weighted-average to simulate non-integer pixel locations. That's what the PIXInterpUV flag tells the toolbox to do.

When you use the PIXInterpUV flag, the integer location of the four pixels surrounding the desired point will be collected, and the PIXInterp function can then be used on the returned data to do the weighted-average interpolation.

Because this flag allows non-integer UV pairs to be collected, the back-calculation of x and y are not performed. In effect, this flag supercedes the need for PIX-FixedXY.

NOTE: this flag causes approximately four times the number of pixels to be collected.

2.6.4 Using Multiple Flags

The flag values may either be added or 'or'd to create the final flag value. E.g.,

```
    iid2=PIXCreateInstrument('mBW', ...
                             'matrix',...
                             PIXFixedXY+PIXInterpUV);
```

2.7 PIXBuildCollect

NOTE: If you are using the UAV/No Database version of the PIXel Toolbox, go to Section 3!

Now that you have created a package with instruments that have defined xyz coordinates, you can “build” the collection. This is the step that actually expands the coordinates you have defined, converts the real-world points to UV coordinates, tags those with the instrument names, and allocates the pixels to the appropriate cameras.

The function requires the package id, the epoch time intended for the collection, the tidal elevation, and a hint for the priority of cameras. It returns a structure that contains all the data about the package, the coordinates, the cameras, the pixels; in other words, just about everything.

```
r = PIXBuildCollect( pid, epoch, tide, [2 3 1 4 5] );
```

This function will build the collection structure ’r’ using the package with id ’pid’, at epoch time ’epoch’, with a tidal z of ’tide’, with camera 2 given highest priority and camera 5 the lowest. Remember, however, that any instrument that has been created with the PIXFixedZ flag will not use the tidal z value here, but will use the z values of the points, lines or matrices that are part of that instrument.

2.7.1 What is “priority”?

Camera views almost always overlap. When they do, there will be two cameras that ’see’ certain x,y,z locations. How should the software decide which camera to use to collect any points in those overlapping regions? It uses the priority that you give in the call to PIXBuildCollect.

In the example above, the cameras were listed explicitely. Any points that appear in both cameras 2 and 4 will be collected using camera 2, since it appears before camera 4 in the list. You can specify the string ’fov’ to give cameras priority by field of view, which means that any zoomed-in camera has priority over one that is not zoomed-in. This is the default if only three parameters are passed to PIXBuildCollect.

A special case is when you want to collect all pixels, even from overlapping views. Setting the priority to ’none’ is how you tell PIXBuildCollect to do that.

2.8 Scheduling the Collection

Even though you have specified a time for the collection to take place in PIXBuildCollect, you have not yet scheduled the collection to take place. This has not yet happened for several reasons. First, PIXBuildCollect allows the user to experiment with different instruments and settings prior to actually causing any collection to take place. Second, it is possible to create packages that are time-independent, so even though a time has been specified, it may not be the actual time the collection will be performed.

To actually schedule a collection that has been built, use the function PIXScheduleCollectIII. This function converts the 'r' collection structure into pixel lists and the commands needed by the remote station to actually do the collection. The only remaining bits of information required are the number of samples to be collected and the frequency.

```
    PIXScheduleCollectIII( [3 4], 1024, 2, r );
```

This call schedules a collection using cameras 3 and 4, for 1024 samples, at 2 Hertz, using the struct built by PIXBuildCollect. Keep in mind that the maximum frequency for two cameras is 2 Hz, because the software does remember and will beep at you if you exceed that rate.

The files produced by PIXScheduleCollectIII are created in a 'temp' directory underneath the current working directory. E.g., if you are in your home directory (˜), you will find your output files in ˜/temp. These will all start with the epoch time for the collection. The pixel lists will have the extension '.pix'. The commands to the collection software are in the file with the extension '.sched'. The '.mat' file is a copy of the collection struct (r), which will be required for processing the stack data once it has been recovered from the station.

At this point, the .pix and .sched files need to be transferred to the remote station. An automated system has not yet been produced for this since almost every station requires a different means of transporting them. The .mat file does not need to be sent to the Argus station, but it does need to be copied to a location so that the stack processing routines know how to find it. Currently, this is the directory 'collects' located under the station data archive directory. E.g., '/ftp/pub/tinmuth/collects'.

