Analyzing the Development of Bad Pixels on Detectors in the Spartan IR Camera

AST 310: Directed Studies, Professor Ed Loh Fall 2007

Nathan Sanders

Department of Physics & Astronomy
Michigan State University, East Lansing, MI 48824

sande253@msu.edu

28 September 2007

Abstract

A process related to the routine cooling of the detectors on the Spartan Infrared Camera was believed to cause the development of bad pixels on the detectors. A normalized threshold method was developed using the Image Reduction and Analysis Facility (IRAF) to identify the development of bad pixels on the detectors over time. An automated system was developed for applying this method to arbitrary data sets to assist in tracking the development of bad pixels. The method was applied to images of three detectors taken with the Spartan Infrared Camera in March and June of 2007.

The hypothesis that corner regions of detectors are particularly susceptible to bad pixels produced by detector cooling due to their large distance from the detector center was confirmed for two quadrants. The first such corner region, on quadrant 1 of detector SN92, was found to be relatively light in bad pixel count (200 bad pixels, as of June 2007), but growing at a fast pace (23% following one cooling). The second corner region, in quadrant 0 of detector SN66, was found to have a relatively high count of bad pixels (1383 as of June 2007), but growing at a rate that may be negligable (1.5% following one cooling). This indicates that the growth rate of the corner regions may be proportional to the present bad pixel count of the region.

Isolated bad pixel regions were also analyzed, and found to be insignificant in magnitude but growing at a significant rate. From a four megapixel array, detector SN66 was found to have only 280 bad pixels as of June 2007, but the bad pixel count increased by 34% following one cooling. Findings were similar for the two other detectors analyzed (SN24: 664 bad pixels, 24% increase, SN92: 200 bad pixels, 29% increase).

1 Background

It has been well known that cooling the detectors to operating temperature and subsequently warming them to room temperature would cause bad pixel formation. The cooling causes differential contraction between the Si and HgCdTe materials which constitute the instrument. This process should introduce isolated bad pixels throughout the detector and a systematic pattern of bad pixels at the corners of the detector quad. The corner regions should be affected more strongly because their distance from the center of the detector is greatest and differences in contraction thus have the largest effect. Furthermore, upon each subsequent cooling, it was expected that the bad pixel corner regions would grow in toward the center of the image. A statistical understanding of this phenomenon would be helpful in anticipating the bad pixel development caused by instrument cooldown.

Two image sets from the Spartan Infrared Camera on the SOAR telescope, from March and June of 2007, were analyzed for this paper. The earlier set was recorded at LST 2007-03-05T11:47:31.460 and is referred to as the "March data", while the later "June data" was recorded at LST 2007-06-07T13:46:59.639. A detector cooling occurred between these dates.

The four detectors of the Spartan Infrared Camera are each divided into four quadrants. The detectors and quadrants are each identified by numbers from 0 to 3, in the format "d0q0" for "detector 0 quadrant 0." The detector identification numbers correlate to the serial numbers of each detector currently installed on the instrument as indicated below:

Detector Identification Number	Serial Number
0	24
1	92
2	97
3	66

To demonstrate the effect of the bad pixel phenomenon, sample images for each detector from the June data appear in Figure 1. The bad pixels appear as conspicuously dark regions in the image. In these images, it is possible to readily identify corner regions where bad pixels have developed. It is also possible, though more difficult, to visually identify isolated bad pixel regions throughout each detector.

In addition to dark, bad pixel regions on the detector images, bright regions have also been observed. The bright regions are often associated with borders of bad pixels, which often become more pronounced after coolings. The cause of these bright-region-associated bad pixels is not necessarily thought to be related to the cooling process, unlike the development of other bad pixels. For the purposes of this paper, the bright

