

AD-A168 882

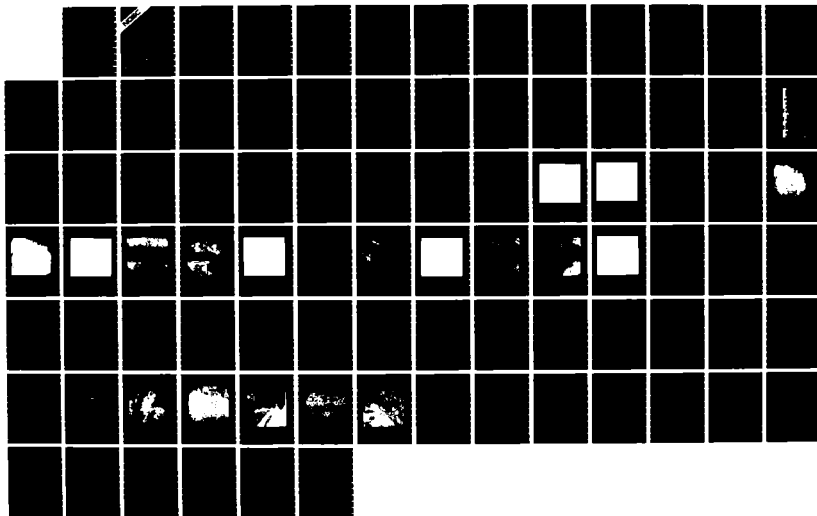
NONLINEAR PIXEL REPLACEMENT ESTIMATION(U) NAVAL OCEAN
SYSTEMS CENTER SAN DIEGO CA R CIGLEDY APR 86
NOSC/TD-879

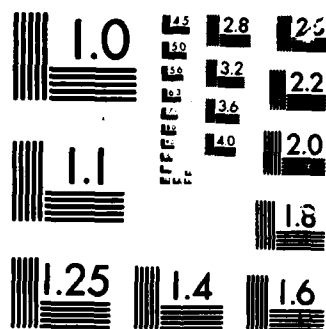
1/1

UNCLASSIFIED

F/G 28/6

NL





12

NOSC
 NAVAL OCEAN SYSTEMS CENTER San Diego, California 92152-5000

Technical Document 879

April 1986

Nonlinear Pixel Replacement Estimation

R. Cigledy

AD-A168 882

DTIC FILE COPY



DTIC
ELECTE
S JUN 23 1986 **D**
 B

Approved for public release. distribution is unlimited.

86 6 11 004

NAVAL OCEAN SYSTEMS CENTER

San Diego, California 92152-5000

F. M. PESTORIUS, CAPT, USN
Commander

R. M. HILLYER
Technical Director

ADMINISTRATIVE INFORMATION

The work discussed in this report was done under the guidance of Naval Ocean Systems Center, Code 742, for the Space and Naval Warfare Systems Command.

Released by
D. K. Forbes, Head
Special Systems Branch

Under authority of
R. L. Petty, Head
Electromagnetic Systems and
Technology Division

ACKNOWLEDGEMENTS

The author would like to thank F. D. Groutage and L. B. Stotts for their professional encouragement and advice; and S. M. Kocsis for his assistance in developing some of the computer programs used in this investigation.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

AD-A168882

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED			1b RESTRICTIVE MARKINGS	
2a SECURITY CLASSIFICATION AUTHORITY			3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.	
2b DECLASSIFICATION/DOWNGRADING SCHEDULE				
4 PERFORMING ORGANIZATION REPORT NUMBER(S) NOSC TD-879			5 MONITORING ORGANIZATION REPORT NUMBER(S)	
6a NAME OF PERFORMING ORGANIZATION Naval Ocean Systems Center		6b OFFICE SYMBOL (if applicable) Code 742	7a NAME OF MONITORING ORGANIZATION	
6c ADDRESS (City, State and ZIP Code) San Diego, CA 92152-5000			7b ADDRESS (City, State and ZIP Code)	
8a NAME OF FUNDING SPONSORING ORGANIZATION Space and Naval Warfare Systems Command		8b OFFICE SYMBOL (if applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
8c ADDRESS (City, State and ZIP Code) Washington, DC 20363-5100			10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO 62712N	
			PROJECT NO SX34	TASK NO DN388 563
11 TITLE (Include Security Classification) Nonlinear Pixel Replacement Estimation				
12 PERSONAL AUTHOR(S) R. Cigledy				
13a TYPE OF REPORT Final		13b TIME COVERED FROM _____ TO _____		14 DATE OF REPORT (Year, Month, Day) April 1986
15 PAGE COUNT 85				
16 SUPPLEMENTARY NOTATION				
17 COSATI CODES FIELD GROUP SUB GROUP			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number) DAEDALUS images, convolutional/nonlinear filtering	
19 ABSTRACT (Continue on reverse if necessary and identify by block number) The intent of this report is to describe the potential merits of three nonlinear estimation techniques for replacing noisy pixels in computer contaminated DAEDALUS imagery.				
20 DISTRIBUTION AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS			21 ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED	
22a NAME OF RESPONSIBLE INDIVIDUAL R. Cigledy			22b TELEPHONE (include Area Code) (619) 225-7639	
			22c OFFICE SYMBOL Code 742	

DD FORM 1473, 84 JAN

83 APR EDITION MAY BE USED UNTIL EXHAUSTED
ALL OTHER EDITIONS ARE OBSOLETEUNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

CONTENTS

INTRODUCTION . . .	1
PROBLEM APPROACH . . .	2
PROPOSED NOISY PIXEL REPLACEMENT ESTIMATORS . . .	2
Neighborhood Averaging Replacement Filter . . .	2
Biweight Filter . . .	3
Median Filter . . .	4
Four-way Median Filter . . .	4
TEST IMAGERY . . .	5
EXPERIMENTAL PROCEDURE . . .	7
NOISE REPLACEMENT . . .	7
BAD PIXEL CHECK . . .	8
BAD PIXEL CHECK NEIGHBORHOOD . . .	9
SIMULATION GROUND RULES AND LIMITATIONS . . .	12
DIGITAL PROCESSING PROCEDURES . . .	12
DESIGN LIMITATIONS . . .	13
FILTER WEAKNESSES . . .	13
RESULTS . . .	17
RAW VERSUS SCALED DAEDALUS IMAGES . . .	17
BAD PIXEL CHECK VERSUS ENTIRE IMAGE FILTERING . . .	17
EFFECTS OF NOISE UPON FILTERING ACTION . . .	19
STATISTIC FOR PERFORMANCE MEASUREMENT . . .	22
FILTERING OF DIFFERENT IMAGES . . .	22
MAXIMUM PERFORMANCE EXPECTED FROM A FILTER . . .	22
IMAGE DISTORTION WITH ZERO-NOISE . . .	24
IMPROVED BAD PIXEL CHECK . . .	25
SUMMARY OF FILTER PERFORMANCE . . .	28
FILTER POTENTIAL . . .	28
FILTER EFFECTIVENESS . . .	28
SURVEY OF FILTERED IMAGES . . .	29
RECOMMENDATION FOR FUTURE STUDY . . .	46
REFERENCES . . .	48
APPENDIX A PLOTS OF RESULTS . . .	A-1
APPENDIX B ILLUSTRATIONS OF ORIGINAL IMAGES . . .	B-1
APPENDIX C PROGRAM LISTINGS . . .	C-1

ILLUSTRATIONS

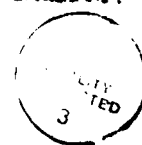
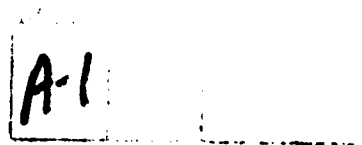
Figure

1. Intensity histograms for raw and scaled lower San Francisco Bay DAEDALUS images . . . 6
2. Effect of noise on image statistics . . . 9
3. Neighborhood versus number of sigmas (SF11DED.IMG) . . . 10
4. Submatrix size geometry . . . 12
5. Four-way median filter processing . . . 15
6. NxN pixel processing geometry . . . 15
7. Effect of increasing noise mean on bad pixel detection . . . 16
8. No bad pixel check versus bad pixel check (SF11DED.IMG) . . . 18
9. Pixel replacement (SF11.IMG) . . . 21
10. Actual/expected performance lower San Francisco Bay, biweight and neighborhood replacement . . . 23
11. Actual/expected performance lower San Francisco Bay, median and four-way median . . . 24
12. Improved filter performance (SF3DED.IMG) . . . 26
13. Improved filter performance (SF11DED.IMG) . . . 27
14. 1% noise mask . . . 30
15. 5% noise mask . . . 31
16. 50% noise mask . . . 32
17. 1% noise on ocean and clouds off Santa Cruz . . . 34
18. ocean and clouds off Santa Cruz median filtered . . . 35
19. Residual noise mask for ocean and clouds after median filtering . . . 36
20. 5% noise on lower San Francisco Bay . . . 37
21. 5% noise median filtered on lower San Francisco Bay . . . 38
22. Residual noise after median filtering of 5% noise on lower San Francisco Bay . . . 39

23.	10% noise on northern San Jose . . .	40
24.	10% noise median filtered on northern San Jose . . .	41
25.	Residual noise on northern San Jose . . .	42
26.	30% noise on Santa Cruz . . .	43
27.	30% noise median filtered on Santa Cruz . . .	44
28.	Residual noise on Santa Cruz . . .	45
29.	Bad pixel mask . . .	46

TABLES

I.	Best filter performance . . .	20
II.	Filter image distortion with zero-noise . . .	25
III.	Improved bad pixel check (11x11 neighborhood) . . .	26



INTRODUCTION

Optical surveillance sensors are sometimes subject to static noise processes which can adversely affect the optimal processing of their resulting imagery. For example, image intensity values derived from line outages, dead pixels, "popcorn" noise and other such noise mechanisms will contaminate both local and global estimates of the power spectrum density, probability density function and other key statistical properties inherent to the original observed scene. If these estimates are used to derive optimum filters for detecting specific targets in said imagery or registering sequences of images, poor processing performance could result. References 1 through 7 provide excellent reviews of current image processing trends dependent on good quality pictures and illustrate their utility for enhancing the inherent information content found in remotely sensed images such as those taken by the LANDSAT and NIMBUS-7 satellites. They also show the effect of noisy pixels on these techniques and the types of performance degradations incurred; which can be significant. This suggests that methods for replacing the "bad" pixel values with numbers commensurate with the inherent statistics of a detected image can be important to optimum image processing in many applications.

The standard techniques for replacing bad pixels is to replace their recorded values with the scene mean intensity value, or to average the eight intensity values surrounding the bad pixel and substitute the results for the incorrect values. Unfortunately these methods do not necessarily take into account the spatial coherence properties of the inherent surface clutter and may not give the best representation of the true intensity values which should be there. The intent of this report is to describe the potential merits of three nonlinear estimation techniques for replacing noisy pixels in computer contaminated DAEDALUS imagery. The performance of these techniques will be benchmarked against the neighborhood averaging technique cited above.

PROBLEM APPROACH

During the past decade several nonlinear estimation techniques have been used to smooth spiky noise-contaminated time series and optical-image data, and their success has been reported in references 5, 6, and 8-10. Median filters have emerged as one of the best methods found since they are effective in estimating reasonable insertion values, while still preserving any monotonic step edges present in the data (8). However, these filters and other similar data estimators do contaminate the basic intensity statistics and this can affect subsequent multispectral or time series data processing in some nonlinear fashion. This can be especially detrimental when strong image-to-image correlations are desired (11). The question is, "which nonlinear data estimator provides the best data representation after application?" In an attempt to answer this question, three popular nonlinear data filtering techniques were assessed in their ability to correct computer contaminated DAEDALUS imagery of significant outliers, with and without a bad pixel localization routine, and create reasonably accurate renditions of the uncontaminated scenes. The benchmark performance levels were assumed to be those obtainable from linear filtering using a neighborhood average and replacement technique on the same noisy imagery. In the next two subsections, noisy pixel replacement estimators, specific DAEDALUS imagery, and detailed experimental procedures used in this investigation will be described.

PROPOSED NOISY PIXEL REPLACEMENT ESTIMATORS

Four data replacement estimators were assumed for this work; the first a linear technique and the last three were of a nonlinear nature: namely

- Neighborhood Averaging Filter
- Biweight Filter
- Median Filter
- Four Way Median Filter

This subsection provides a detailed description of each. Each filter operates on the nine pixels found in a 3x3 pixel window about the pixel to be replaced and these pixels are denoted in the following matrix form:

$x(i-1,j-1)$	$x(i-1,j)$	$x(i-1,j+1)$
$x(i,j-1)$	$x(i,j)$	$x(i,j+1)$
$x(i+1,j-1)$	$x(i+1,j)$	$x(i+1,j+1)$

In the above matrix, $x(i,j)$ is the candidate pixel to be replaced. For the biweight and median filters, these nine values were passed to a sorting routine (SHELLSORT) so they could be assembled in low to high order. The other two filters did not require sorting and no other additional prefiltering operations were imposed.

Neighborhood Averaging Replacement Filter

The neighborhood averaging replacement filter is one which sums the amplitude values of the eight array elements surrounding $x(i,j)$ and divides the result by eight to yield the replacement value for $x(i,j)$. This computation is similar to a local neighborhood mean estimate, except that the center element is missing from the calculation. The effectiveness of this estimation

method is dependent upon the percentage of noise inherent in the image, the spatial variability of the scene, and whether some sort of noisy pixel screening procedure is applied to determine the need for filtering. Since this is the benchmark technique for the study, these points will be discussed in greater detail.

Most infrared and optical imagery are Markov-like random processes and have autocovariance functions with envelopes which fall off exponentially over some linear distance(s). If the clutter level of the image is low, a bad pixel replacement can result because of either the presence of outliers, or rapid decay of the spatial covariance. This latter situation implies the spatial variability of the scene is high, such as an image of a city or other such cluttered terrain. A noisy pixel test can minimize the occurrence of the former, but nothing can really be done about the former without more information. If the autocovariance of images falls off slowly, which implies slowly varying clutter fields, bad pixel replacement will most likely only occur with outliers present. Here again a noisy pixel test will help minimize efforts in pixel replacement.

When the noise level is high, bad pixel replacements can occur because the mean and standard deviations derived from the imagery may replace good values with bad or pass noisy pixels off as good. This is especially possible when the spatial variability of the clutter is high. Obviously, a noisy pixel test may not improve anything at all if its value cannot be easily determined by the chosen test.

Given the above, nonlinear estimation techniques suffer the same problems and/or to the same degree. This is the objective of the study and will be the aspect investigated in some detail in the sections to come.

Biweight Filter

For every pixel in an image of interest, an ordered nine element string of intensities,

$$x(1)<x(2)<x(3)\dots<x(8)<x(9),$$

is passed to the biweight filter subroutine (BIWEIGHT) for pixel replacement. The first operation performed is to calculate the string standard deviation based upon the interquartile values $x(3)$ and $x(7)$, i.e. the subroutine computes

$$\text{standard deviation (SD)} = [x(7)-x(3)]/1.349; \text{ for } x(7).ne.x(3) \quad (1a)$$

$$= 1.483 (=1/.6745) \text{ for } x(7).eq.x(3). \quad (1b)$$

The subroutine then derives an estimated mean using those elements in the string within five standard deviations of the median element $x(5)$. Mathematically the program calculates

$$\hat{x} = \frac{\sum_{i=1}^9 x(i) \cdot \text{rect}[(x(i)-x(5))/(10 \cdot \text{SD})]}{\sum_{i=1}^9 \text{rect}[(x(i)-x(5))/(10 \cdot \text{SD})]} \quad (2)$$

where $\hat{x}$ is the estimated mean and $\text{rect}(z)$ is the rectangular function given by

$$\begin{aligned} \text{rect}(z) &= 1; \text{ for } \text{abs}(z) < 1/2 \\ &= 0; \text{ otherwise.} \end{aligned}$$

This particular form of mean estimate allows one to derive an estimated mean devoid of significant outliers which would contaminate the estimate.

A weighting function is then calculated using the formula

$$w(i) = [1 - ((x(i) - \hat{x}) / (6 \cdot \text{SD}))^2]^2 \quad (3)$$

for $[(x(i) - \hat{x}) / (6 \cdot \text{SD})]^2 > 1.0$ the term becomes 1.0 and $w(i) = 0$. A new estimate of the mean is computed through the relation

$$\hat{x} = \frac{\sum_{i=1}^9 x(i) \cdot w(i)}{\sum_{i=1}^9 w(i)} \quad (4)$$

and this number replaces the array element $x(i,j)$. This technique has been suggested as a more robust means of smoothing noisy data sets than the median filter [12] and this assertion will be evaluated in the study.