2.9 Fixing an Old Collection

Once you have built the collection for a set of instruments, you have effectively locked yourself into locations of pixels. If, for example, the camera shifts over time, then the 'r' array that you created will contain the wrong real-world coordinates (compared to the UV coords), or the wrong UV coords for the real-world points you want to sample. If you are processing a stack and want to know where the pixels you collected actually lie, you need to use a CURRENT geometry (not the one stored in the 'r' structure) and FindXYZ to back-calculate the real x and y locations associated with the UV coordinates you have.

However, suppose you have an 'r' structure from an old collection and want to reschedule the same collection using updated geometries. The best solution is to simply rerun the original instrument construction code. But, alas, the graduate student who wrote the original program to create the instruments has graduated (or dropped out) and the original code is no longer available. The functions 'PIXPrepareR' and 'PIXRebuildCollect' were created to deal with this problem. Normally, PIXBuildCollect gathers all the instrument data you have created and packs it into an 'r' structure, and then converts the real world space into image coordinates. PIXPrepareR takes an 'r' stucture and inserts the current camera, geometry, and IP data for the epoch time recorded in 'r'.[1] PIXRebuildCollect takes that updated 'r' structure (containing xyz data) and replaces the old conversions. To reschedule the collection, simply pass the new 'r' structure to PIXScheduleCollectIII and new pixel list files (and a new .mat with the new 'r') will be created.

3 Using the PIXel Toolbox Without an Argus Database

The PIXel toolbox has been designed so that the station-specific data required to convert real-world coordinates into image UV pairs is localized in one function: PIXPrepareR. If you wish to create a pixel list for images coming from cameras that are not managed in the CIL/argus database, then all you need to do is provide the relevant information in similar format and insert that into the 'r' structure prior to actually building the collection. 'PIXBuildCollect' calls the three essential functions assuming the CIL/argus database, but you can call each in your pixel array creation function.[2]

You may use any of the PIXel toolbox functions already described to create and modify instruments. Instead of calling PIXBuildCollect to populate 'r' with data, you will want to use the three following functions.

3.1 PIXCreateR

PIXCreateR requires as input the same parameters that PIXBuildCollect does: the package ID, the epoch time, the tide level, and the camera priority (or "sortBy"). It produces a basic 'r' struct with all of the instruments expanded into individual x,y,z locations.

```
    r = PIXCreateR( pid, epoch, tide, 'fov');
```

[1]NOTE: when you are rebuilding an existing 'r' structure, you MUST change the value of r.epoch to represent the epoch time you wish the 'r' rebuilt for. Otherwise you will simply get the same old, wrong geometry data.

[2]This discussion assumes that you are using the PIXel toolbox to create UV pairs but will not be creating a stack file in the standard CIL format (either Sun rasterfile or NetCDF). Thus the stack loading and manipulation tools will not be used, and have not yet been sanitized of database calls.

3.2 Parameterization

Before the U/V pairs can be calculated, the camera, geometry, and image processor data must be inserted into the 'r' stucture. If you are not using the CIL database, then you are responsible for creating this data in the correct format. These are the necessary structures:

3.2.1 cams

This is an array of structures containing the data about your camera(s). Each structure contains the following fields:

Field	Contents
cameraNumber	The camera number for this camera. You can use any camera numbering system you wish, but the numbers here must match any numbers you use in the priority list.
x	The x location for the camera
y	The y location for the camera
z	The z location for the camera
D1 and D2	The D1 and D2 distortion parameters for the camera/lens pair
K, Drad, D_U0, D_V0	The system intrinsic parameters K, the radial distortion vector Drad, and the image center
lcp	'lcp' structure from UAV processing, using Caltech distort/undistort

The first four fields should be obvious. One of the latter three sets is required to perform a 'distort' operation on the U/V pairs. Use whichever system you have data for.

3.2.2 geoms

An array of geometry data for the cameras, in the same order as the cams array has them. The fields required are:

Field	Contents
m	The 'm' vector for the geometry
azimuth	The azimuth from the geometry solution
fov	The field of view of this camea

The 'fov' value is used if you select 'fov' as the priority, and both fov and azimuth are used in a calculation to help prevent back-mapping of x,y,z points improperly into UV space. I.e., points from behind the camera should not map into the image. This is done by calculating where the horizon is in the image.