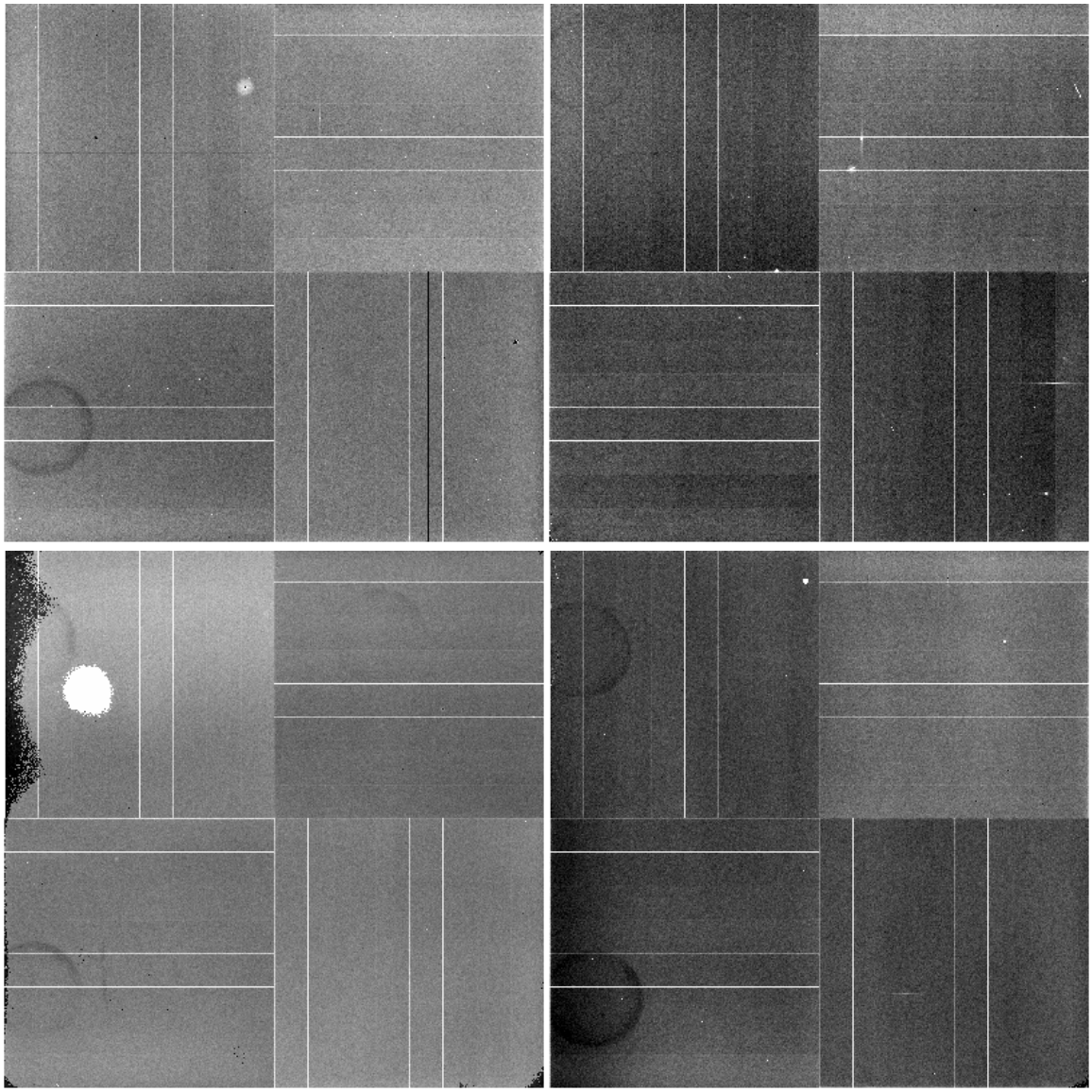

Figure 1: Top: from left to right, unaltered June images for detectors 0 and 1. Bottom: from left to right, unaltered June images for detectors 2 and 3. Dark regions such as the lower-left corner of d1 (d1q1) are visually identifiable as bad pixels.

regions will not be considered. No attempt will be made to distinguish any bright region-associated bad pixels from other bad pixels.

2 Characterization

The bad pixels are easily identifiable by an extreme difference in pixel value between the bad and adjacent pixels. This distinction is most visible in the “short images” produced by the detectors, as opposed to the “long - short images” that serve as the final detector output. The short images are therefore used in bad pixel detection.

Because there is such a drastic difference between the pixel value of bad and unaffected pixels, it is convenient to use a threshold method for detecting the bad pixels. By this method, all pixels recording values beneath a manually defined threshold value are considered bad pixels.

However, the images must be normalized before a threshold can be defined because the pixel amplitude is not constant over entire detectors, or even entire quadrants. An unusually dark pixel in one region of a detector image may actually be brighter than the average pixel in another region. Normalization should establish a common threshold for a maximum amount of bad pixels on the image. Normalization is done by subtracting an individualized constant from each pixel in an image, where the constant is the median of the pixels within a predefined area around the pixel. In order to avoid large concentrations of bad or bright pixels skewing the median value, very bright and dark pixels are thrown out of the regional median set. Furthermore, the region size must be large enough that at least one good pixel will be included, even if the target pixel is in the center of a large bad or bright pixel region. As demonstrated in Figure 2, this method is very effective at normalizing large-scale brightness differences over a detector image, but not at reducing localized image noise from pixel to pixel. It is sufficient to normalize the image such that a single bad pixel threshold can be established for entire detector images.

Bad pixel threshold values can be easily established by looking at the histogram of pixel values for normalized images. The ds9 image viewing program can display such histograms. The largest peak in the histogram represents the mode value of the image, approximately 0 for a normalized image. Peaks at greater values than the mode represent concentrations of bright pixels of various values. Peaks lower than the mode value represent concentrations of bad pixels. There will generally be several bad pixel peaks, some of which represent distinct concentrations of bad pixels caused by electrical problems which produce entire rows or columns of bad pixels in the image. One such histogram is reproduced in Figure 3. An appropriate threshold value has been defined as one centered in the trough between the peak at the mode and the first bad pixel peak.

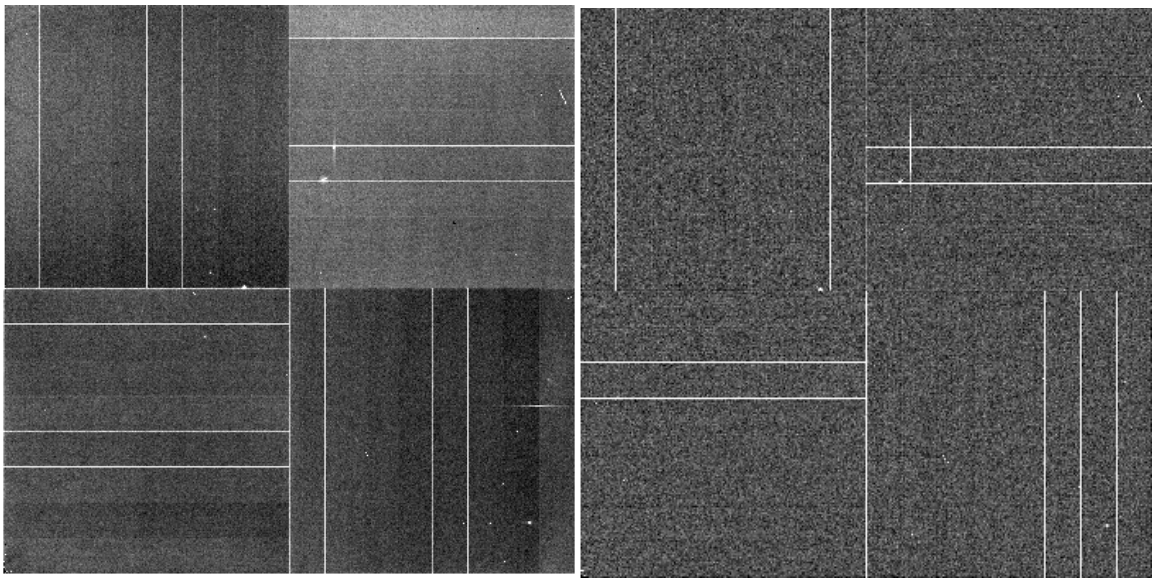

Figure 2: Left: the unaltered d1 image from the June dataset. Right: the normalized image. In the normalized image, there are no longer regional brightness variations in the image, while the appearance of bad pixels is unchanged. (Note that the uneven appearance and disappearance of white lines, which normally appear at even 128 pixel intervals in each quadrant, is a product of the image resizing for this paper rather than the normalization method)