Median Filter

For every pixel in an image of interest, an ordered nine element string of intensities,

$$x(1) < x(2) < x(3) \dots < x(8) < x(9),$$

is passed to the median filter subroutine and the median string element $x(5)$ is substituted for $x(i,j)$. The major advantage of median filtering is that constant backgrounds, slopes and edges are preserved, while isolated pulses less than or equal to $m = (n-1)/2$ are suppressed, n being the length of the data string and equal to 9 in this case.

Four-way Median Filter

The four-way median filter is a filtering routine which applies a one-dimensional median filter in the north-south, east-west, southwest-northeast and northwest-southeast directions sequentially. Pictorially, the filter operates in the following sequence order:

$$\begin{array}{ccc} 4 & 1 & 3 \\ 2 & * & 2 \\ 3 & 1 & 4 \end{array}$$

Although this filter only operates on three-pixel strings at any one pass, this pixel estimation technique requires a larger initial array size upon which to operate than the previous techniques. This is because the subroutine performs an in-place four-way filtering operation and the resulting array size from each pass is reduced by 2 rows and 2 columns.

Unlike the neighborhood average and biweight filters previously described, this method does not replace array elements with computed values, but rather replaces them with one of the nine original pixel values. The replacement may be a good or noisy pixel, depending on its relationship to its adjacent neighbors on each pass, and whether a bad pixel test is applied. Technically, this type of filter derives a proper replacement only when one noisy pixel exists in any chosen 3x3 pixel array and may make an error otherwise if more than one outlier exists in these windows. Clearly the particular position of the noisy pixels relative to the order sequence of the filter will define whether an error is made or not. For example, all the pixels in the 3x3 matrix could be noisy except for the upper left and the lower right corners. On the fourth pass, a good replacement would occur.

TEST IMAGERY

The test imagery chosen for this investigation is a set of 512x512x8 bits DAEDALUS derived images provided by E.M. Winter of Technical Research Associates [13]. The DAEDALUS sensor was flown in the NASA/AMES U-2 over San Francisco Bay and points south on 14 September 1983, and specific locations interrogated included San Jose, Santa Cruz, Monterey Bay and Point Mugu, California. The images used in this study were from the channel 5 segment and possess an individual pixel field-of-view of approximately 28 meters. More information on this flight can be obtained by contacting Dr. Winter at (805) 987-1972 in care of Technical Research Associates, Inc., 445 Rosewood Avenue, Suite H, Camarillo, California 93010.

This set was chosen because of the diversity of terrain recorded, i.e., the range of spatial variability interrogated. The six images selected for the investigation were:

1. Lower San Francisco Bay (SF1):85% bay, 15% city and flatlands
2. Northern San Jose (SF2):85% city, 15% bay
3. Downtown San Jose (SF3):95% city, 5% flatlands
4. Santa Cruz Mountains (SF6):100% mountains
5. Santa Cruz (SF8):30% city, 70% ocean
6. Ocean and clouds off Santa Cruz (SF11):30% ocean, 70% clouds

These images are shown in their uncontaminated form in appendix B. In the work to be described, two versions of the above imagery were used. One version used the calibrated DAEDALUS imagery directly in the study. Unfortunately, most of the images possessed pixel values between 30 and 120, and this gave a very low contrast image on the AED512 displays used. To alleviate this problem, a second version was created which had the six images minimum/maximum scaled into an 8 bit-digitized intensity form, i.e., the minimum value of the image was set to 0, and the maximum value set to 255 and the rest of the pixel values were sorted into the 254-integer levels in between. This allowed better definition on the display and a better sense of reality to the viewer. However, it does affect the inherent image statistics in a nonlinear way.

Figure 1 illustrates the resulting intensity histograms for lower San Francisco Bay scene. It is apparent that the figure possesses very similar envelopes, as one would expect. The scaled DAEDALUS histogram is simply an expanded version of the unscaled DAEDALUS by a factor 2.83. All of the DAEDALUS imagery used in this study exhibited similar type scaling. The impact of this scaling is that the scaled images will produce much larger difference variances than the unscaled images when both are subjected to the same statistical manipulation. However, this is an artificial difference. The relative performance of the various filters on the two image versions is the same and one will be able to draw the same conclusions by focusing upon one or the other set. In other words, absolute variances derived from unscaled results should not be compared with those obtained from scaled results.

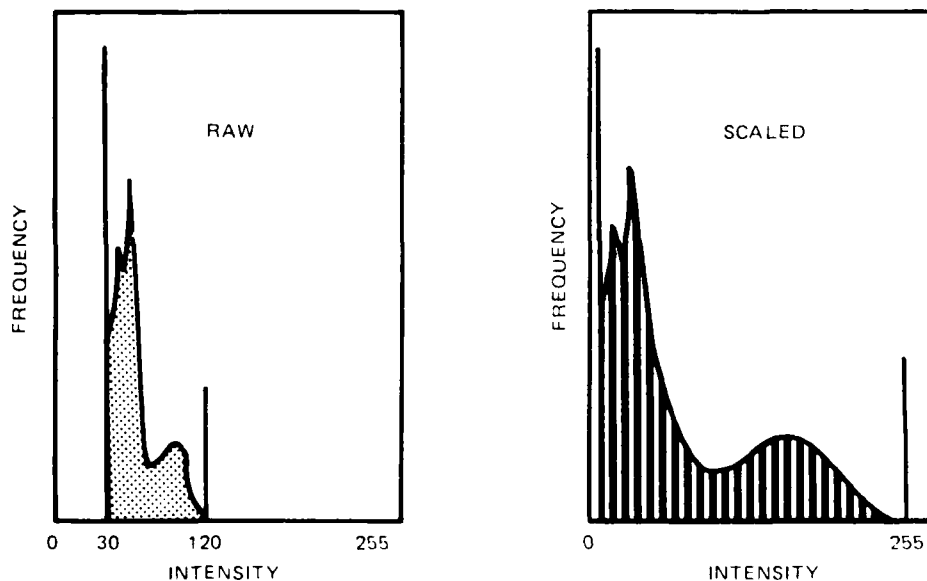


Figure 1. Intensity histograms for raw and scaled lower San Francisco Bay DAEDALUS images.

EXPERIMENTAL PROCEDURE

Programs (see appendix C) were written to allow the operator to select an image, read it into memory, offset the entire image from zero by +300, and allow selection of the following options:

- Introduction of Gaussian noise: percent, mean, and sigma of noise
- Bad pixel check (skip or perform)
- Number of sigmas from the mean of the selected neighborhood (for bad pixel check)
- Neighborhood size; 3x3, 5x5, 7x7, 9x9, 11x11 (for bad pixel check)

After selection of these options, the program introduced noise or not, then displayed the mean and sigma of the difference between the original and noise contaminated images. The program then either skipped or performed the bad pixel check on a 452x452 subset of the original image. If the bad pixel check was desired, the mean and sigma of the selected neighborhood around the pixel was compared with the value of the pixel itself, and if it was within the selected number of sigmas from the neighborhood mean, it was determined to be a good pixel, and would not have to be filtered.

If the bad pixel check was skipped, then every pixel was filtered; when the bad pixel check was performed, only the outlier pixels had the filtering technique applied.

After the application of the selected filtering algorithm, the program displayed the correlation coefficient between the original and the filtered images, the mean and sigma of the difference between the original and the filtered images, the percent of noise contamination selected, and the mean and sigma of the added noise. In addition, the numbers of pixels replaced by noise and the number of pixels corrected by the filter were displayed. The number of noise pixels corrected and the number of good pixels disturbed were also displayed.

NOISE REPLACEMENT

The procedure for noise replacement was first to move the range of the pixel values from 0-255 to 300-555 for the scaled DAEDALUS images, and from approximately 30-120 to 330-420 for the original DAEDALUS images. This was done by adding 300 to every pixel value after the image was taken from disk. By this method, the introduction of Gaussian noise with a mean of 0.0 and a sigma of 5.0 would make the noise distinctive from the original pixel values. Noise values replacing original image values were then -20 to +20, with the majority close to zero.

The actual noise replacement was done randomly over a 462x462 subset of the 512x512 image. This allowed the actual experimental window of 452x452, over which the bad pixel check and the filtering would occur, to be well within the noise-contaminated area. For the four-way median filter, the contaminated area was slightly larger, 468x468, due to the added size of the image required, as the filter shrinks the image by two rows and columns for each of the four filter passes, resulting finally in the standard 452x452 image. As a result, the percentage of noise replacement appears to be less than that of the other filtering algorithms; on the other hand since the four-way median filter makes four passes; the percentage of noise dealt with is equal in percent to the noise dealt with by the other filtering methods.

As the program was repeated for different percentages of noise and with and without the bad pixel check, and for six different images, the random stream was always the same so that no one filter was given a different, or perhaps adverse set of noise upon which to work.

The program allows replacement with Gaussian means and sigmas other than 0.0 and 5.0. The decision to use these values was based upon the need to use noise that was different from the original image pixel values, and thus stand out.

Noise replacement values used in the experiment were 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, and 30 percent. Zero percent was selected so that a gauge could be established on how much the particular filter would disturb the image when there was no noise replacement. By careful examination of the original DAEDALUS images, flaws were discovered in the scanning sensor output in every image. Therefore, it is reasonable to assume that in every filtering action, some of this "noise" would be replaced.

BAD PIXEL CHECK

The program allows for two variations in the bad pixel check; first, to skip it completely, and second to vary the number of sigmas.

If the bad pixel check is skipped, then the filter is applied to every pixel of the image, which for a 452x452 image is 204,304 pixels. This is the maximum work for the filter, and the test of how well or badly the filter performs in correcting the noisy pixels and not disturbing too many of the good pixels.

The second variation, the number of sigmas, determines how much work the filter does. During the bad pixel check, the sigma and mean are computed for the selected neighborhood (3x3, 5x5, 7x7, 9x9, or 11x11) around the pixel which is a candidate for filtering. If the candidate pixel is within plus or minus the selected number of sigmas from the mean, the pixel is considered good and does not require filtering. If it falls beyond the selected number of sigmas about the mean, then the pixel is a candidate for filtering. By reducing the number of sigmas, more filtering is achieved; by increasing the number of sigmas; filtering is reduced.

If the filtering is reduced, less harm is done when the candidate pixel is one of the original image pixels, but when the candidate pixel is a noisy pixel, then it may escape detection as a bad pixel when the number of sigmas

is high (i.e., 3.0 or more). On the other hand, when the number of sigmas is low (making the good pixel window smaller) more pixels will be filtered; some good and some of the noisy pixels. This is the tradeoff in using the bad pixel check; a number of sigmas has to be selected which will detect the noisy yet not filter the good pixels. Several of the filters, by nature of their design, tend to do less harm to good pixels, and have more success with elimination of noisy pixels. These are the median and four-way median filters. These filters do not arithmetically modify pixel values; but by the process of elimination move pixels around. Noise pixels become the outliers and pixels within the neighborhood of the candidate pixel take the place of the noisy pixel. The neighboring pixels are not the same as the replaced pixel, but are very close to its value if the gradient of the neighborhood is low.

BAD PIXEL CHECK NEIGHBORHOOD

Another variable for the bad pixel check is the neighborhood of the candidate pixel. The operator is given the opportunity of selecting 3x3, 5x5, 7x7, 9x9, or 11x11 submatrix for the neighborhood. During the bad pixel check, the pixels in the submatrix selected are averaged, while at the same time the standard deviation of the neighborhood is computed. Then the candidate pixel is checked to see if it is within plus or minus the number of sigmas around the mean. If so, the pixel is considered good; if not, it is filtered. The size of the neighborhood affects the outcome of the bad pixel check; if the neighborhood is small, one or two bad pixels in the neighborhood can contribute to the mean and standard deviation in such a way as to bias the true mean of the neighborhood. This is shown in figure 2.

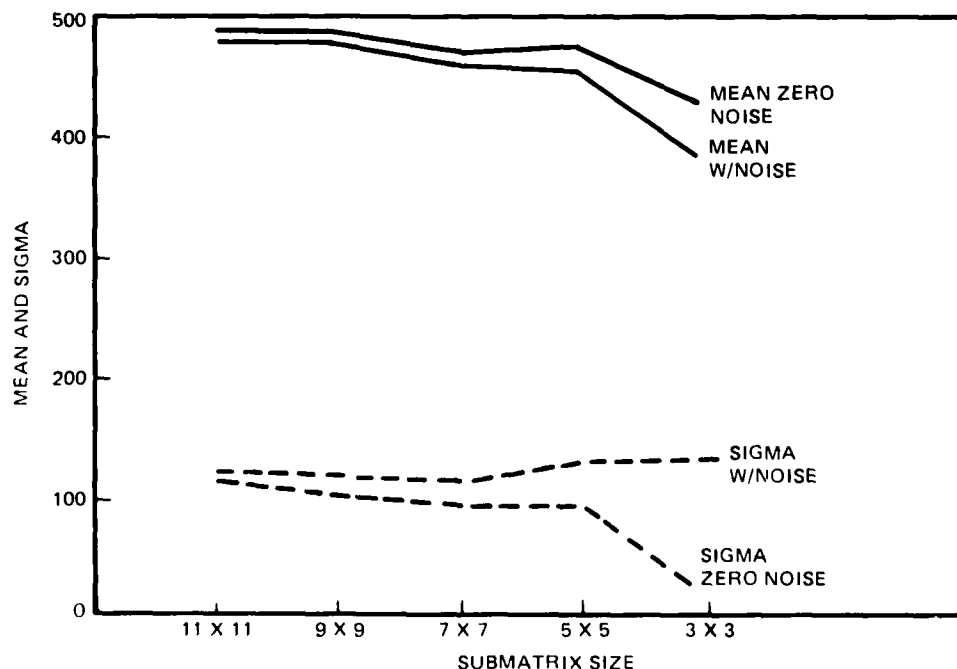


Figure 2. Effect of noise on image statistics.

For the zero-noise case, the mean and sigma for the 3x3 neighborhood are 430 and 22; after introduction of one noisy pixel, the mean dropped to 383, and the sigma increased to 137. On the other hand, for the 11x11 neighborhood, the mean shifted down from 492 to 488, and the sigma increased from 120 to 127, with the introduction of only one noisy pixel. This information was produced with simulated data for an image which had a moderate gradient. The main purpose of including the information is to show the effects of small amounts of noise on small neighborhoods, and indicate how a single bad/noisy pixel can radically affect the normal statistical measures of quality on applications in image processing.

In figure 3, the combined effects of number of sigmas and neighborhood size selections are shown upon the numbers of noisy pixels and good pixels replaced. This study was made with only one percent noise using the median filter on the ocean and cloud image (SF11DED). The plots 3x3g and 3x3b represent respectively the amount of good and bad pixels replaced by the action of the filter with increasing number of sigmas selected for the 3x3 neighborhood. The 3x3b plot at number of sigmas = 2 replaces less than the total (2076) noisy pixels, so its effectiveness is not perfect at number of sigmas = 2. In addition, at number of sigmas = 2, the 3x3 (3x3g) neighborhood is responsible for replacing (and perhaps disturbing) 4000 good pixels. At number of sigmas = 1 (not shown), the 3x3 neighborhood replaces all of the 2076 noisy pixels, but disturbs more than 61,000 good pixels. The best operating point for the 3x3 neighborhood is at number of sigmas = 2.8265, where 1906 bad pixels are replaced and only 104 good pixels are disturbed. The neighborhood which performs the bad pixels check best is the 9x9 neighborhood with number of sigmas = 4, where 2051 noisy pixels are replaced and only six good pixels are disturbed. Of course these plots were made for the most benign (least cluttered) image in the study (SF11DED.IMG).