If you cannot provide fov (e.g. you are working with Caltech calibrations where fov is not an angle, it is in units of pixels) then set fov to -1 and the horizon mapping will not be used.

3.2.3 ips

This is an array of structures containing basic image processor details for the cameras. Fields required are:

Field	Contents
width	Width of the image
height	Height of the image
U0/V0	Fields containing the image center. U0 is negative.
lx/ly	The scaling from real pixel space to 640/480 space

Note: the last four fields (U0/V0/lx/ly) are required only if the D1/D2 distortion system is being used.

3.2.4 PIXParameterizeR

To insert the geometry and camera data into the 'r' struct, you may use the following call:

```
r = PIXParameterizeR( r, cams, geoms, ips );
```

3.2.5 PIXDataRToInsts

Assuming you have collected your data from multiple frames as an NxM array, where N is the number of frames (samples) and M is the number if U/V pairs, and you have the 'r' struct that was used to create the U/V pairs, then you can deconstruct the single data array into individual instruments using the function 'PIXDataRToInsts'. This function currently assumes a ONE camera collection, and thus refers only to r.cams(1) for name, U/V, and xyz information.

Also, since there is no more accurate realtime information, the epoch time of the data is calculated assuming that the collection starts at r.epoch and has a 2Hz (0.5 sec) collection rate. If you use a different start time, you can modify r.epoch prior to calling PIXDataRToInsts. If you use a different collection rate, you will need to modify the output of this function prior to further use.

```
stackOut = PIXDataRToInsts( data, r );
```

The result 'stackOut' is an array of structs where each struct contains the field 'name' containing the instrument name, and 'data' which has the following fields:

Field	Contents
RAW	NxP raw pixel data
CORRECTED	NxP corrected pixel data
T	Nx1 array of epoch time for each sample
CAM	Px1 array of camera number
XYZ	Px3 array of pixel X/Y/Z

where N is the number of sampled frames and P is the number of pixels in that instrument. Note: since there is no correction data (gain and shutter) the CORRECTED data will be a duplicate of RAW, and is present to maintain compatibility with other CIL or user functions that may require one or both.

3.3 PIXRebuildCollect

The final step in creating a full 'r' structure with U/V information properly doled out to each camera is to call 'PIXRebuildCollect'. This function uses the x,y,z data expanded by PIXCreateR with the geometry data from PIXParameterizeR to convert all x,y,z data into UV space, distort it into the camera view, and then allocate pixels that are in each view to the appropriate camera.

```
r = PIXRebuildCollect( r );
```

At this point you may now call PIXScheduleCollectIII to produce the pixel lists for each camera, or you may access the UV data within the 'r' struct directly.

4 "I have points left after assigning them to all cameras."

You may see this message appear when you build a collection. It means, simply, that some of the x,y,z coordinants you have defined in the instruments did not appear in any of the cameras available. This may happen for any number of reasons. Coordinates near the edges of an image may move outside camera range because the tidal elevation changes. You may have deliberately designed a transect (runup or bathymetry line, e.g.) so that it always extends off the edge of the image. Or you may have accidentally created such a line.

If you see this message and you don't think any of your points should be outside camera range, you should plot the UV points for each camera on an image from that camera to see what might be extending off the edge. See the section describing the 'r' structure for information on how to find the UV pixel coordinates for a camera.