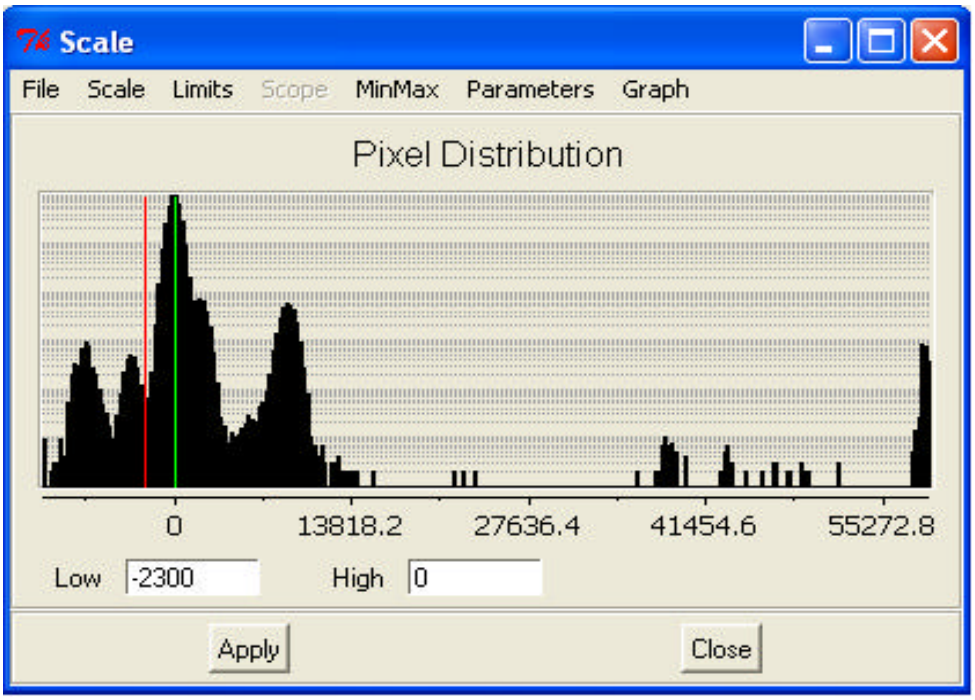

Figure 3: A logarithmic pixel value histogram produced with ds9 for the normalized June d0 image. Note the mode peak at 0 and two bad pixel peaks at negative values. The threshold for this image was defined between the mode peak and the first bad pixel peak, at -2300.

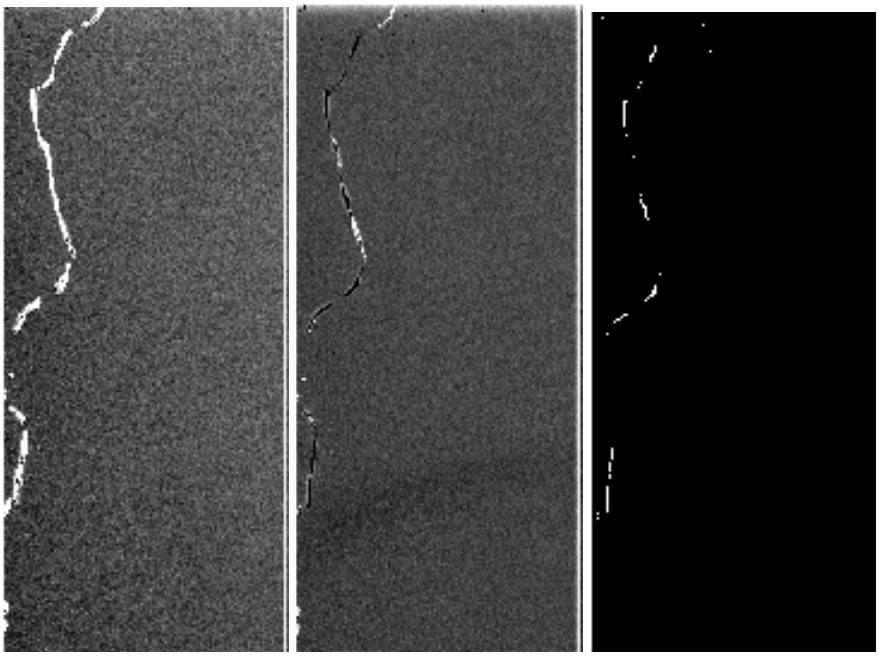

Figure 4: From left to right, the corner region of d3q2 in the unaltered March data, unaltered June data, and in a difference mask produced from those two images. From its appearance in the unaltered data, the string-like feature does not seem like it could be caused by the cooling process. However, it appears as a new bad pixel region in the difference mask. It is thus justifiable to subtract this region from the difference mask.

Image masks are made to represent the location of bad pixels on detector images. When considering the development of bad pixels, the distribution is significant in addition to the count. An awareness of the location of new bad pixels is necessary to test the hypothesis that the bad corner regions are growing outward toward the center. The mask is made by assigning a value on one to pixels that are defined as “bad” by the threshold method and assigning a value of zero to good pixels. Two detector masks from different data sets can be subtracted to produce a difference mask. The number of bad pixels on the difference mask can then be counted to produce a count of the bad pixels which developed between the capture dates of the data sets.

Before summing the pixels on a difference mask to develop a count of new bad pixels, it may be necessary to remove “lost” pixels so that they do not subtract from the new bad pixel count. The difference mask has values of 0 where no bad pixels appear in either image or where bad pixels appear in the same location in both images. The difference mask has values of 1 or -1 where bad pixels appear in one mask, but not the other.

Assuming the earlier mask is subtracted from the later mask, the -1 pixels represent bad pixels that seem to have disappeared between the capture dates. These cases are referred to as “lost” pixels. The removal of lost pixels is justified by observation which indicates that lost pixels do not necessarily represent a phenomenon related to bad pixels produced by cooling and, in any case, occur very rarely, generally only one or two times per detector image. Isolated lost pixels have been observed to be related to both errors in the threshold (when a pixel narrowly escapes being detected by one threshold, but not the other) and uncertainty in the equipment itself (when a bad pixel in the earlier data appears to have become a good pixel in the later data, which is not thought to be possible).

Image defects not relating to the development of bad pixels due to detector cooling should also be removed from the difference mask before summing. Equipment defects unrelated to the cooling process may produce very dark regions that are inadvertently identified as bad pixels in the threshold masks, as illustrated in Figure 4. Bad pixel lines caused by electrical problems are also not related to the cooling process and should be removed from the mask.

Also to be removed from the difference mask before summing are corner regions, which will be examined separately in §4.

3 Results for Isolated Bad Pixels

The March and June datasets were analyzed by the methods described in §2 and §5. During the recording of the March dataset a defective detector, SN74, was installed in channel 2. It is therefore not possible to compare channel two images between the two datasets.

Analysis of the difference masks produced for each detector demanded that some regions of each detector be disregarded, as justified in §2. Several image defect regions not related to the cooling process were removed from the difference mask before bad pixel counts were made. Bad corner regions were also removed and will be analyzed separately.

Table 1 lists the thresholds chosen for each detector.

The difference masks produced seem to be accurate representations of the new bad pixels. Several regions were visually checked to ensure accuracy. One such region is reproduced in Figure 5.