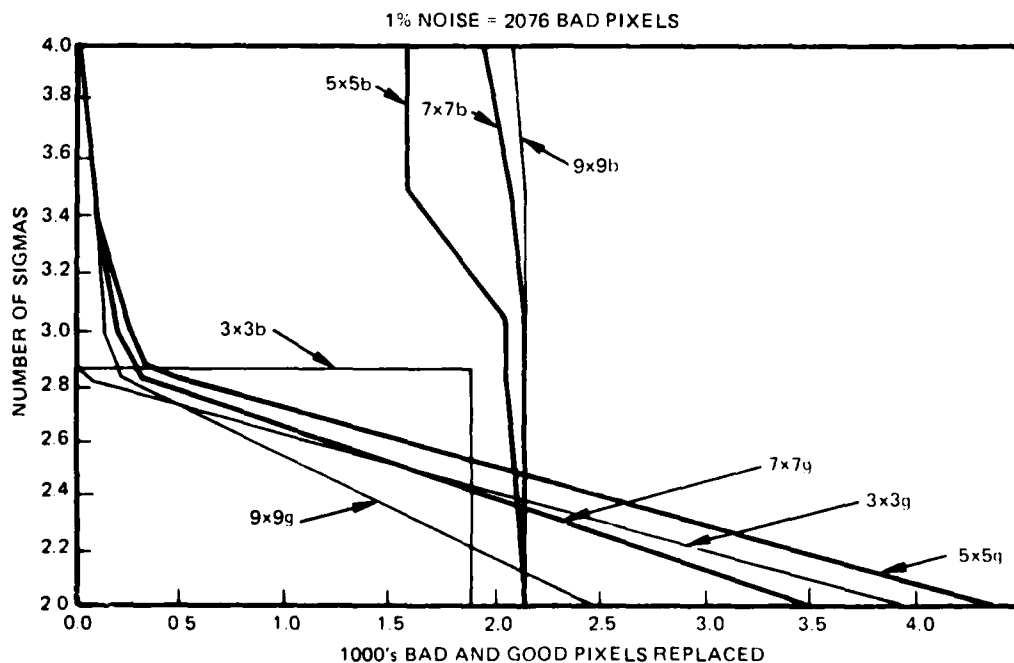


Figure 3 Neighborhood versus number sigmas (SF11DED.IMG).

The apparent result of this analysis is to select a large neighborhood. The original set of data (shown in appendix A) was taken with the bad pixel check using a 5x5 neighborhood and number of sigmas = 2. When the discovery was made that using the bad pixel check with the 5x5 neighborhood and the number of sigmas = 2 resulted in not detecting all of the bad pixels, and at the same time disturbing many good pixels, more data were taken using an 11x11 neighborhood and varying numbers of sigmas depending upon the amount of noise in the image. The number of sigmas varied from 4.1 for zero noise to 1.0 for 40% noise.

SIMULATION GROUND RULES AND LIMITATIONS

This section outlines the major ground rules and limitations of the performance and are described in the following sections.

DIGITAL PROCESSING PROCEDURES

In the initial design of the experiment, everything was done to assure uniformity of procedures for each filter, so that comparison of filtering techniques would be based upon identity of noise-contaminated images for each filter input. The random number generator was seeded at the same starting point so that every image for each filter would have the same value of noise entered into the same pixels. The selection of a 452x452 submatrix image of the original 512x512 DAEDALUS images was standard throughout the study. The introduction of noise was over a 462x462 submatrix for the biweight, median, and neighborhood replacement filters. The four-way median filter required noise to be introduced over a 468x468 submatrix, due to the shrinking of the four-way median filter through each of its four passes, resulting finally in the 452x452 image which was compared to the original image. The geometry for the submatrix sizes is shown in figure 4.

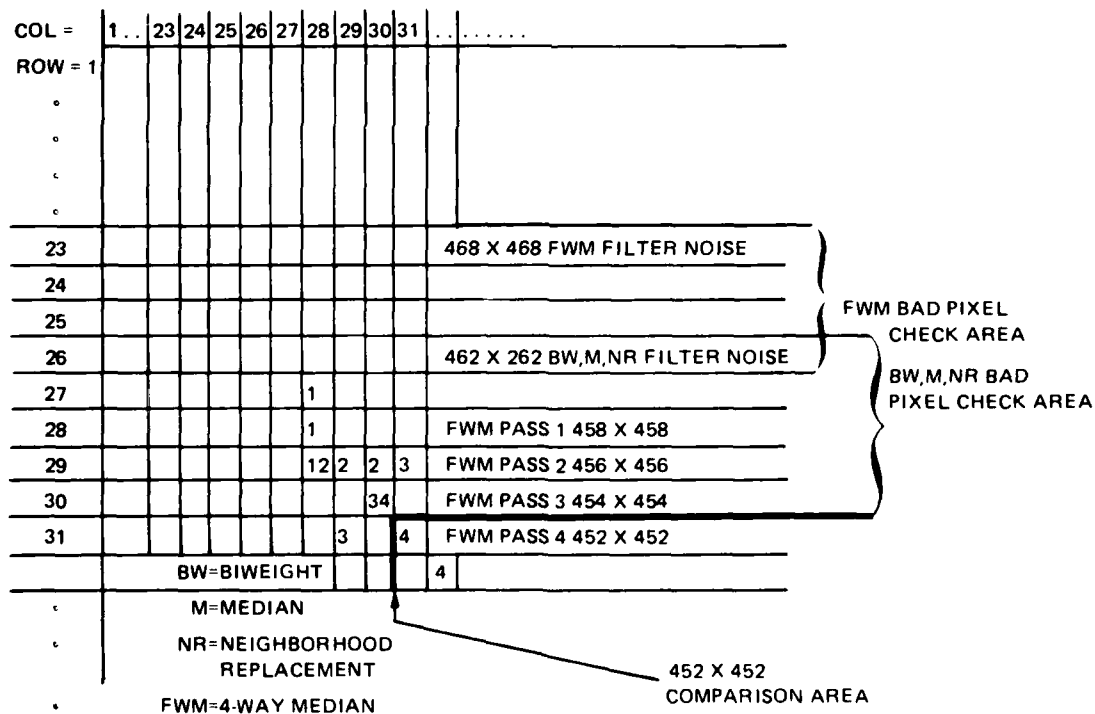


Figure 4. Submatrix size geometry.

In the design of the programs, options were added to vary the mean and sigma of the Gaussian noise to be replaced in the image and to vary the number of sigmas for the bad pixel check. In addition, provision was made for the

selection of any one of the five different neighborhoods for the bad pixel check. These were extra features which extend the capability of testing.

DESIGN LIMITATIONS

Some of the limitations caused by program design considerations are the images themselves. Initially taken from the DAEDALUS tapes, these images are in raw, byte form, eight bits per pixel. The raw data image pixels have a limited integer range, usually from 30 to 120. In order to display these images and obtain a good definition of geographic features, the programmer must scale the images into the color table (or gray-scale) range of 0 to 255. Both types of images, the raw and the scaled images, were included in this study. The difficulty of using various filters is that floating point values generated by the filtering algorithm (for example, the biweight and the neighborhood replacement filters) are rounded to the nearest integer before being placed back into the image. When the difference between the original and the filtered images is taken, these roundoff errors have been insignificant due to the randomness of the dropped fraction and the fact that NINT (nearest integer) function is used. If the pixel value is nn.00 to nn.4999, only those amounts are lost. If the pixel value is nn.5000 to nn.9999, the corresponding amounts are gained. Thus, over a large n, the differences disappear. This has been verified by a program. The differences are not visible until the third decimal place, and the differences are inconsistent depending upon the amount of noise induced into the image.

In the illustrations of the filtered images, a small border of noise remains around the edge of the image. This five-pixel noise border is outside of the 452x452 filtered images area, but noise has been added to this area (462x462) to simulate a full image with noise. The five-pixel border allows for an 11x11 neighborhood for the bad pixel check. The noise area for the four-way median filter also contains a five-pixel border with residual noise for the same reason. The four-way median filter five-pixel noise border is three pixels in each direction farther out from the other three filters' noise border.

FILTER WEAKNESSES

In the original study, the effectiveness of the filters begin to deteriorate rapidly after 10% noise. Beyond 30%, the filters begin to lock onto noise and begin to reject the remaining good pixels. Tests (not shown) have been conducted to verify this condition. However, it is not expected that images with more than 10% noisy pixels would be worth processing.

The complexity and consequent time involved in running these filters on a computer are not too great; but compared among themselves, there is quite a difference in execution time.

The simplest is the neighborhood replacement filter. It requires no sorting or multiple passes. It simply sums the eight neighbors and divides by eight.

The next in complexity is the median filter, which assembles the nine pixels from the 3x3 array around the candidate pixel into a 1x9 array. This array is then passed to the routine SHELLSORT, which on the average makes six

passes to sort the nine numbers. The routine would be more efficient if there were more than nine values. When the array passed to SHELLSORT is already in low to high order, three passes are required which performs 20 IF statements, but no movement of values. When the routine is done, the array contains the values in ascending order. The last step of the filter is to store the median value $x(5)$ into the candidate pixel.

A complicated filter is the biweight filter. It performs the same steps as the median filter, assembling the 1×9 array, passing it to SHELLSORT, but on return from SHELLSORT, the biweight routine is called to do the rest of the processing as described above. Formerly, biweight was an iterative routine which refined the answer through as many as 200 iterations. But since the test for convergence was 0.01 between the former ($\hat{x}$) and the new ($\hat{x}$), this portion of the routine was removed. The operation of the NINT statement in converting the floating point result to integer, more than compensated for any further refinement.

The application of the four-way median filter requires four passes, making it the most time-consuming filter in this study. The actual method used was to do an in-place four-way filter reducing the loop control by two rows and two columns each iteration. Since for each iteration the number of pixels involved is only three, the sort routine was not used. In figure 5, five IF statements were used to solve the truth table.

Programming application requires that when applying a two-dimensional loop control to an $n \times n$ matrix, the solution of a matrix operation proceeds row by row and column by column. When in-place filtering is performed using a 3×3 window, as all the selected filters in this study are, the following phenomena becomes apparent as shown in figure 6.

The pixels above the bold line are pixels which have already been filtered; the pixels below the bold line are pixels which have not been filtered, including the candidate pixel. This is an example of in-place filtering. There are problems and also advantages to this method of processing. The advantages are:

1. If noise has been eliminated from the pixels that have already been processed, the neighborhood mean and sigma for the bad pixel check will aid in determining more accurately whether the candidate pixel is good or has to be filtered.

2. If the pixel has to be filtered, the filtering algorithm will have had fewer noisy pixels to deal with in determining a substitute value.

The problems are:

1. If a good pixel has been replaced by another, or a noisy pixel has not been filtered and still remains in the already processed area, then the neighborhood mean and sigma for the bad pixel check will possibly result in tagging a good pixel bad, or tagging a noisy pixel as good. In either case, this is not desirable.

2. In the filtering algorithm, whether the candidate pixel is good or bad, the chances are that it will be replaced with one that is either worse or

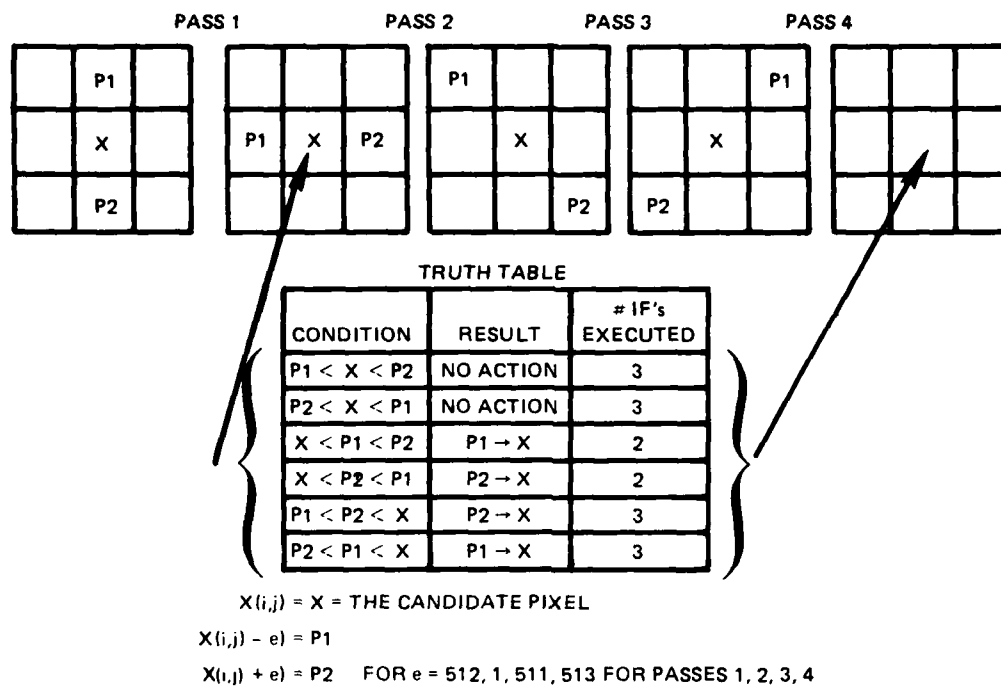


Figure 5. Four-way median filter processing.

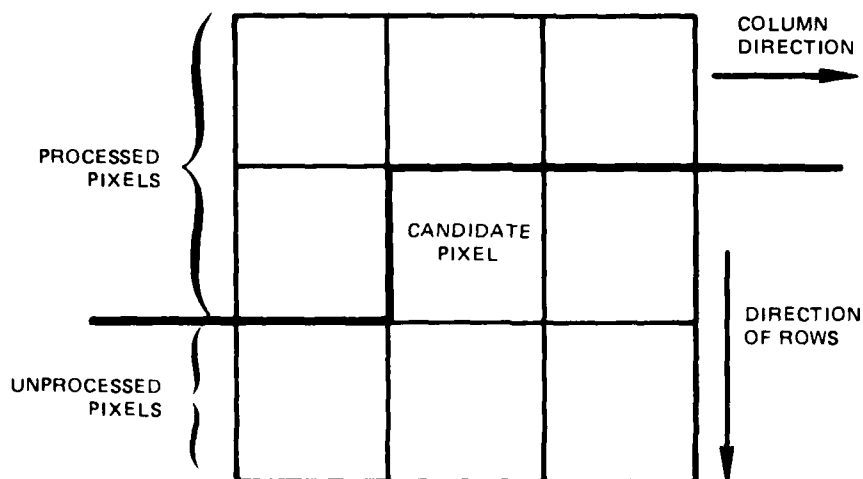


Figure 6. $N \times N$ pixel processing geometry.

almost as bad. Unfortunately, some of the advantages and some of the problems prevail throughout the filtering techniques studied in this experiment.

The solution then, could be to not perform in-place filtering. This is accomplished by building the filtered image into a separate image. This was tried, and in all cases the results were slightly degraded, which means that the advantages of in-place filtering slightly out-weigh the disadvantages. Of course when the bad pixel check is skipped, a large part of the problems of in-place filtering disappears, and the advantages become preponderant.

Experimentation was performed with a noisy mean other than 0; a mean of 50 was tested which made all of the noise positive, and as expected, the difference sigmas were slightly improved.

The closer the mean of the noisy pixels approached the mean of the good image pixels, the difficulty in detecting the noisy pixels become more pronounced. As shown in figure 7, the upper tail of the noisy pixel distribution intrudes upon the lower tail of the good pixel distribution. When this happens, a lower number of sigmas for the bad pixel check results in a smaller number of noisy pixels replaced and a greater number of good pixels disturbed. The number of sigmas for the bad pixel check must include all the good pixels, while at the same time excluding all of the noisy pixels.

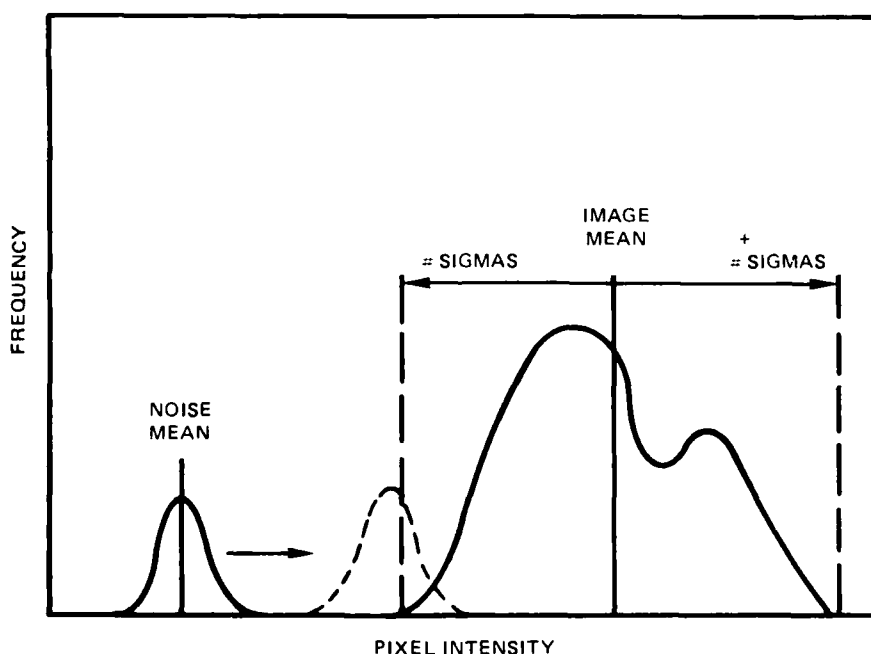


Figure 7. Effect of increasing noise mean on bad pixel detection.