5 A Sample Program

This is a sample program demonstrating the use of the PIXel toolbox.

```
%%%%%%%%%%%%  PART I - Design of pixel instruments  %%%%%%%%%%%%
PIXForget;                      % clear any old local information
sitenm = 'argus02b';            % pick a station now.
PIXSetStation(sitenm);
zmsl = 0.0;             % difference of msl from NGVD

%% create a large matrix coverage for cBathy
x1 = 80;   x2 = 800;  dx = 5.0;
y1 = -500; y2 = 1500; dy = 10;

idb=PIXCreateInstrument('mBW','matrix', PIXFixedZ+PIXDeBayerStack);
PIXAddMatrix(idb,x1, y1, x2, y2, zmsl, dx, dy);

%% create runup timestacks
y_runup_locs=[650:50:1100 945];
xshore = [160]; xdune = [0];
rot =0;
dx = 0.1;
for i = 1: length(y_runup_locs)
 [xr, yr, zr] = runupPIXArray(xshore,xdune,y_runup_locs(i),zmsl,rot);
 instName = ['runup' num2str(fix(y_runup_locs(i)))];
 idr(i) = PIXCreateInstrument(instName,'runup',PIXFixedZ+PIXUniqueOnly);
 PIXAddPoints(idr(i), xr, yr, zr, instName );
end

%% create some new Vbar arrays
yVBarMinMax = [0 1500];
xVBar = [125:25:225];
for i = 1: length(xVBar)
 instName = ['vbar' num2str(xVBar(i))];
 idv(i) = PIXMakeVBAR(xVBar(i), yVBarMinMax(1), ...
                      xVBar(i), yVBarMinMax(2), zmsl, instName);
end

%% %%%%%%%%%%  PART II - Create and Schedule colletion  %%%%%%%%%%%%
idPack = PIXCreatePackage('DuckArray_20150226', [idb idr idv]);
etime = matlab2Epoch(now+8/24);
cams=[1 6 2 5 3 4];
r = PIXBuildCollect(idPack, etime, zmsl, cams);
err = PIXScheduleCollectIII(cams, 2048, 2.0, r);
```

5.1 What is this 'r'?

A good question. It is sometimes referred to as the "collection structure". Another good question is why it is called 'r'. It's called 'r' for the same reason that the SGI data collection program is called 'm' (and ArgusIII is 'hotm').[3]

What is it? It is a structure of structures that contains almost everything about the stack collection. This is an example from the Scripps station:

```
r =
pid:      1
epoch:    1.002126162000000e+09
tide:     0.19000000000000
sortBy:   [0 1]
station:  'argus04'
cams:     [2x1 struct]
geoms:    [1x2 struct]
origx:    [1x1382 double]
origy:    [1x1382 double]
origz:    [1x1382 double]
names:    {1x1382 cell}
f:        [1x1382 double]
types:    {1x1382 cell}
```

Most of the entries are self-explanatory. The 'cams' array is an array of camera structures (as found in the database) with data specific to the camera added. The raw U, V, name and flag data are included, as are the final U, V, names and types for each UV pair collected for that camera. The cams struct also contains the X, Y, and Z data for the raw UV coordinates. If these arrays are empty, then there were no UV pairs calculated for this camera. Note carefully that the cameras are NOT guaranteed to be in camera number order. I.e., r.cams(1) is not guaranteed to be camera 1.

6 Processing stack data

Because the PIXel toolbox has used a standard set of names for the pixel list files and matlab collection struct storage, it can extract the pixel list name from the stack datafile itself and locate the information about the instruments contained therein. This makes pixel data processing much simpler.

[3]Because John wanted a short name to use while debugging the code during initial testing and never changed it.

6.1 Showing PIXel instruments

A convenient routine for showing the location of pixel instruments on a camera image has been created. showPIXInstruments will take a stack file name and produce an image from the corresponding snapshot with the instrument locations plotted. This image will be in oblique mode. E.g.,

```
showPIXInstruments('1000000000.xxx');
```

6.2 loadStack

"loadStack" is the Matlab interface into the Sun rasterfile format stack data. That function knows the encoding for the data contained in the rasterfile, which is much more than just the pixel values. Also included in the stack file are the UV coordinates themselves, the sample rate, the name of the pixel file, and time information about each sample.

There are several functional forms of loadStack. The simplest form is:

```
params = loadStack( FILENAME, 'info' );
```

This call returns the parameter structure for the requested stack. This structure contains information about how the stack was collected, with the pixel file (also known as the "Area Of Interest" or AOI file). This filename is used to find the collection structure file.