Figure 5 illustrates a particularly difficult case in which the normalized threshold method performs well, but not perfectly. While it is possible that a yet more sophisticated method may correctly identify the two pixels in that region which this method failed

Detector	March Threshold	June Threshold
0	-2500	-2300
1	-2300	-2500
3	-2500	-2700

Table 1: The parameters used to develop the isolated bad pixel counts for the March and June data. The given threshold values were applied to normalized detector images.

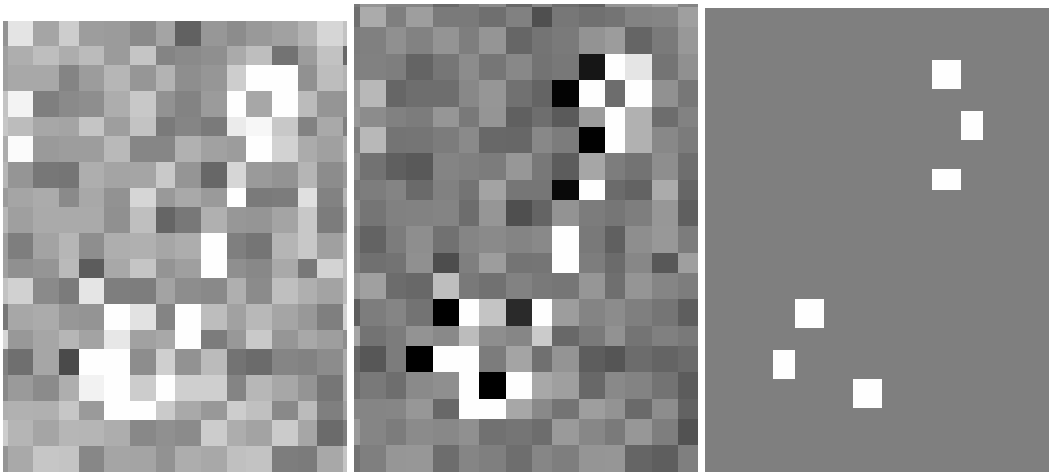

Figure 5: From left to right, region d0 [911:924,915:920] in the normalized March data, normalized June data, and difference mask. As expected, only the bad pixels that appear in June and not in March appear on the difference mask. However, two conspicuously dark (presumably bad) pixels on the normalized June mask were slightly above the June threshold and, therefore, do not appear on the difference mask.

to, the normalized threshold analysis of this region is clearly superior to an earlier trial of non-normalized threshold analysis. The earlier trial, in which the original detector image was not normalized before a threshold was picked, returned only one pixel in this region on the difference mask, because most of the bad pixels in the June image escaped the threshold.

With erroneous regions removed, bad pixel counts were made for the March and June threshold masks and the difference masks. It is useful to compare the bad pixels in the threshold and difference masks to determine the fractional change in bad pixels. These data are summarized in Table 2.

The data presented in Table 2 indicate that there is significant growth in the isolated bad pixels during coolings, while the absolute count of the bad pixels remains a very small fraction of the entire 2048x2048 detector pixel array. Further analysis of bad pixel

Detector	March	June	New in June
0	510	664	158 (24%)
1	143	200	58 (29%)
3	185	280	96 (34%)

Table 2: Isolated bad pixel counts for three detectors for March and June data. “New in June” was calculated using the difference mask.

development over time will be necessary to confirm whether these conclusions apply generally or are specific to the time period and data sets analyzed for this paper.

4 Results for Corner Bad Pixel Regions

The prediction that the corner regions of each detector would be affected most by the cooling phenomenon was confirmed. Only two of the twelve detector corners analyzed (in d1q1 and d3q0) demonstrated this prediction. However, the two affected corners exhibited the highest concentration of bad pixels found anywhere on the three detectors. Furthermore, as expected, it appears that the bad corner regions are growing in toward the center of the detector after coolings.

The corner regions of d1q1 and d3q0 were extracted from the March and June threshold masks and analyzed individually. Difference masks were made for these regions and are reproduced in Figure 6.

Statistics were compiled for the bad pixel growth in these corner regions, presented in Table 3.

Region	March	June	New in June
d1q1	165	200	45 (23%)
d3q0	1462	1383	21 (1.5%)

Table 3: Corner region bad pixel counts for three detectors for March and June data. “New in June” was calculated using the difference mask.

While lost pixels were not a factor when analyzing isolated bad pixels (each bad detector was found to have at most 2 lost bad pixels), they are significant to the analysis of the corner regions.

The lost bad pixels in d1q1 are strongly related to the bright pixel phenomenon. Most of the lost pixels in the difference mask were caused by March bad pixels turning into

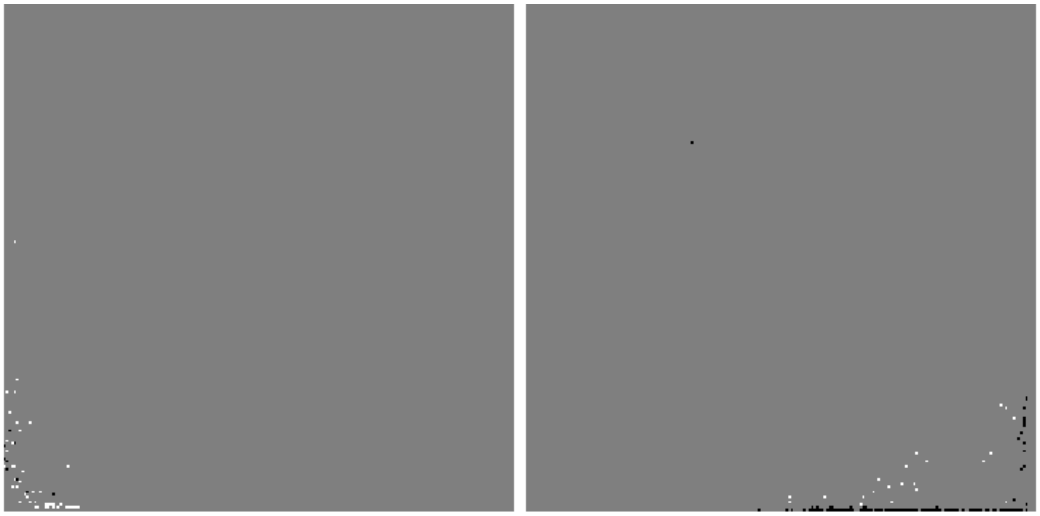

Figure 6: Left: the bad pixel difference masks for the corner of d1q1. Right: the corner of d3q0. Grey pixels indicate regions where no bad pixels existed in either dataset or where bad pixels existed in both datasets. Black pixels indicate lost pixels, white pixels indicate bad pixels new to the June dataset. Note that most of the white pixels occur at the inner edge of the corner regions, toward the center of the detectors.

bright pixels in the June data. The cause of this phenomenon will not be speculated upon here.