RESULTS

In addition to the filter performance results, this study provided findings regarding:

1. The use of raw versus scaled DAEDALUS images.
2. The use of the bad pixel check versus applying the filter over the entire image.
3. Characteristics of increasing noise and the effects upon the filters' performances.
4. Measurement statistic for filter performance.
5. Filtering of different images.
6. Maximum performance expected from a given filter.
7. Image distortion with zero-noise.
8. Improvement of the bad pixel check.

RAW VERSUS SCALED DAEDALUS IMAGES

In all of the raw DAEDALUS image plots, all four filters performed better than when the scaled DAEDALUS images were used. The performance was similar for both types of data, but the raw images resulted in lower difference sigmas between the original and filtered versions. This was due to the narrower range of histogram values exhibited by the raw DAEDALUS data which were usually between 30 and 120, as opposed to the scaled DAEDALUS data which were between 0 and 255. These differences appear significant, but they are only relative.

BAD PIXEL CHECK VERSUS ENTIRE IMAGE FILTERING

The ranking of the filters changed radically when the filter was applied over the entire image (no bad pixel check), and specifically, the neighborhood replacement filter performance declined adversely over the low range of noise (0 to 4 percent), see figure 8.

Generally using the bad pixel check, the following results were true:

1. At 0 and 1 percent noise the biweight filter was better than the rest sometimes by only a very slight margin. The four-way filter outperformed the biweight at 0 percent in 3 out of the 6 images, doing better on land alone, as opposed to the biweight's better performance when water was involved in the image.

2. At 2 percent noise, sometimes the biweight, median, or four-way was the best. With the raw DAEDALUS images, the biweight didn't do as well; a tie usually resulted between the median and the four-way median.

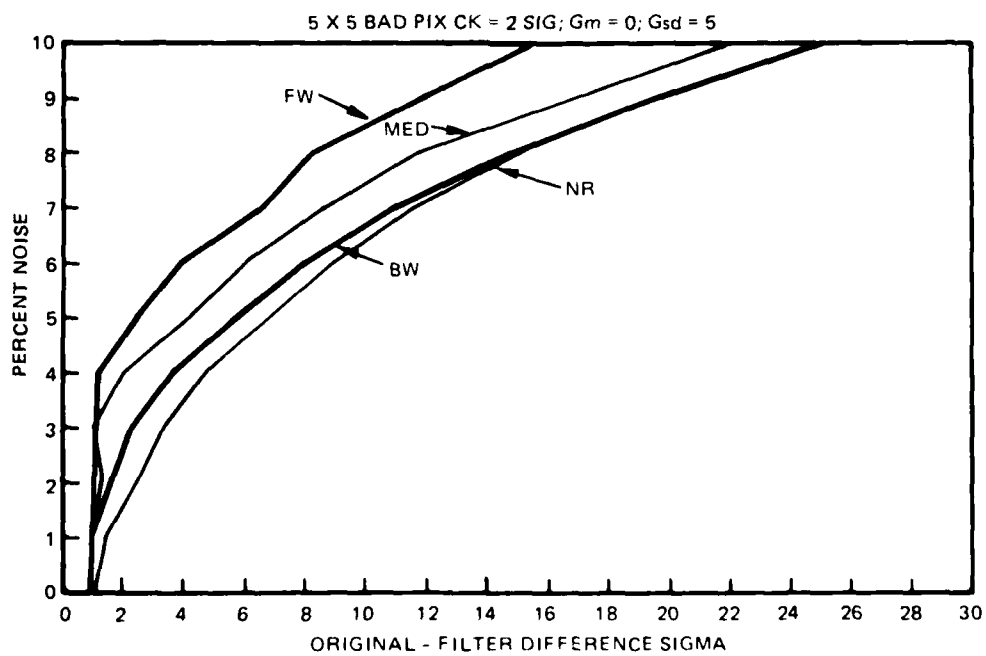
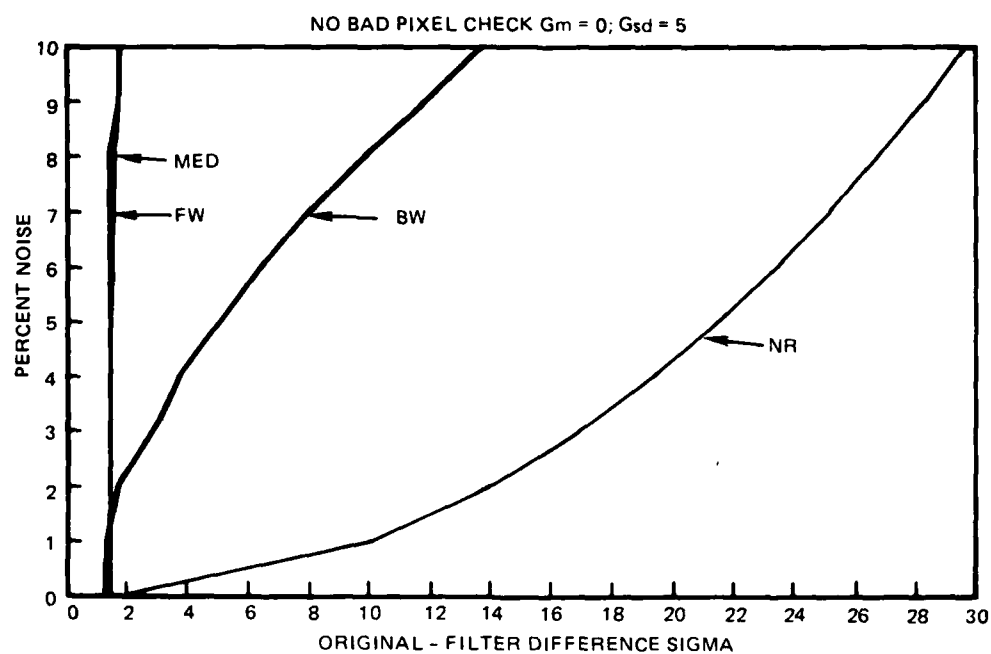


Figure 8. No bad pixel check versus bad pixel check (SF11DED.IMG)

3. At 3 to 5 percent noise, the four-way filter was usually the best, except occasionally the median filter outperformed the four-way filter using the scaled versions of the images.

4. From 6 to 30 percent the four-way filter outperformed all the other filters for both the scaled and raw images.

Generally, not using the bad pixel check (when the filter was applied over the entire image) the following results were true:

1. At 0 and 1 percent noise, the biweight filter performed the best for both raw and scaled images, but only by a slight amount over the median filter.

2. At 2 percent noise the median filter performed best for the raw images, and the biweight filter performed best for the scaled images. In the case of the ocean scene the four-way filter performed best for the raw image.

3. At 3 to 30 percent noise, the median filter performed best for four of the scaled images (bay, downtown San Jose, mountains, and Santa Cruz). The four-way filter shared best performance with the median filter in this range for two of the scaled images (North San Jose and ocean). For the raw images (San Francisco Bay and downtown San Jose), the median filter performed best. For four raw images (North San Jose, mountains, Santa Cruz, and ocean) the four-way filter shared with the median filter, in this range.

The results of best filter performance are shown in table I.

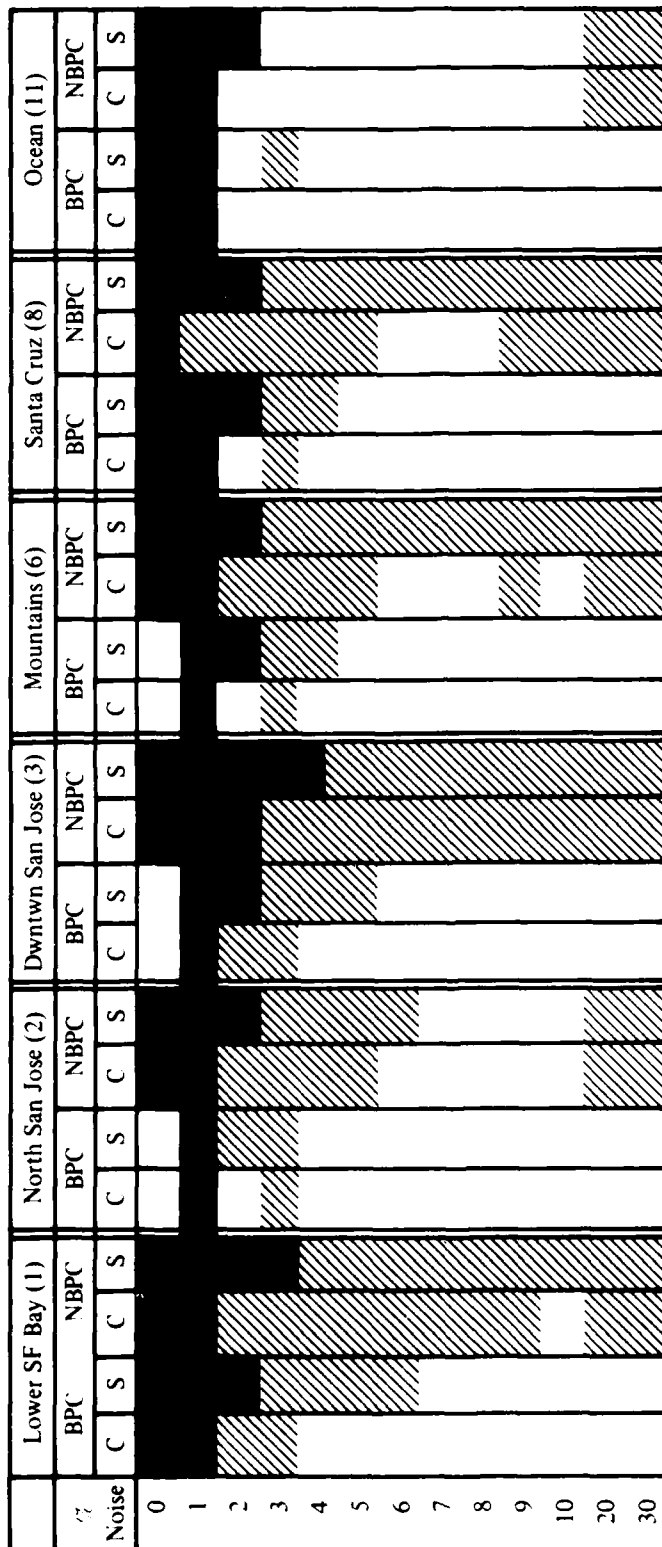
EFFECTS OF NOISE UPON FILTERING ACTION

Figure 9 shows the effects of increasing noise upon pixel replacement for the bad pixel check. Though this plot is for the ocean image, the remaining images have very similar plots. The data for the no bad pixel check (when the filter is applied over the entire image) are not worth considering since the number of pixels replaced are 204,304 for the biweight, median, and neighborhood replacement filters, and, though the numbers vary, between 431,000 and 618,000 for the four-way filter.

The interesting aspect of figure 9 is that it clearly depicts the actions of the filters throughout the noise range of 0 to 30 percent. The key points are 0 and 10 percent. At 0 percent noise, all of the filters perform filtering when in reality, none is required. This of course means that this tendency of a filter to replace good pixels is going to persist to some degree as noise is added to the image. Between 0 and 10 percent noise contamination, the number of pixels replaced by most of the filters appears to approach the true number of noisy pixels, but this is merely an illusion, for during the gradual introduction of noisy pixels, the respective filters continue to replace good as well as bad pixels, presumably to a lesser degree.

Up to 20 percent noise contamination, most of the filters appear unable to keep up with the increasing noise contamination, and though they replace more pixels in absolute number, it is apparent that a good deal of the noisy pixels are not being replaced, except for the four-way median filter.

Table I. Best filter performance.



BPC = Bad Pixel Check NBPC = No Bad Pixel Check C = Compressed (Raw) Images S = Scaled (0-255) Images

 = Biweight Filter
  = Median Filter
  = Four-way Median Filter

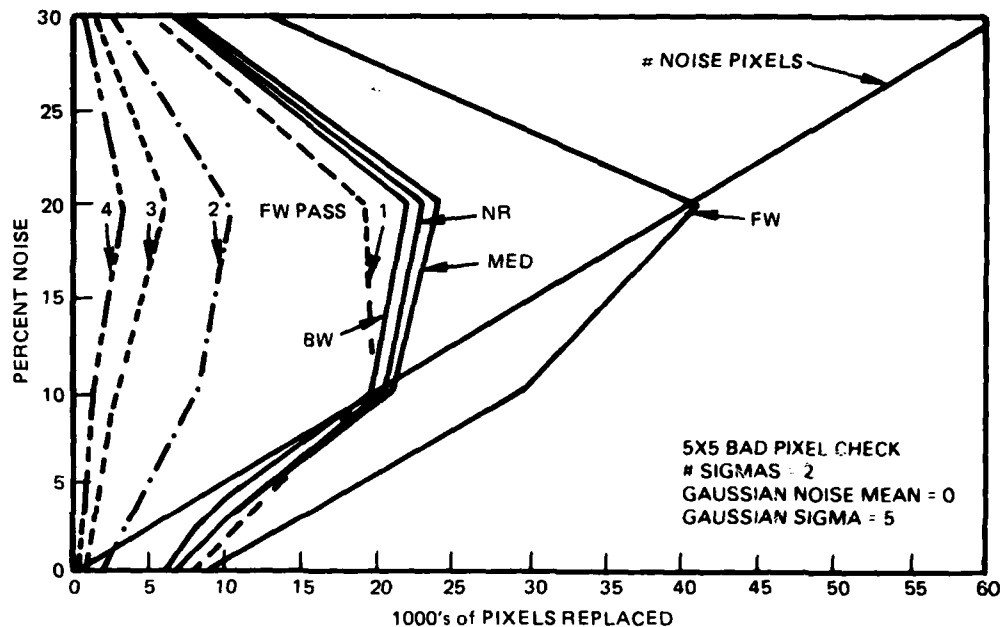


Figure 9. Pixel replacement (SF11.IMG).

From 20 to 30 percent the filters show strong signs of degradation. The noise is beginning to strongly degrade the bad pixel check which allows the retention of the noisy pixels.

The four-way median filter deserves special discussion since its method of application is unique. In figure 9, the four-way filter clearly does more pixel replacement than any of the others at every percentage of noise introduction, and at zero percent noise, the four-way filter replaces more good pixels than any other. As each pass of the four-way filter takes place, fewer and fewer pixels are replaced. From the shape of the four-way filter envelope in Figure 9, its general behavior is similar to the other filters, except that it performs more pixel replacement, even at the higher noise percentages.

The method of counting the number of replacements for the four-way filter is different than the method used for the other filters. For the other filters, if the bad pixel check indicates that the pixel needs replacement, the count of the replaced pixels are incremented, and then the filtering action is taken. For the four-way filter, if the bad pixel check indicates that the pixel requires replacement on one of the passes, the filtering action is taken (via the five IF statements), and if the pixel to be replaced is determined to be replaced by itself, then the count of the replaced pixels is not incremented. The replacement pixel count is only incremented when the pixel to be replaced is replaced with one of its two neighbors (vertical, horizontal, or on one of the diagonals).

STATISTIC FOR PERFORMANCE MEASUREMENT

The best statistic for measuring the performance of the filters was found to be the standard deviation of the difference between the original image and the filtered image after the filter had been applied to remove the noise. For every level of noise, each type of image, whether raw or scaled, and with and without the bad pixel check, the correlation coefficient and the sigma of the difference between the original and the filtered images were computed for each filter. The correlation coefficient is not as good a comparison statistic as the difference sigma. Hence the difference sigma was used to produce the plots shown in appendix A.

If the filtering action had removed all of the induced noise from all of the images, the mean and the standard deviation of the difference between the original and filtered images would have been zero, and the correlation coefficient would have been 1.00.

However, since at each percentage of noise replacement, there is residual noise after the filtering action, the mean of the difference is usually approximately zero, with a nonzero standard deviation, and a correlation coefficient less than 1.00. As increasingly more noise is placed into the images, and as filtering of noise decreases as shown in Figure 9, the mean of the difference between the original and the filtered images becomes more and more positive, due to the greater number of noisy pixels that remain in the filtered image.

FILTERING OF DIFFERENT IMAGES

With the exception of the ocean and cloud scene (SF11.IMG and SF11DED.IMG), most of the filtering actions upon the various images was similar. The four-way median filter fared better overall with the ocean and cloud scene than the other filters. All the filters had better performance with the ocean and cloud scene than the other images; the initial disruption of the zero-noise version was less, and from that point on with increased noise, every filter out-performed itself compared to its performance with the other images.