Here is the parameter struct from a typical stack:

```
p =
magic:      1.5041e+09
width:      1534
height:     2056
pixels:     1526
when:       1.0263e+09
camera:     1
version:    4
increment:  15
isColor:    0
lines:      2048
where:      'argus04'
ust:        6.4871e+06
aoifile:    '1003508562.c1.pix'
syncTime:   1.0263e+09
U:          [1526x1 double]
V:          [1526x1 double]
order:      [1526x1 double]
```

The important values in that struct are the 'pixels' (number of UV's per line), the 'lines' (number of images sampled), the 'increment' (30/increment is the sample frequency), and the aoifile.

The second most useful version of loadStack is:

```
[p, epoch, msc, data] = loadStack( filename, UV );
```

This returns the parameter struct 'p', an array of epoch times (one per stack line), an array of "media system counters" (basically, a frame counter), and a 2-D array of pixel intensities. each column in "data" corresponds to one UV coordinate as listed in the Nx2 UV array.

Further information on loadStack can be found by using "help loadStack".

An important note about 'loadStack': it can be VERY slow. To make processing faster, the function has been written to cache the last file it reads. That is, if you ask for the same file twice, the first request will actually read the file from the disk, the second will read it from an in-memory copy. If you are processing several stack files, you should keep this in mind and try to access all the data you will want from one stack file before you try to load the next.

Under Argus III, 'loadStack' output has been modified somewhat so that it returns camera gain and shutter values (in real-world quantities) if possible. "help loadStack" for more details specific to the version you have.

6.3 PIXGetRFromAOIName

This function takes the station name (e.g., 'argus04') and the AOI filename (from loadStack), and finds the stored collection structure that produced that AOI file. Note that multiple stacks can be produced from the same AOI file.

Once you have retrieved the 'r' structure for a stack by getting the AOI filename from the stack and then the R for that AOI file, you can use the names in that struct to find UV coordinates, and pass those UV values to loadStack to retrieve your data.

While creating stack lists is now much simpler, there is a tradeoff.[4]You must be careful to note that the order of the pixels in the pixel list may no longer match the order you expect them to appear in. It is quite likely that a runup transect you have defined as one instrument will be interspersed in the stack with pixels from other instruments. This is a side-effect of collecting only the unique UV pixels in the full list, which is a side-effect of allowing people to define a line of pixels with a spacing of 1 cm. The goal is to reduce the size of the stack by eliminating duplicate columns. The result is that you must keep track of the actual order of the UV pixels. Once you have selected all the UV coordinates

[4]There is always a tradeoff.

that coorespond to a runup transect ('r123', for example), a simple sort based on V will most likely return the array of pixel values to the sequence you would expect.

The function 'showPIXInstruments' contains an example of the use of loadStack and PIXGetRFromAOIName.

6.4 PIXFindUVByName

If you know the name of the instrument you are seeking, this function will retrieve the U and V coordinates for those instruments. It will also optionally retrieve the raw U and V and the x, y, z coordinates. The U and V coordinates can then be passed to loadStack to retrieve the intensity values.

What is the difference between the U and V, and the raw U and V, you ask? The raw version is the double value, while the U and V are the integers. The integer values are what the remote system actually collects. The raw values are the idea coordinates.

6.5 PIXInterp

"PIXInterp" performs a weighted average of the sampled pixel intensity to reduce the aliasing ('stairstep') problem mentioned earlier. It takes the intensity data produced by loadStack and the rawUV information from PIXFindUVByName.

6.6 A Special Note on Interpolation

You probably recall the previous discussion of the PIXInterpUV collection flag. This flag tells the PIXel toolbox to sample the four pixels surrounding the actual physical location. Thus, when you retrieve the UV coordinates for an instrument that has this flag set, you will get four times as many UV coordinates as you would expect. The function PIXInterp takes the quadruplicate data set and reduces it to the interpolated values. In order to do this, it requires the floating point UV coordinates (rawUV mentioned above).

This function requires that NO sorting of the coordinants take place prior to interpolation. It expects the intensity data it works with to be in a specific order, and any sorting of the data by UV or xyz location will destroy this order. Sort the resulting data only after PIXInterp.

7 Further Routines

This toolbox is in the user-driven stage. The functions necessary to create pixel lists have been written, and basic data access routines are available. Now I need users to determine what else they need; what other functions should be and would be useful. Those will be added. There are other routines in the toolbox, most of which are used internally. The function of the remainder may be ascertained from the souce.