In d3q0, the lost bad pixels are concentrated on the edges of the corner and are almost undoubtedly erroneous. Most of the lost pixels appear slightly above the threshold value in the normalized June detector image, indicating a failure in the threshold procedure that may be related to the large size of the d3q0 bad corner region. This failure should not have a noticeable affect on the new pixel count given in Table 3. However, to find the percentage of new bad pixels, it may be more sensible to compare the new pixel count to the unaffected and more accurate March bad pixel count (with the confirmed new pixels subtracted) than the affected June count. This calculation yields approximately the same value as found previously: 1.5%.

It was verified that the normalization process was not inducing errors in the bad pixel detection technique for the corner region by manually comparing these results to an earlier trial where normalization was not performed. This comparison indicated that the normalized method was more accurate than the plain threshold method. There is no indication that the results in Table 3 may be significantly flawed.

It can be reasonably concluded that there was not a substantial growth in bad pixels in the d3q0 corner region between March and June, while there was substantial growth

of the d1q1 corner region. It is possible that the growth of the bad pixel corner region is related to its size: the already very large d3q0 region remains essentially unchanged, while the relatively small d1q1 region grows at a rapid pace.

5 Appendix: Applying IRAF

IRAF was installed and configured on a computer running the Microsoft Windows XP operating system. Because IRAF is native to Unix-like systems, it was necessary to install it on top of a Cygwin environment. The ds9 program running on Cygwin was used to visualize the IRAF data.

The analysis of bad pixels created by the cooling process begins with the preparation of detector short image samples from two dates in which a detector cooling occurred in between.

5.1 Stitching Detector Images

In some cases, an image set from a certain date will consist of 16 images (4 quadrant images for each of the 4 detectors) rather than 4 images (one for each detector). Because it is more convenient to analyze 4 images rather than 16, the quadrant images are first stitched into detector images. This was performed on the March 5th dataset analyzed for this project.

Before stitching quadrant images into full detector images, it is necessary to use the *imcopy* command to crop off excess pixels in each quadrant which do not represent detector output. These pixels appear as very bright vertical columns in the image. For the March data, the first column and the final 47 columns were removed.

It is also necessary to rotate each quadrant by the angle specified in [1, p. 4]. To rotate the quadrants, the *rotate* command with the *interpolant* parameter set to “nearest” is used.

The quadrant images are then median-normalized such that each detector image will have approximately the same general amplitude. While the image will be normalized again later by the method described in §2, that method would be sensitive to sudden brightness changes at the edges of the stitched quadrant images, as would appear if each image had a different median. The images are median-normalized here to prevent error in the later method. First, the median of each quadrant is found using the *imstatistics* command. Then, a value is chosen on which to standardize the medians. This could be the median of the detector image from the other dataset, if it is pre-assembled, or just the mean of the quadrant medians. To standardize the medians,

the *imarith* command is used to subtract from each quadrant a constant equal to the quadrant median minus the standard median. In other words, the *imarith* operation will be $image_{quadrant} - (median_{quadrant} - median_{standard})$.

The images are then stitched into detector halves with the *imjoin* commands. The halves are then stitched into entire detector images with the same command.

This process can be automated using the combinequad.cl script described in §**??**.

5.2 Making Detector Threshold and Difference Masks

For each detector image from each dataset, a threshold mask is made representing the location of bad pixels. The creation of this mask involves normalizing the image as described in §2.

To normalize the image, first identify which pixels are assuredly bad or undoubtedly too bright to exclude from the regional median as described in §2. Use ds9 to view a histogram of the image and then identify peaks which undoubtedly represent bad pixels and bright pixels rather than normal pixels or lens artifacts.

Use these boundary values as parameters in the *median* command to normalize the image. The optimal size of the region to median-analyze has not been exactly defined. The region must be large enough to reach to a good pixel from within the center of a region where most pixels are beyond the boundary thresholds. For the data sets analyzed for this project, a region of 17 by 17 pixels was found to be appropriate.

The output of the *median* command is then subtracted from the detector image using the *imarith* command to produce a normalized image.

A bad pixel threshold can then be found using the normalized image. A ds9 histogram and the method described in §2 is used to identify an appropriate threshold.

A threshold mask is then made with the *imexpr* command.

The *imreplace* command and IRAF region notation can be used to remove image defects not related to the cooling process from the threshold mask. Bad pixel lines will commonly have to be removed. The *imcopy* command can be used to isolate corner regions for individual analysis. Corner regions should be removed once copied for separate analysis.

A difference mask can then be produced, lost pixels can be removed, and a count of new bad pixels can be determined. Two image masks can be subtracted with the *imarith* command to produce a difference mask. Lost pixels can be removed from the difference mask using the *imreplace* command. To find a total new bad pixel count, the pixels in the difference mask can be summed with the *blkavg* command (with the *op* paramter set to "sum") and read out with the *listpixels* command.

This process can be automated using the normdiffmask.cl script described in §8.

6 Appendix: Automated Quadrant Stitching with combinequad.cl

The combinequad.cl script has been developed to automate the process of normalizing, rotating, and stitching four individual detector quadrant images into a single detector image as described in §5. The script is written as a procedure script for the IRAF ecl and can be defined, like other procedure scripts, with the ecl command *task $combinequad =/[script location]/combinequad.cl*.

The script has been reproduced verbatim in §7. The script was developed with the assistance of Eric Pellegrini.

The script expects that all four quadrant images are in the local directory. The script then creates the "combinequad" directory to store intermediate and output files. If a directory by this name already exists, it will be necessary to rename or delete it before running combinequad.cl.

In most cases, the script requires only that the four quadrant images be specified in a predefined order as parameters. This order corresponds to the detector arrangement specified in [1, p. 4]: parameter *qul* should correspond to the upper left quadrant of the detector, *qur*=upper right quadrant, *qll*=lower left quadrant, and *qur*=lower right quadrant.

For convenience, Table 4 specifies the exact command line parameters necessary for each detector, following from [1, p. 4].