MAXIMUM PERFORMANCE EXPECTED FROM A FILTER

This study presented an opportunity to evaluate a given filter in order to determine its maximum possible filtering capability. Since the original images were offset from 0 by 300, since Gaussian noise was placed in the image in percentages from 0 to 30 percent, and since the value of noise added varied between -20 and +20, a very simple algorithm was employed to test the maximum potential of a given filter. Instead of employing the usual bad pixel check, in its place a check was substituted to determine if the candidate pixel was less than 300. If less than 300, it was noise and the filtering action was invoked; if not, the pixel was considered good, and filtering was by-passed.

Figure 10 compares the expected and actual performance of the biweight and neighborhood replacement filters. These two filters are grouped together because they essentially act the same. The falloff in performance as noise is increased is apparent both in the maximum possible and actual performance. Of course, the actual performance falls off more rapidly than the expected. The

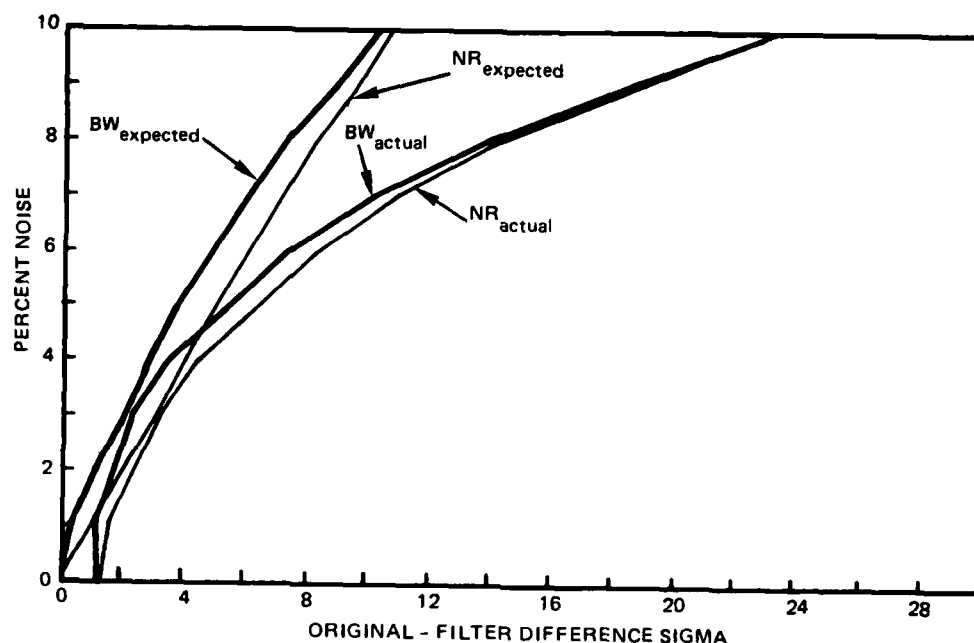


Figure 10. Actual/expected performance lower San Francisco Bay, biweight and neighborhood replacement.

closest area of coincidence is between 2 and 4 percent noise contamination, with the closest point at 3 percent.

Figure 11 compares the expected and actual performance of the median and the four-way median filters. These two filters are grouped together because they act in a similar manner. Both of these filters perform bad pixel replacement with actual image values, and not computed values as do the biweight and the neighborhood replacement filters. The maximum performance of both these filters is very fine and very similar. The actual performance, with the exception of the zero-noise offset, follows the slope of the expected performance very well until after 4-percent noise, when it degrades in a manner similar to the biweight and neighborhood replacement filters. The expected performance of these filters shown in Figure 11 is mirrored by the performance of these same filters when the bad pixel check is skipped and the filter is applied over the entire image, refer to appendix A.

When the four-way filter was run using the less-than-300 check to determine its maximum possible performance, it was noted that during the first pass, a greater percentage of pixels were replaced than were in the 452x452 image. This was due to the fact that the area for the first pass was 458x458 and thus contained more noisy pixels. On the subsequent passes, lesser and lesser amounts of pixel replacement occurred, indicating that not all pixel replacements were made with good pixels, but as the noise pixels were gradually eliminated, the fewer remaining had a better chance of being replaced by a good scene pixel.

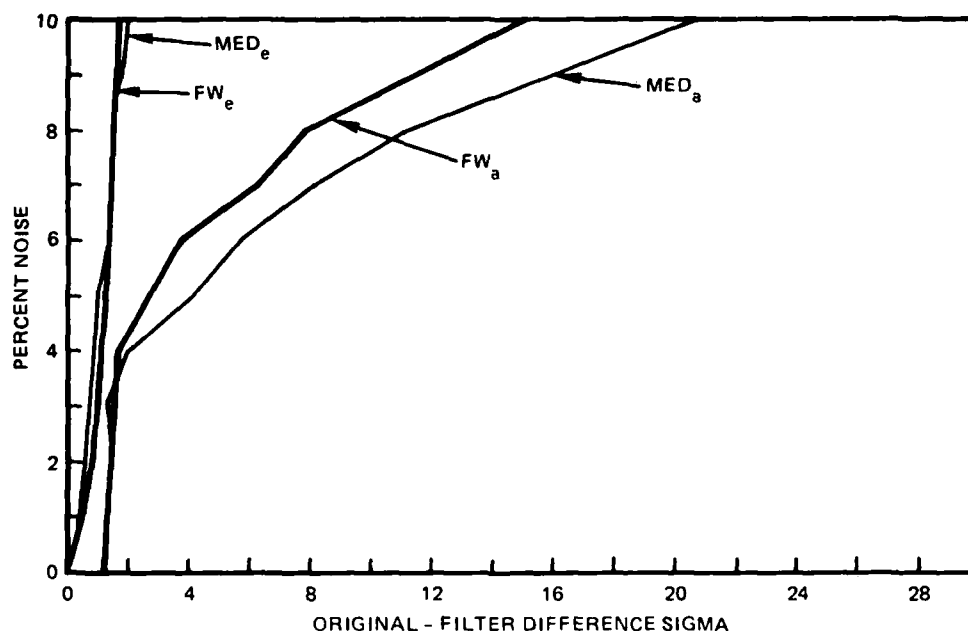


Figure 11. Actual/expected performance lower San Francisco Bay, median and four-way median.

IMAGE DISTORTION WITH ZERO-NOISE

The effectiveness of the various filters evaluated in this study depends also upon the amount of distortion caused by the filter at zero-noise, for if any filter is to perform well, it must have minimal distortion on the image when there is no noise in the image. If there is a great deal of image distortion caused by the filtering action when there is no noise, then this tendency will continue regardless of the amount of noise actually in the image. An analysis was made of the zero-noise level of filtering for all of the images (raw and scaled) and for the bad pixel check versus the no bad pixel check. The sigmas of the difference between the original and the filtered images were used as the criteria. The sigmas for all of the filters at zero-noise were averaged and were used to construct table II. The average of the zero-noise values was used because for all of the filters, on any one image, the sigmas for each of the four filters were very close together in value, as can be seen from the plots in appendix A. In most cases the deviation was only .01 to .66 in magnitude. The individual filters were also ranked by their ability to minimize zero-noise image distortion.

Table II. Filter image distortion with zero-noise.

IMAGE	BAD PIXEL CHECK					NO BAD PIXEL CHECK			
	Scaled Sigma	Order	Raw Sigma	Order		Scaled Sigma	Order	Raw Sigma	Order
SF1	3.61	1 2 3=4	1.2	1 2 3 4		10.8	1 2 3 4	3.59	1 2 4 3
SF2	2.89	4 1 2 3	.97	4 1 2 3		8.03	1 2 4 3	2.7	1 2 4 3
SF3	7.1	4 1 2 3	2.36	4 1 2 3		17.62	1 2 4 3	5.88	1 2 4 3
SF6	3.75	4 1 2=3	1.48	4 1 3 2		9.13	1 2 4 3	3.65	1 2 4 3
SF8	3.37	1 4 3 2	.84	1 4 3 2		8.06	1 2 4 3	2.01	1 2 4 3
SF11	1.44	1 4 2 3	.51	1 2 4 3		3.92	1 3=4 2	1.35	1 3 2=4

Biweight=1, Median=2, Neighborhood Replacement=3, Four-way Median=4

All sigma values are averaged for all four filters.

One outstanding characteristic of all of these filters is that they all have the same propensity of image distortion at zero-noise. Even though the biweight filter was better than the rest, its performance in terms of the others was not that much better; the ranking is only an artifact of comparing the absolute values, and being forced to select one as greater than or less than another. In view of this comment, the performance of the filters at zero-noise is (1) biweight, (2) median, (3) four-way median, and (4) neighborhood replacement, from best to worst.

At zero-noise, the effect of not performing the bad pixel check is to introduce more filter distortion into the image, however this tendency is not so pronounced when comparing the raw images. At zero-noise approximately 3 percent of the pixels are distorted from the original and if as mentioned earlier, some of this distortion is actually removing noise that was present in the original image, then this concern of distortion at zero-noise would become lessened. However, when the filter is applied to the entire image (without the bad pixel check), the median and four-way median filters do tend to follow their maximum possible filtering curve better than when the bad pixel check is performed.

IMPROVED BAD PIXEL CHECK

The bad pixel check was improved by increasing the neighborhood to 11x11 and using various number of sigmas for increasing amounts of induced noise as shown in table III.

Table III. Improved bad pixel check (11x11 neighborhood).

Noise (%)	# of Sigmas	Noise Pixels	Noise Replaced	Good Disturbed
0	4.1	0	0	14
1	3.5	2076	2076	24
2	3.5	4074	4074	14
3	3.5	6021	6021	6
4	3.1	8035	8035	19
5	3.0	10,069	10,068	11
6	2.9	12,084	12,079	5
7	2.8	14,056	14,047	5
8	2.7	16,068	16,061	3
9	2.6	18,075	18,067	3
10	2.5	20,110	20,105	5
20	1.5	40,259	40,259	0
30	1.3	60,716	60,712	0
40	1.0	81,424	81,424	16

Tests were run on the raw DAEDALUS images downtown San Jose (SF3DED) and ocean and clouds off Santa Cruz (SF11DED). These are respectively the most and least cluttered of the images. The results are shown in figure 12 for downtown San Jose, and in figure 13 for ocean and clouds off Santa Cruz. The improvement in both images over the performance using the bad pixel check with a 5x5 neighborhood and constant number of sigmas = 2 is substantial. The unevenness of the plots between 5- and 10-percent noise could be eliminated by fine-tuning the selection of number of sigmas, because, as table III shows, not all of the noise was removed. Residual noise contributes more to increasing the difference sigmas (difference between the original and filtered images) than does the small amount of good pixels disturbed.

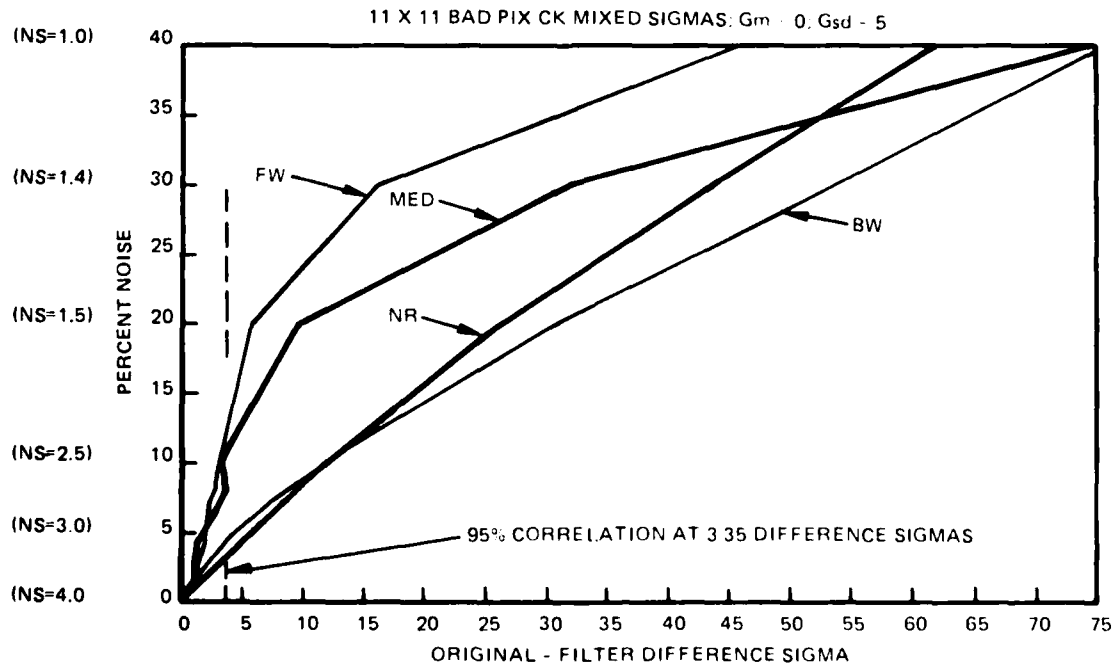


Figure 12. Improved filter performance (SF3DED.IMG).

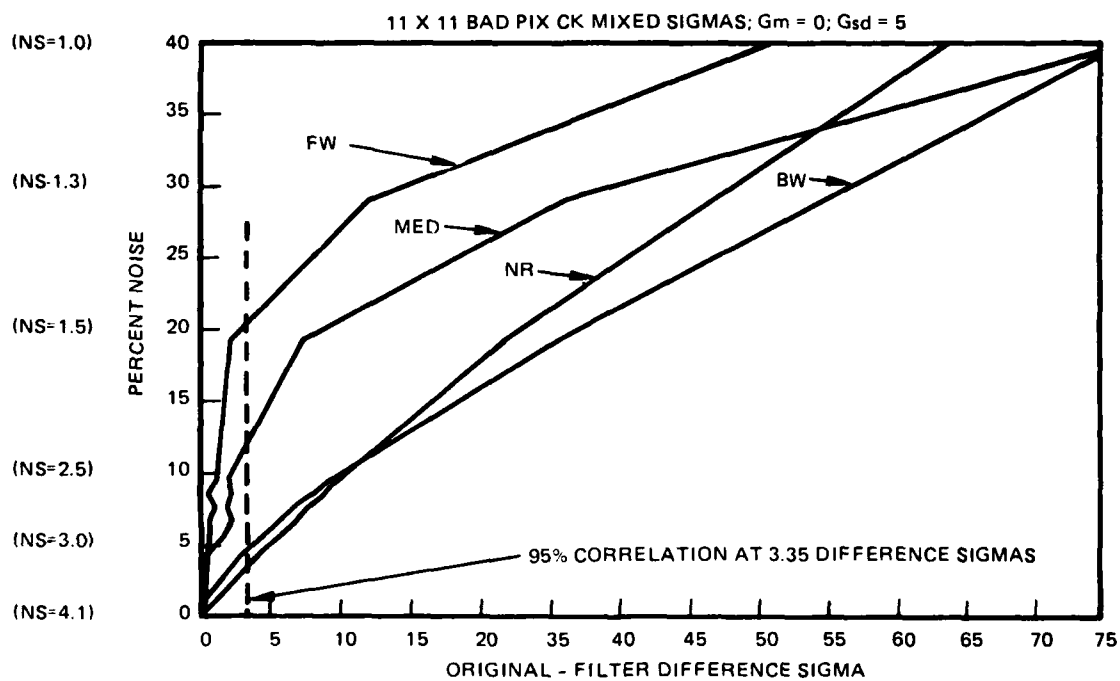


Figure 13. Improved filter performance (SF11DED.IMG).

SUMMARY OF FILTER PERFORMANCE

The major consideration in arriving at a final determination of filter performance is not to look at the statistics alone, but also to look at the final filtered image. All of the filters have a tendency to blur or distort the original in some manner. The manner of inducing noise in this study was not to add or subtract some constant (or even variable) to the pixels in the image. The actual pixel values were replaced with values that have no relevance to the original pixel value. The original pixel value is gone; it can never be completely recovered. The filtering techniques were evaluated to see how well the substituted value came close to the original pixel value.

Another consideration is the amount of noise that a filter is expected to replace. Even with levels of noise as high as 50 percent, it is surprising that a human being can still recognize the major content of an image. In this respect, all of the filtering techniques appear to be performing well. On the other hand, if restoration of an image is required to make intelligible some small 2x2 or 4x4 pixel area for identification purposes when 50 percent of the pixels are replaced by noise, then these types of filters would be inadequate, if indeed any would be adequate.

Several criteria have been selected to evaluate the filters in view of the above:

1. The potential of the filter at different levels of noise
2. The effectiveness of the filter at different levels of noise

FILTER POTENTIAL

Figures 10 and 11 show that the maximum potential for the four filters is limited assuming that a bad pixel check can be designed which will precisely identify only the bad pixels, and if these are compared, the ranking of the filters would be (1) median, (2) four-way median, (3) biweight, and (4) neighborhood replacement. This is the best that the filters could perform. The expected performance is thus the best it could possibly be throughout the entire range of noise replacement, from 0 percent to 30 percent.