Detector	*qul*	*qur*	*qll*	*qlr*
0	q3 image	q2 image	q0 image	q1 image
1	q0 image	q3 image	q1 image	q2 image
2	q1 image	q0 image	q2 image	q3 image
3	q2 image	q1 image	q3 image	q0 image

Table 4: Parameters for combinequad.cl appropriate for each detector, following the arrangement in [1, p. 4].

Optional parameters for rotation, normalization, and cropping also exist. The counterclockwise rotation angle necessary to orient the x-axis of the individual quadrant images with the alignment of the detector may be passed by the parameters *rotul*, *rotur*, *rotll*, and *rotlr* respectively for each quadrant. The default values will rotate the quadrant images to meet the alignment specified in [1, p. 4].

By default, the script will not attempt to normalize the images. However, if the *normv* parameter is set to a value other than 0, the script will normalize the median of each quadrant image to that value. A sensible choice for *normv* is the value of the detector

median from some other dataset or the mean of the individual quadrant medians, though the choice should not affect the eventual difference mask.

By default, the script will crop off the first column of each quadrant image and any columns which occur after the following 1024 lines. These dimensions were found to be appropriate for the March 2007 data. If these cropping dimensions are not appropriate for a dataset, it is important that the cropping parameters *cropxa*, *cropxb*, *cropya*, *cropyb* be specified accordingly. If no cropping is necessary, each of these parameters should be set to 1. If alternative cropping dimensions are necessary, *cropxa* and *cropxb* should be specified to match the first and last column of the data in the image, respectively, while *cropya* and *cropyb* should be specified to match the rows of the image data.

7 Appendix: combinequad.cl Script

```
#IRAF Procedure Script: combinequad
#Version 1.0
#Nathan Sanders and Eric Pellegrini September 2007
#
#
#Stitches four quadrant images into a single detector image, rotating and normalizing as per user parameters.
#Make sure there is an existing "combinequad" directory before launching!


procedure combinequad (qul,qur,qll,qlr)

##################Query Parameters
file qul {prompt="The image to be the upper left quadrant"}
file qur {prompt="The image to be the upper right quadrant"}
file qll {prompt="The image to be the lower left quadrant"}
file qlr {prompt="The image to be the lower right quadrant"}

##################Hidden Parameters
int cropxa = 2 {prompt="Leftmost vertical pixel row to include in image to be rotated (must be integer > 0)"}
int cropxb = 1025 {prompt="Rightmost vertical pixel row to include in image to be rotated (must be integer > 0)"}
int cropya = 1 {prompt="Bottommost horizontal pixel row to include in image to be rotated (must be integer > 0)"}
int cropyb = 1024 {prompt="Topmost horizontal pixel row to include in image to be rotated (must be integer > 0)"}

int rotul = 180 {prompt="Number of degrees counterclockwise to rotate the upper left image"}
int rotur = 90 {prompt="Number of degrees counterclockwise to rotate the upper right image"}
int rotll = 270 {prompt="Number of degrees counterclockwise to rotate the lower left image"}
int rotlr = 0 {prompt="Number of degrees counterclockwise to rotate the lower right image"}

int normv = 0 {prompt="Value to normalize the image medians to (0 for no normalization)"}


begin

int medul = 0
int medur = 0
int medll = 0
int medlr = 0

mkdir combinequad

##################Rotate and crop
rotate (interpolant="nearest", rotation=rotul, input=qul//"["//cropxa//":"//cropxb//","//cropya//":"//cropyb//"]", output="combinequad/rotul.fits")
rotate (interpolant="nearest", rotation=rotur, input=qur//"["//cropxa//":"//cropxb//","//cropya//":"//cropyb//"]", output="combinequad/rotur.fits")
rotate (interpolant="nearest", rotation=rotll, input=qll//"["//cropxa//":"//cropxb//","//cropya//":"//cropyb//"]", output="combinequad/rotll.fits")
rotate (interpolant="nearest", rotation=rotlr, input=qlr//"["//cropxa//":"//cropxb//","//cropya//":"//cropyb//"]", output="combinequad/rotlr.fits")

##################Normalize
if (normv != 0) {
imstatistics ("combinequad/rotul.fits", fields="midpt", format=no) | scan(medul)
imstatistics ("combinequad/rotur.fits", fields="midpt", format=no) | scan(medur)
imstatistics ("combinequad/rotll.fits", fields="midpt", format=no) | scan(medll)
```

```
imstatistics ("combinequad/rotlr.fits", fields="midpt", format=no) | scan(medlr)

imarith (operand1="combinequad/rotul.fits", op="-", operand2=((medul)-(normv)), result="combinequad/rotul.fits")
imarith (operand1="combinequad/rotur.fits", op="-", operand2=((medur)-(normv)), result="combinequad/rotur.fits")
imarith (operand1="combinequad/rotll.fits", op="-", operand2=((medll)-(normv)), result="combinequad/rotll.fits")
imarith (operand1="combinequad/rotlr.fits", op="-", operand2=((medlr)-(normv)), result="combinequad/rotlr.fits")
}

##################Stitch
imjoin ("combinequad/rotll.fits,combinequad/rotul.fits","combinequad/left.fits",2)
imjoin ("combinequad/rotlr.fits,combinequad/rotur.fits","combinequad/right.fits",2)
imjoin ("combinequad/left.fits,combinequad/right.fits","combinequad/final.fits",1)

print ("Stitched image saved to /combinequad/final.fits")

end
```

8 Appendix: Automated Bad Pixel Detection with normdiffmask.cl

The normdiffmask.cl script has been developed to automate the process of median-normalizing detector images, determining bad pixel thresholds, preparing threshold and difference masks, and summing bad pixels for two images from the same detector where a cooling occurred in between the capture dates as described in §5. When run, normdiffmask.cl will output the number of bad pixels detected in each detector image as well as the number of new bad pixels. The script is written as a procedure script for the IRAF ecl and can be defined, like other procedure scripts, with the ecl command *task normdiffmask =/[script location]/normdiffmask.cl*.

Unlike when defining combinequad as a task, it is imperative that there be no dollar sign ($) preceding the normdiffmask task name in the preceding command. If the dollar sign is included when defining the task, the list-directed struct variables used in the script will be improperly defined and it will be necessary to do a task redefinition by simply rerunning the IRAF *task* command.

The normdiffmask.cl script has been reproduced verbatim in §9. The script was developed with the assistance of Eric Pellegrini.

The script expects that a newer and older image from a particular detector are in the local directory. The script then creates the “normdiffmask” directory to store intermediate and output files. If a directory by this name already exists, it will be necessary to rename or delete it before running normdiffmask.cl.