Viewing these plots shows that the consistency of maintaining a good replacement of noise with values close to perhaps the original pixel values is continuous throughout the range for the median and four-way median filters. On the other hand, the biweight and the neighborhood replacement filters fall off in potential performance as the amount of noise contamination increases. The median and the four-way median filters replace noise with actual original pixel values, whereas the biweight and the neighborhood replacement filters manufacture new pixel values that may never have existed in the original image.

FILTER EFFECTIVENESS

Until the bad pixel check was improved to reduce or eliminate zero-noise image distortion, the performance of all four filters throughout the entire range of 0 to 10 percent noise fell short of the expected performance, as shown by Figures 10 and 11. The four-way median and median filters performed

well in the 0- to 4-percent noise range, so they could be classed as the better filter.

Without the bad pixel check, clearly the median and four-way filters are more effective than the biweight and neighborhood replacement filters as shown in figure 8.

Figures 12 and 13 show that the median and four-way median filters outperform the biweight and the neighborhood replacement filters, and that the four-way median filter outperforms the median filter at noise levels higher than 10 percent. These plots were made using the improved bad pixel check discussed earlier.

SURVEY OF FILTERED IMAGES

A program was written to display the original image, the original image contaminated by noise, the noise mask, the filtered image, and the mask of the residual noise and good pixels disturbed by filtering. Due to the fact that in many cases residual noise remained in the filtered image, the image would have to be rescaled in order to display it. The rescaling with noise values from -20 to +20 distorted the filtered image compared to the original. It was decided to produce a synthetic representation of the filtering action. The original image was input from disk and displayed as is. Then another working image was built of the original image plus the 300 offset and plus noise. This was the image upon which the filter worked. The original image had the same noise pixels as the working image, but instead of values from -20 to +20 the values were what the operator selected, usually zero, which displayed as black. The original image with black noise was displayed, then a noise mask was displayed. The noise mask was built at the same time the black noise was added to the original image. The noise mask was black with white noise. Then the filtering action took place. As each pixel requiring replacement (as determined by the bad pixel check) was corrected in the working image, a check was made in the original image for presence of zero pixel values (indicating noise). If the pixel to be replaced was zero, then the count of noise replacement was incremented and the noise pixel in the mask was zeroed; if not zero the count of pixels disturbed was incremented and the pixel in the noise mask was made purple.

In either event the pixel in the original image was replaced with the replacement value in the working image minus 300. This was not the correct value, but served to indicate that a correction had been made, and also removed the zero (noise) values from the original. After filtering, the original with synthetic noise removal was displayed, and finally the residual noise mask was displayed. Pictures were taken of the sequences for different images and varying amounts of noise. The filter used for the filtering action was the median filter.

Figures 14, 15, and 16 show respectively 1-, 5-, and 50-percent noise masks. It is not likely that any images containing 50-percent noise would be processed, but figure 16 gives an idea of that amount of noise.



Figure 14. 1% noise mask.



Figure 15. 5% noise mask.

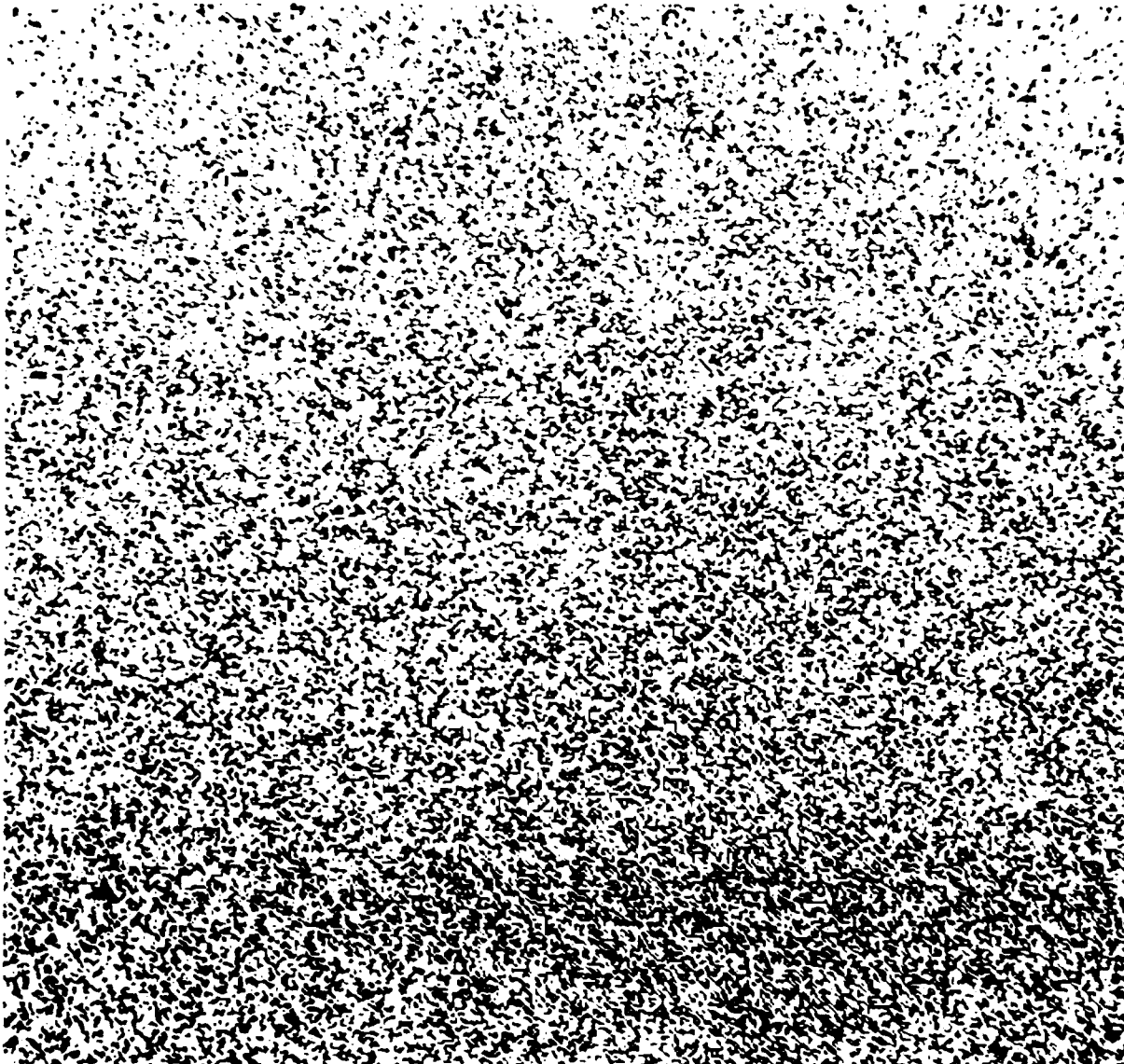


Figure 16. 50% noise mask.

Figure 17 shows 1-percent noise applied to the ocean and cloud image off Santa Cruz. The picture was dark so white noise was synthetically added. Below and to the right of the image center is a ship. Noise is not added to the edge of the image.

Figure 18 shows the results of filtering using a 3x3 neighborhood and number of sigmas = 2 for the bad pixel check. Original noise remains in the five-pixel border, which the filter does not cover, and some residual noise remains as seen by the white noise still in the image.

Figure 19 shows the residual noise mask. The large dots are the residual noise; the small dots are good pixels which have been disturbed. In black and white photos the difference between the residual noise and the good pixels that have been disturbed does not stand out too well.

Figure 20 shows 5-percent noise replacement in the lower San Francisco Bay image. The noise is in black.

Figure 21 shows the results of median filtering. The bad pixel check employed a 9x9 neighborhood and number of sigmas = 2.5. Of the 10,069 noisy pixels in the original, the filter removed 10,040, and disturbed only 24 good pixels. The residual noise mask for this filtering is shown in figure 22.

Figure 23 shows northern San Jose with 10 percent noisy pixels in black. Using a 9x9 neighborhood and number of sigmas = 2.0, 19,996 noisy pixels were replaced out of 20,110, and no good pixels were disturbed. The results of filtering are shown in Figure 24 and the residual noise mask is shown in figure 25.

Figure 26 shows Santa Cruz and the ocean with 30 percent black noise added. Using 9x9 neighborhood and number of sigmas = 1.0, 60,709 of the 60,719 noise pixels were replaced by the median filter, and only two good pixels were disturbed. The results of the filtering is shown in figure 27, and the residual noise mask is shown in Figure 28.

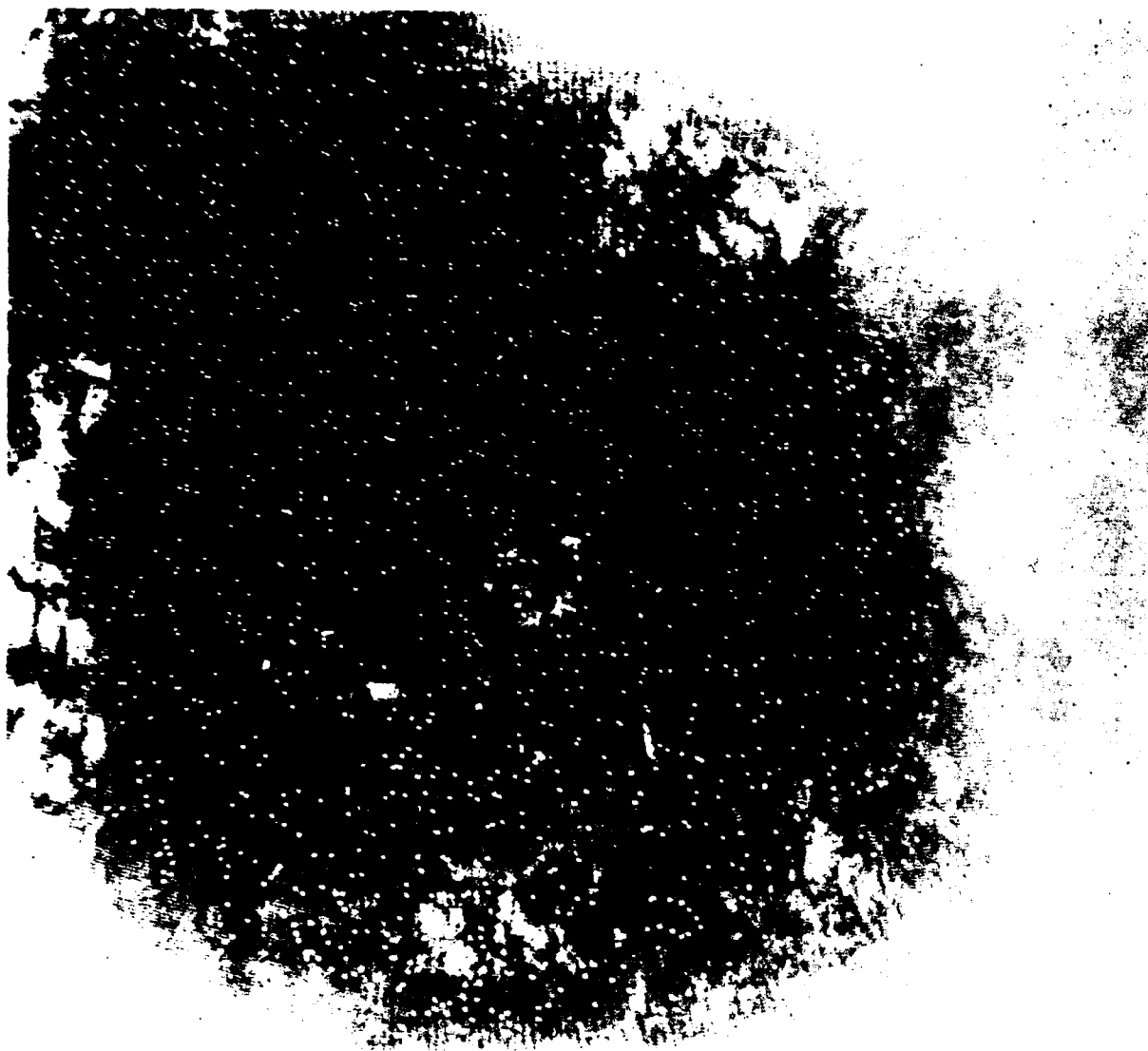


Figure 17. 1% noise on ocean and clouds off Santa Cruz.



Figure 18. Ocean and clouds off Santa Cruz median filtered.

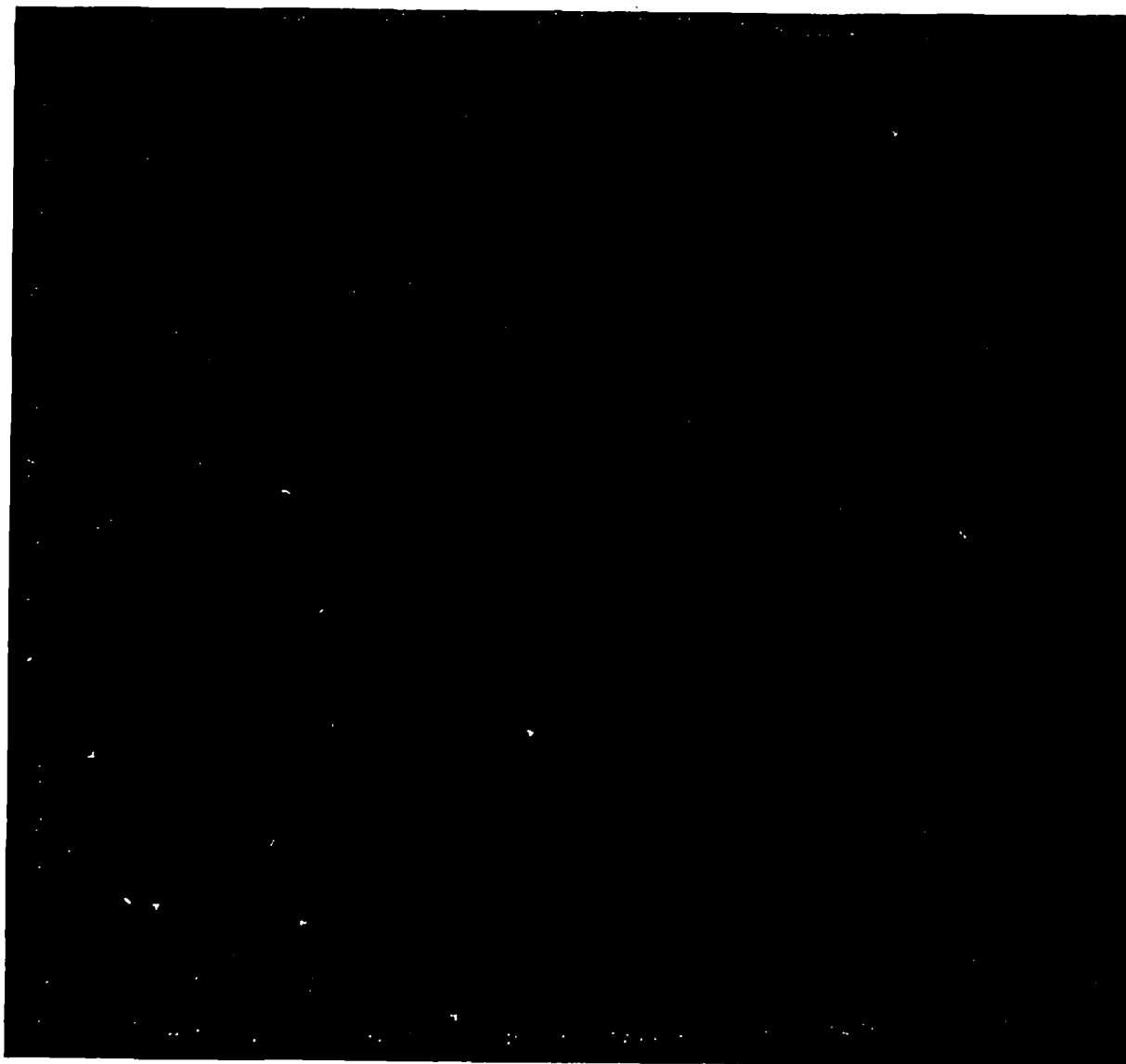


Figure 19. Residual noise mask for ocean and clouds after median filtering.



Figure 20. 5% noise on lower San Francisco Bay.



Figure 21. 5% noise median filtered on lower San Francisco Bay.



Figure 22. Residual noise after median filtering of 5% noise on lower San Francisco Bay.



Figure 23. 10% noise on northern San Jose.