The script requires only that the two detector images be specified as parameters. The newer of the two detector images should be passed as the *newer* parameter, and the older of the two as *older*.

The median set dimensions for normalization may be passed as *regionx* and *regiony*. The default value, “17”, for *regionx* and *regiony* is a suitable value for most images. Inspect the median mask produced in the “normdiffmask” directory for abnormalities if you suspect that the median set dimensions are inappropriate for a particular image.

Special usage of normdiffmask.cl is required to differentiate corner region bad pixel development from isolated development. To do so, the script must be run multiple times: once to analyze each affected corner region, and once more with the corner regions specifically excised to analyze isolated pixel development.

To analyze just a corner region of an image, use IRAF region notation when specifying the *newer* and *older* parameters. For instance, to analyze the d1 corner region addressed in §4 for the Juned1.fits and Marchd1.fits image files, use the parameters *newer=Juned1.fits[1:200,1:200] older=Marchd1.fits[1:200,1:200]*.

To excise corner regions and analyze only isolated bad pixel development, employ normdiffmask.cl's bad region parameters. If these parameters are specified, normdiffmask.cl can automatically remove bad regions from the detector images before creating difference masks for counting new bad pixels. Specifying any regions where false-positive bad pixels might be produced, such as bad pixel lines as discussed in §2, will result in more accurate bad pixel statistics. When corner regions are not relavent to the analysis (when analyzing isolated bad pixel development), specify the corner regions as bad regions. Unaltered threshold and difference masks (newermask.fits, oldermask.fits, and diffmask.fits) will still be produced in the "normdiffmask" directory if these bad pixel regions are removed, as the script will make intermediate copies of the threshold masks (newerclean.fits and olderclean.fits) for producing the final difference mask (diffclean.fits).

Each bad region for the newer image should be passed as a *nbadi* parameter, where *i* can be a number from 1 to 6. Similarly, each bad region for the newer image should be passed as a *obadi*. The script could easily be extended to accommodate more than 6 bad regions, if necessary.

The bad region parameters should be defined as strings corresponding to IRAF region notation. For instance, to excise the corner region of d1 from the bad pixel count, "[1:200,1:200]" was passed as both *nbad1* and *obad1*.

The threshold masks produced by normdiffmask.cl will not be identical to those created by hand. The script uses a simple algorithm to detect the extreme pixel boundaries on the initial image before median-normalizing and a similar algorithm to determine the bad pixel threshold from the normalized image. Testing has demonstrated that the values chosen by the algorithm are similar to those defined manually by the histogram-analysis method described in §2, but not identical. This discrepancy will lead to slightly different threshold masks than would be made manually.

The differences between manually-produced threshold masks and those produced by normdiffmask.cl should not be significant. Informal comparison between automatically and manually generated threshold masks indicate that each method is varyingly more or less accurate in some cases, but they are generally equivalent. It is assumed, but will not be quantitatively demonstrated, that the uncertainty in the automatically-produced

threshold masks is similar to those manually-produced.