Figure 24. 10% noise median filtered on northern San Jose.



Figure 25. Residual noise on northern San Jose.

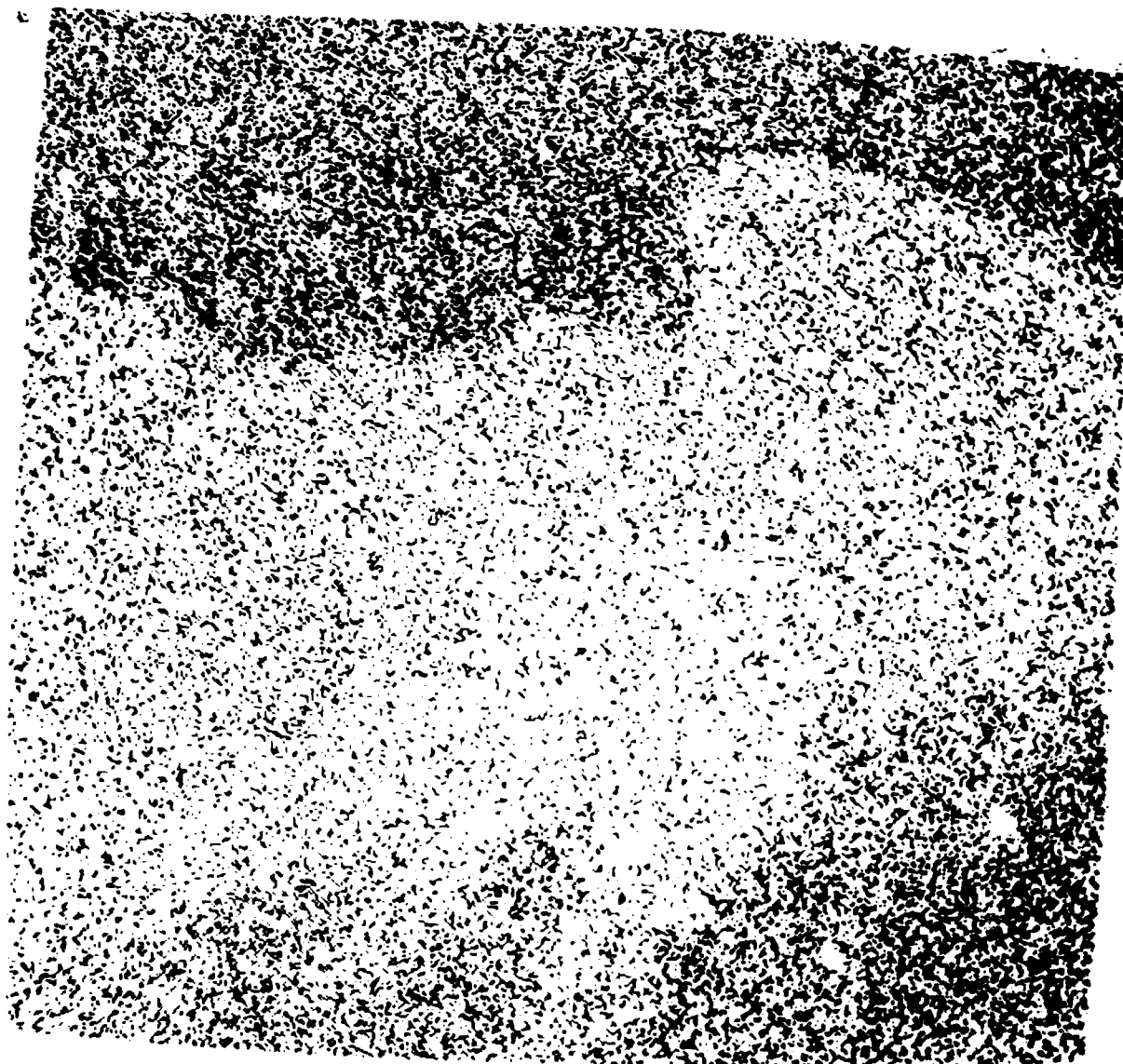


Figure 26. 30% noise on Santa Cruz.

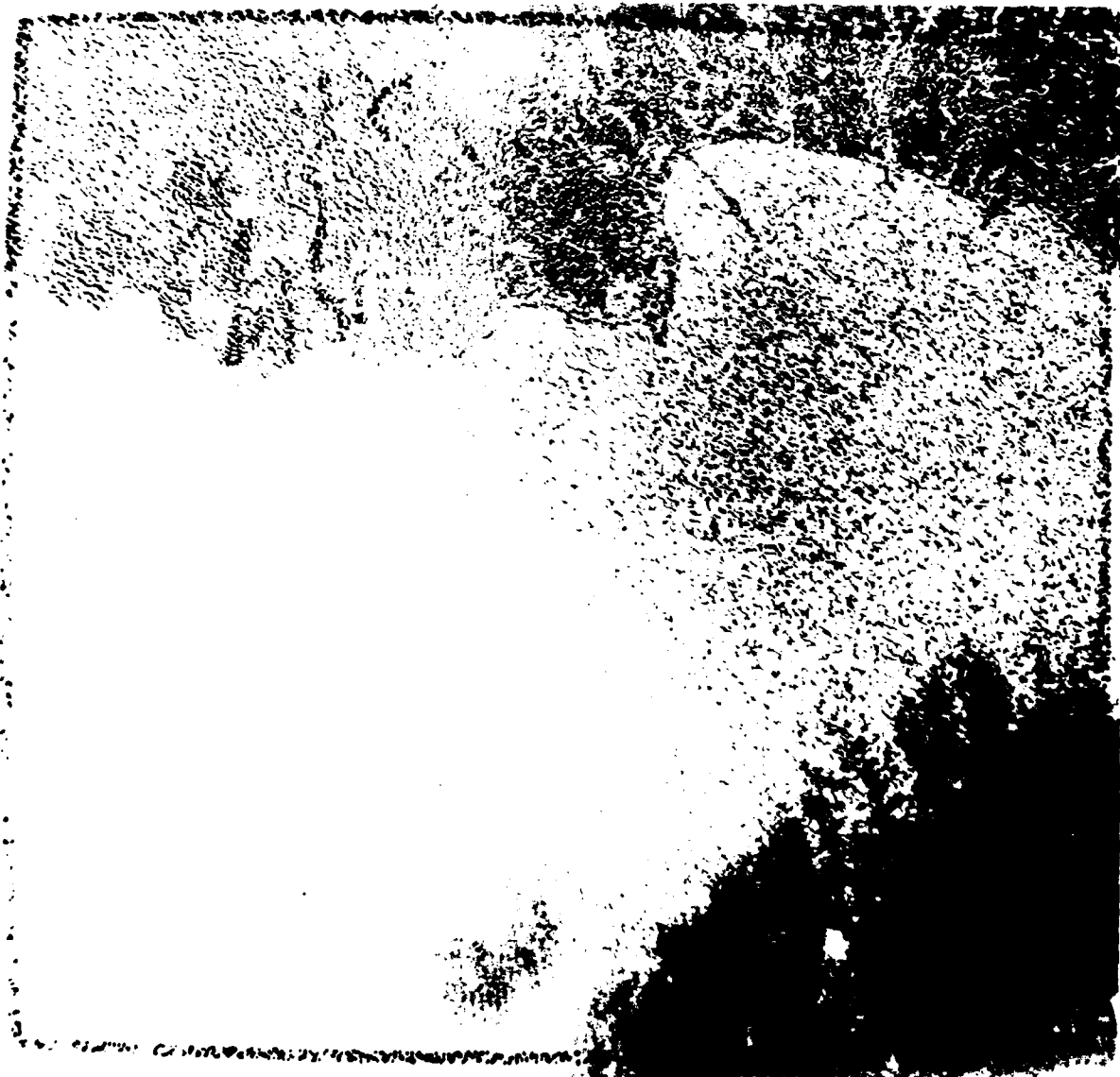


Figure 27. 30% noise median filtered on Santa Cruz.

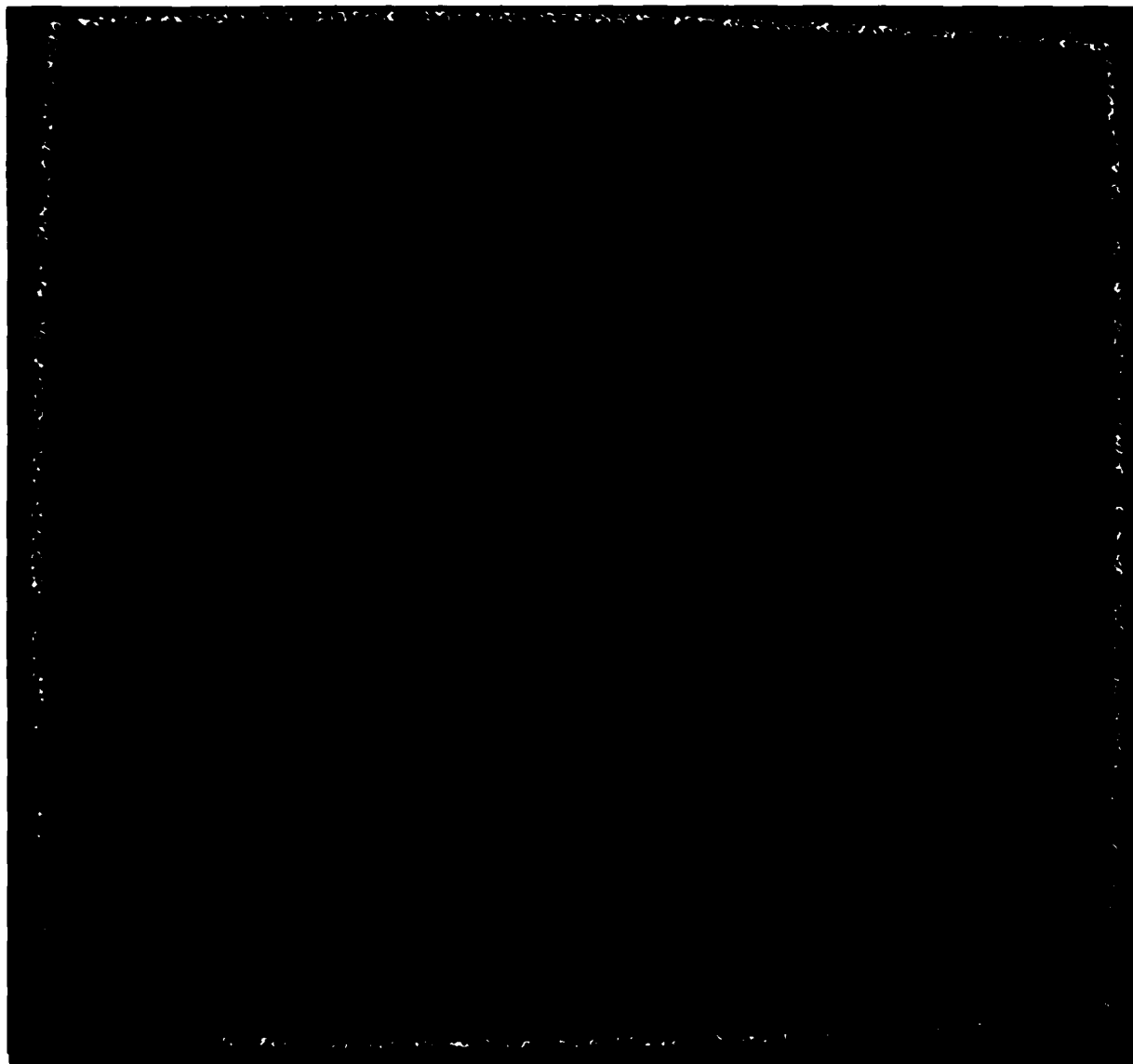


Figure 28. Residual noise on Santa Cruz.

RECOMMENDATION FOR FUTURE STUDY

Another improvement to the bad pixel check would be to build a bad pixel mask as shown in figure 29. This would be done by applying the bad pixel check to the entire image three times. On the first pass the bad pixel check would sum only the good pixels in its neighborhood for computation of the mean and sigma. Since the first time through, the pixels above and to the left of the candidate would have already been marked as a bad or good pixel, the bad pixels can be left out of the sum.

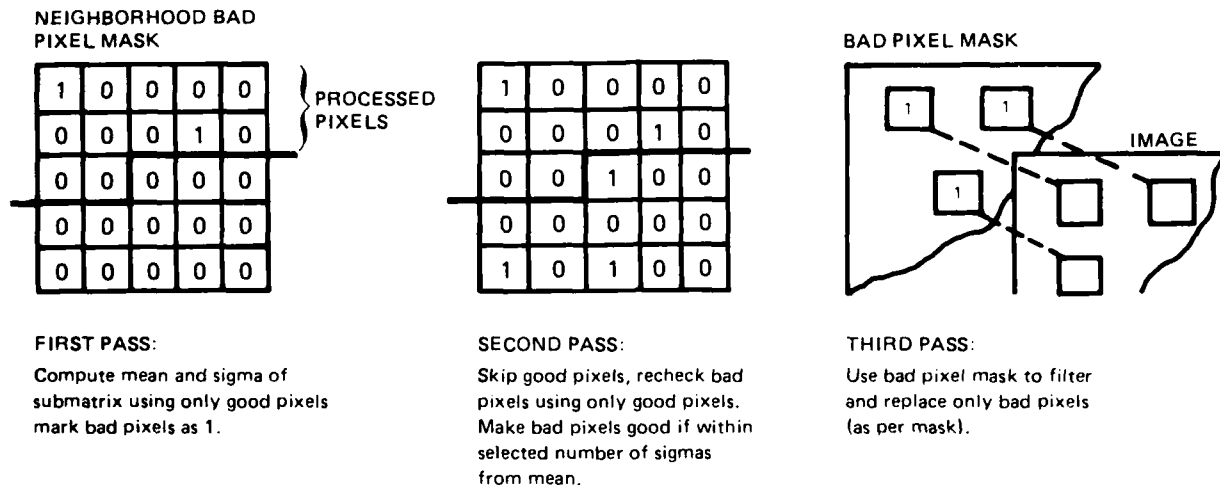


Figure 29. Bad pixel mask.

On the second pass, all bad pixels in the mask can be checked to see if they are really bad pixels; if not, they can be returned to the status of good pixels. The mean and sigma computations would be made using only the good pixels in the selected neighborhood. On the third pass, actual filtering can be carried out using a 3x3 neighborhood, but only the good pixels in that neighborhood would be processed to replace the marked bad pixels. On the third pass, only the bad pixel mask would be screened to determine which pixels require replacement. Time would be saved on the third pass, since the noisy pixels have already been tagged. The filters would use only the good pixels in the 3x3 neighborhood around the candidate pixel. Variations in filtering would have to be made for each filtering operation. The biweight filter would have to be revised to operate on less than nine pixels. The sort routine SHELLSORT is designed to handle any number of values.

The median filter would have to be revised to handle even numbers of pixels (2, 4, 6, or 8) by averaging the first and second (for n=2), the second and third (for n=4), the third and fourth (for n=6), and the fourth and fifth (for n=8); and using this average for replacement.

The neighborhood replacement filter would sum only the good pixels in its neighborhood and divide by the n of the good pixels.

The four-way median filter, on each of its passes, would not replace unless at least one good pixel was found in one of the two wings of the vertical, horizontal, or diagonal sets.

In all the filters, when a replacement is made, the bad pixel mask is updated by making the bad pixel good. This means that the replaced pixel can be used subsequently in the computations for replacing other bad pixels in its 3×3 neighborhood.