9 Appendix: normdiffmask.cl Script

```
#IRAF Procedure Script: normdiffmask
#Version 1.0
#Nathan Sanders and Eric Pellegrini, October 2007
#
#
#Given two input images, normalizes them and creates a difference mask.


procedure diffmask (newer,older)

##################Query Parameters
file newer {prompt="The newer of the two images"}
file older {prompt="The older of the two images"}

##################Hidden Parameters
int regionx = 0 {prompt="The horizontal size of the median set region (if set to 0 or not defined, a value of 17 will be used)"}
int regiony = 0 {prompt="The vertical size of the median set region (if set to 0 or not defined, a value of 17 will be used)"}

string nbad1 =  "" {prompt="A bad region (in standard IRAF notation) to remove from the newer image threshold mask before summing bad pixels"}
string nbad2 =  "" {prompt="A bad region (in standard IRAF notation) to remove from the newer image threshold mask before summing bad pixels"}
string nbad3 =  "" {prompt="A bad region (in standard IRAF notation) to remove from the newer image threshold mask before summing bad pixels"}
string nbad4 =  "" {prompt="A bad region (in standard IRAF notation) to remove from the newer image threshold mask before summing bad pixels"}
string nbad5 =  "" {prompt="A bad region (in standard IRAF notation) to remove from the newer image threshold mask before summing bad pixels"}
string nbad6 =  "" {prompt="A bad region (in standard IRAF notation) to remove from the newer image threshold mask before summing bad pixels"}

string obad1 =  "" {prompt="A bad region (in standard IRAF notation) to remove from the older image threshold mask before summing bad pixels"}
string obad2 =  ""  {prompt="A bad region (in standard IRAF notation) to remove from the older image threshold mask before summing bad pixels"}
string obad3 =  "" {prompt="A bad region (in standard IRAF notation) to remove from the older image threshold mask before summing bad pixels"}
string obad4 =  ""  {prompt="A bad region (in standard IRAF notation) to remove from the older image threshold mask before summing bad pixels"}
string obad5 =  "" {prompt="A bad region (in standard IRAF notation) to remove from the older image threshold mask before summing bad pixels"}
string obad6 =  "" {prompt="A bad region (in standard IRAF notation) to remove from the older image threshold mask before summing bad pixels"}

struct *newhist
struct *oldhist

begin

real nmode = 0
real omode = 0
int a = 100
int omin = 10000
real nthresh = 0
real othresh = 0
real pvalue = 1234567
int pcount = 1234567

real nmedlo = 0
real omedlo = 0
real nmedhi = 0
real omedhi = 0


mkdir normdiffmask

if (regionx==0) {regionx=17}
if (regiony==0) {regiony=17}

##################Find extreme boundaries
imstatistics (newer, fields="mode", format=no) | scan(nmode)

phistogram (input=newer, binwidth=100, listout=yes, z2=nmode, > "normdiffmask/newinitialhist.txt")
newhist = "normdiffmask/newinitialhist.txt"
while ( fscan (newhist, pvalue, pcount) != EOF ) {
if (pcount <= 6) {
nmedlo=pvalue
}
}
newhist = ""
print ("Lower extreme boundary for newer image found to be: " // nmedlo)
```

```
phistogram (input=newer, binwidth=100, listout=yes, z1=nmode, > "normdiffmask/newinitialhisthi.txt")
newhist = "normdiffmask/newinitialhisthi.txt"
while ( fscan (newhist, pvalue, pcount) != EOF ) {
if ((pcount <= 6) && (nmedhi==0)) {
nmedhi=pvalue
}
}
newhist = ""
print ("Higher extreme boundary for newer image found to be: " // nmedhi)


imstatistics (older, fields="mode", format=no) | scan(omode)

phistogram (input=older, binwidth=100, listout=yes, z2=omode, > "normdiffmask/oldinitialhist.txt")
oldhist = "normdiffmask/oldinitialhist.txt"
while ( fscan (oldhist, pvalue, pcount) != EOF ) {
if (pcount <= 6) {
omedlo=pvalue
}
}
oldhist = ""
print ("Lower extreme boundary for older image found to be: " // omedlo)

phistogram (input=older, binwidth=100, listout=yes, z1=omode, > "normdiffmask/oldinitialhisthi.txt")
oldhist = "normdiffmask/oldinitialhisthi.txt"
while ( fscan (oldhist, pvalue, pcount) != EOF ) {
if ((pcount <= 6) && (omedhi==0)) {
omedhi=pvalue
}
}
oldhist = ""
print ("Higher extreme boundary for older image found to be: " // omedhi)


##################Median
median (input=newer, output="normdiffmask/newermedian.fits", xwindow=regionx, ywindow=regiony, zloreject=nmedlo, zhireject=nmedhi)
median (input=older, output="normdiffmask/oldermedian.fits", xwindow=regionx, ywindow=regiony, zloreject=omedlo, zhireject=omedhi)
print ('\n',"Created median images for normalization (newmermedian.fits, oldermedian.fits)")

##################Normalize
imarith (operand1=newer, op="-", operand2="normdiffmask/newermedian.fits", result="normdiffmask/newernorm.fits")
imarith (operand1=older, op="-", operand2="normdiffmask/oldermedian.fits", result="normdiffmask/oldernorm.fits")
print "Created normalized images (newernorm.fits, oldernorm.fits)"


phistogram (input="normdiffmask/newernorm.fits", binwidth=100, listout=yes, z2=0, >"normdiffmask/newhist.txt")
newhist = "normdiffmask/newhist.txt"
while ( fscan (newhist, pvalue, pcount) != EOF ) {
if (pcount <= omin) {
omin=pcount
nthresh=pvalue
} else {
omin=pcount
}
}
newhist = ""

print ('\n',"Threshold for newer normalized image found to be " // nthresh)

phistogram (input="normdiffmask/oldernorm.fits", binwidth=100, listout=yes, z2=0, >"normdiffmask/oldhist.txt")
oldhist = "normdiffmask/oldhist.txt"
while ( fscan (oldhist, pvalue, pcount) != EOF ) {
if (pvalue <= 0) {
if (pcount <= omin) {
omin=pcount
othresh=pvalue
} else {
omin=pcount
}
}
}
oldhist=""

print "Threshold for older normalized image found to be " // othresh

##################Make masks
imexpr (expr="a < "//nthresh, output="normdiffmask/newermask.fits",a="normdiffmask/newernorm.fits")
```

```
imexpr (expr="a < "//othresh, output="normdiffmask/oldermask.fits",a="normdiffmask/oldernorm.fits")
print ('\n',"Uncleaned bad pixel masks for each image output to normdiffmask[newer/older]mask.fits")

imarith (operand1="normdiffmask/newermask.fits", op="-", operand2="normdiffmask/oldermask.fits", result="normdiffmask/diffmask.fits")
print "Uncleaned bad pixel difference mask output to normdiffmask/diffmask.fits"

##################Sum bad pixels and report
print ('\n',"Cleaning image files of lost pixels and bad regions and counting bad pixels...")
imcopy (input="normdiffmask/newermask.fits", output="normdiffmask/newerclean.fits")
imreplace (images="normdiffmask/newerclean.fits", value=0, lower=-2, upper=0)
##################Clean bad regions
if (nbad1 != "") {imreplace (images="normdiffmask/newerclean.fits"//nbad1, value=0)}
if (nbad2 != "") {imreplace (images="normdiffmask/newerclean.fits"//nbad2, value=0)}
if (nbad3 != "") {imreplace (images="normdiffmask/newerclean.fits"//nbad3, value=0)}
if (nbad4 != "") {imreplace (images="normdiffmask/newerclean.fits"//nbad4, value=0)}
if (nbad5 != "") {imreplace (images="normdiffmask/newerclean.fits"//nbad5, value=0)}
if (nbad6 != "") {imreplace (images="normdiffmask/newerclean.fits"//nbad6, value=0)}
blkavg (input="normdiffmask/newerclean.fits", output="normdiffmask/newercount.fits", b1=2048, b2=2048, op="sum")
print "Bad pixel count in newer image: "
listpixels ("normdiffmask/newercount.fits")

imcopy (input="normdiffmask/oldermask.fits", output="normdiffmask/olderclean.fits")
imreplace (images="normdiffmask/oldermask.fits", value=0, lower=-2, upper=0)
##################Clean bad regions
if (obad1 != "") {imreplace (images="normdiffmask/olderclean.fits"//obad1, value=0)}
if (obad2 != "") {imreplace (images="normdiffmask/olderclean.fits"//obad2, value=0)}
if (obad3 != "") {imreplace (images="normdiffmask/olderclean.fits"//obad3, value=0)}
if (obad4 != "") {imreplace (images="normdiffmask/olderclean.fits"//obad4, value=0)}
if (obad5 != "") {imreplace (images="normdiffmask/olderclean.fits"//obad5, value=0)}
if (obad6 != "") {imreplace (images="normdiffmask/olderclean.fits"//obad6, value=0)}
blkavg (input="normdiffmask/olderclean.fits", output="normdiffmask/oldercount.fits", b1=2048, b2=2048, op="sum")
print "Bad pixel count in older image: "
listpixels ("normdiffmask/oldercount.fits")

imarith (operand1="normdiffmask/newerclean.fits", op="-", operand2="normdiffmask/olderclean.fits", result="normdiffmask/diffclean.fits")
print "Cleaned bad pixel masks masks output to normdiffmask/[diff/newer/older]clean.fits"

imreplace (images="normdiffmask/diffclean.fits", value=0, lower=-2, upper=0)
blkavg (input="normdiffmask/diffclean.fits", output="normdiffmask/diffcount.fits", b1=2048, b2=2048, op="sum")
print "New bad pixel count: "
listpixels ("normdiffmask/diffcount.fits")

end
```

References

[1] Ed Loh and Dustin Baker. *Cold Test 3, Run 2: Spartan IR Camera for the SOAR Telescope.* 2007. *http://www.pa.msu.edu/ loh/SpartanIRCamera/coldTest3.pdf*