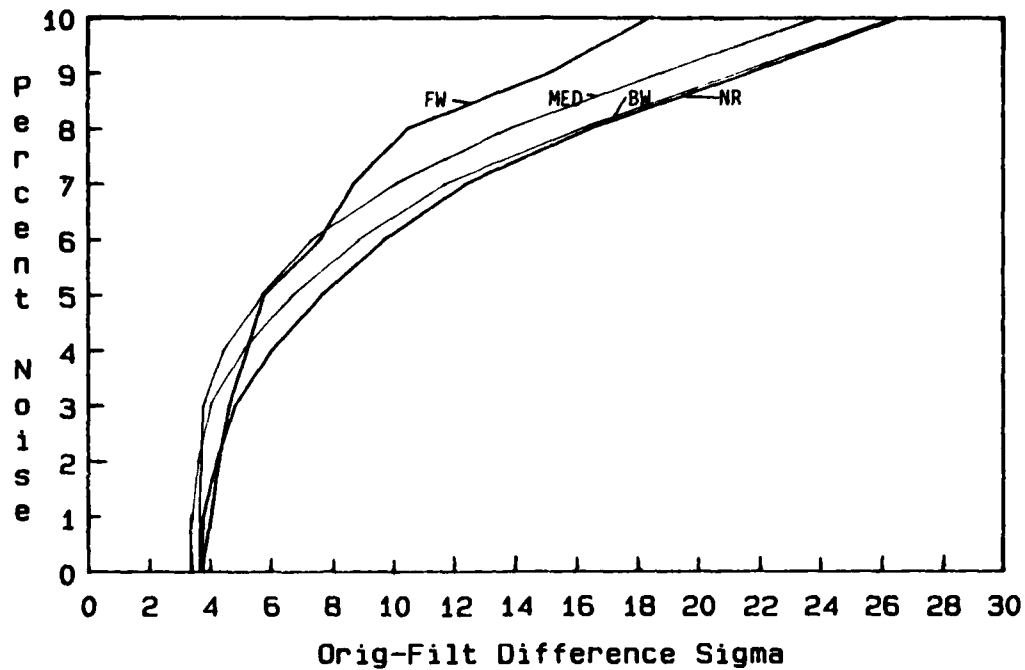
REFERENCES

1. A. Rosenfeld and A. C. Kak, "Digital Picture Processing", Second Edition, Academic Press, New York, Volumes 1 and 2, 1982.
2. J. G. Moik, "Digital Processing of Remotely Sensed Images", NASA SP-431, National Aeronautics and Space Administration, Washington, DC, 1980.
3. ___, "Picture Processing and Digital Filtering", T. S. Huang, Editor; Topics in Applied Physics Series, Volume 6, Springer-Verlag, New York, 1979.
4. ___, "Applications of Digital Signal Processing", A. V. Oppenheim, Editor; Prentice-Hall, New York, 1978, Chapter 4.
5. W. K. Pratt, DIGITAL IMAGE PROCESSING, John Wiley and Sons, New York, 1978.
6. H. C. Andrews and B. R. Hunt, "Digital Image Restoration", Prentice-Hall, New York, 1977.
7. N. G. Jerlov and E. S. Nielsen, "Optical Aspects of Oceanography", Academic Press, New York, 1974.
8. P. M. Narendra, "A Separable Median Filter for Image Noise Smoothing", Proceedings of the IEEE Computer Society Conference on Pattern Recognition and Image Processing, 31 May-2 June 1978, Chicago, Illinois, IEEE Publishing Services, New York, 1978, pp 137-141.
9. H. C. Andrews, "Monochrome Digital Image Enhancement", APPLIED OPTICS, Vol. 15, pp 495-503, 1976.
10. J. W. Tukey, "EXPLORATORY DATA ANALYSIS", Addison Wesley, Reading, Chapter 7, pp 205-236, 1976.
11. I. S. Reed, R. M. Gagliardi, private communications.
12. F. D. Groutage, private communications.
13. E. Winter, Technical Research Associates, private communications.

APPENDIX A. PLOTS OF RESULTS

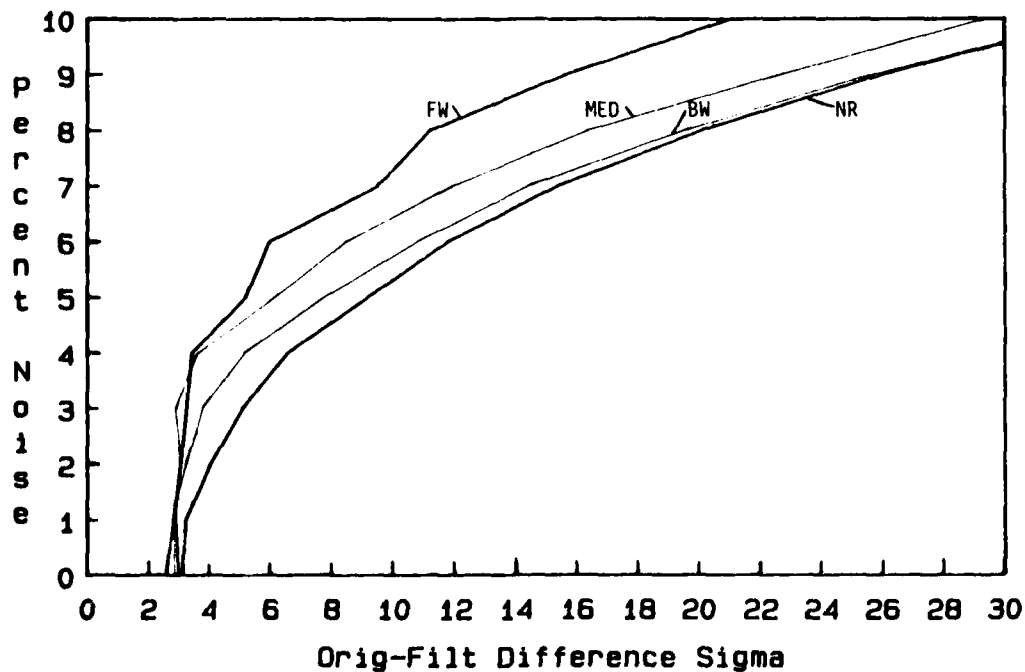
FILTER PERFORMANCE (SF1.IMG)

5x5 Bad Pix Ck=2 sig; Gm=0; Gsd=5



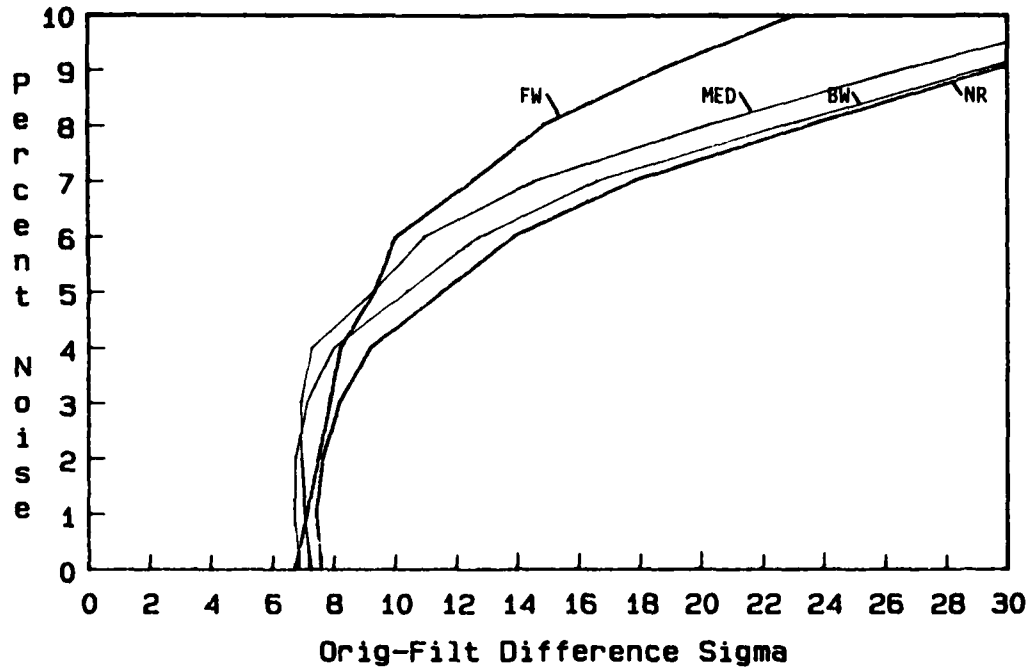
FILTER PERFORMANCE (SF2.IMG)

5x5 Bad Pix Ck=2 sig; Gm=0; Gsd=5



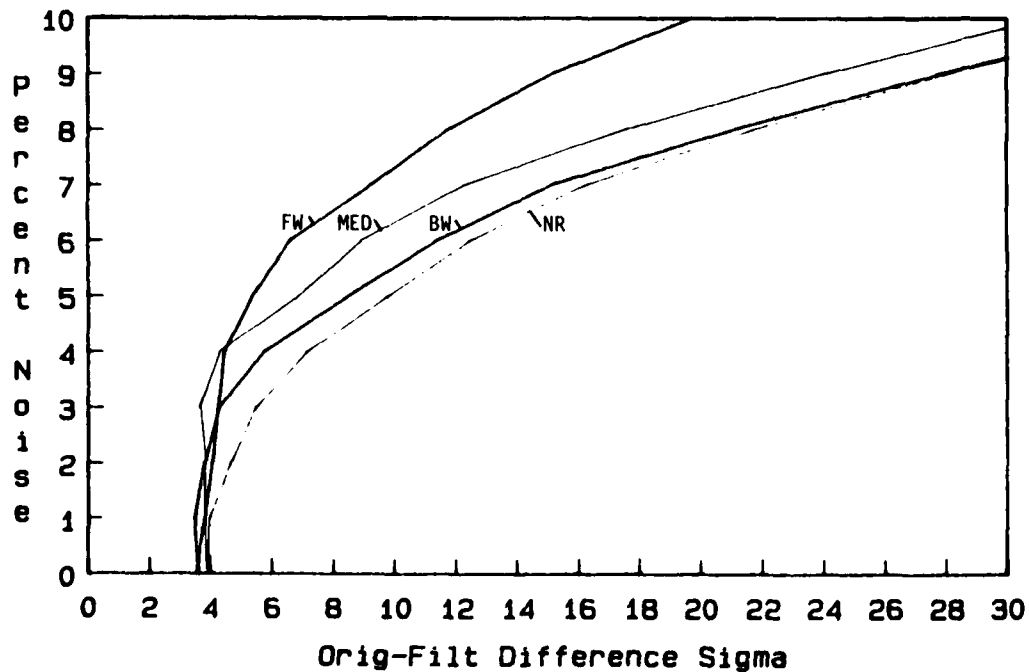
FILTER PERFORMANCE (SF3.IMG)

5x5 Bad Pix Ck=2 sig; Gm=0; Gsd=5



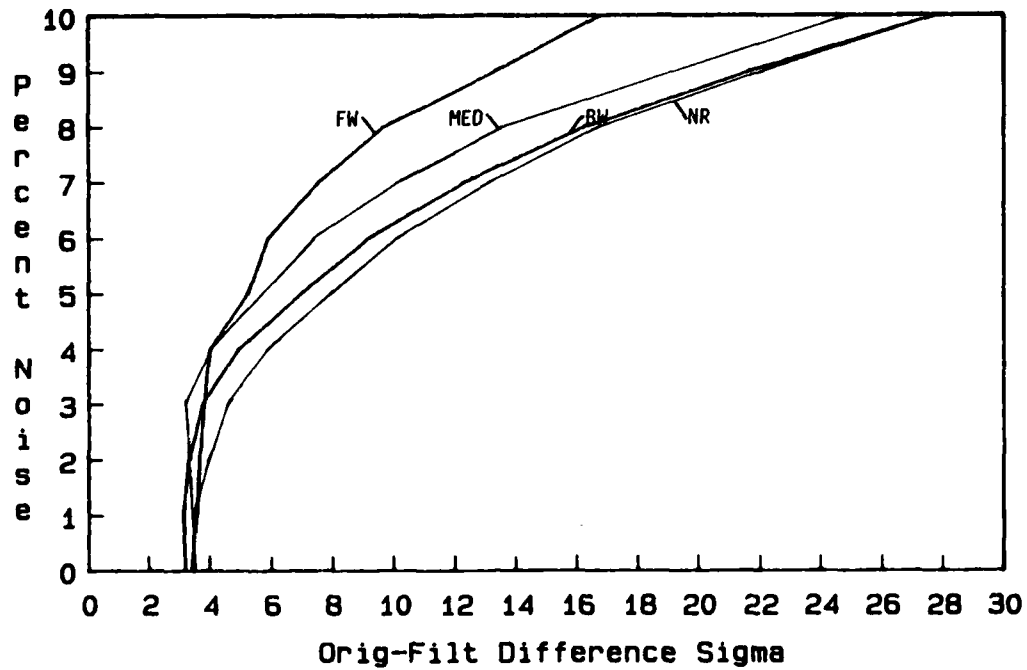
FILTER PERFORMANCE (SF6.IMG)

5x5 Bad Pix Ck=2 sig; Gm=0; Gsd=5



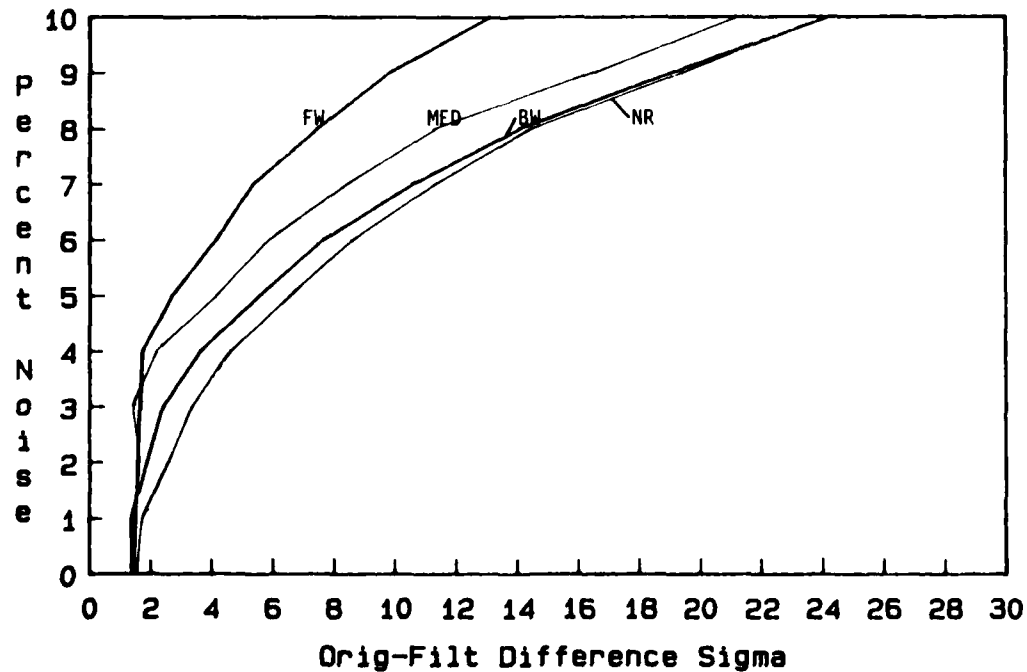
FILTER PERFORMANCE (SF8.IMG)

5x5 Bad Pix Ck=2 sig; Gm=0; Gsd=5



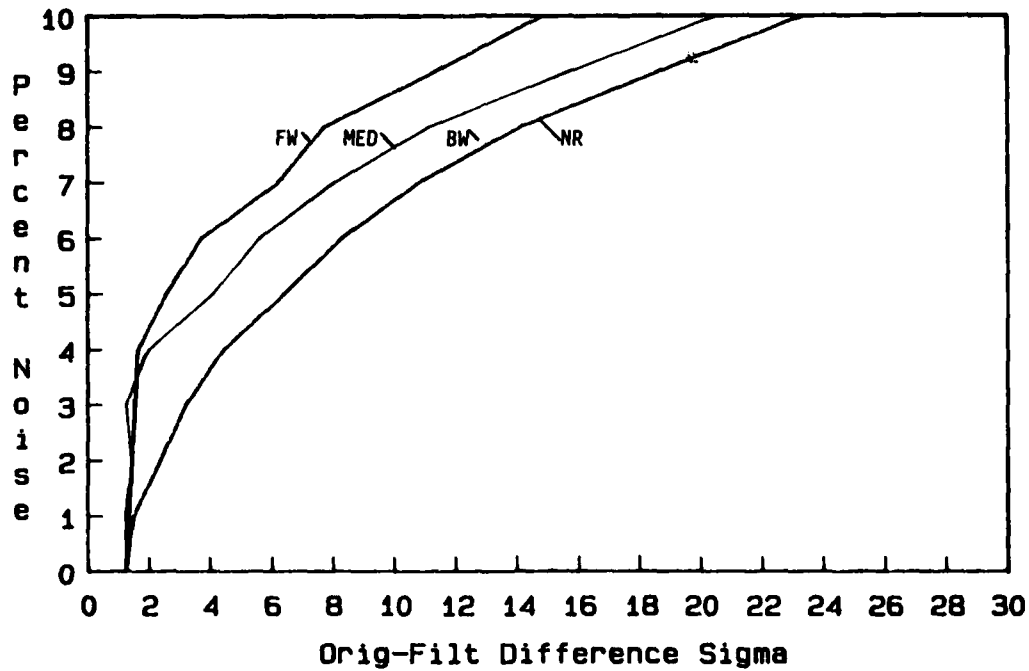
FILTER PERFORMANCE (SF11.IMG)

5x5 Bad Pix Ck=2 sig; Gm=0; Gsd=5



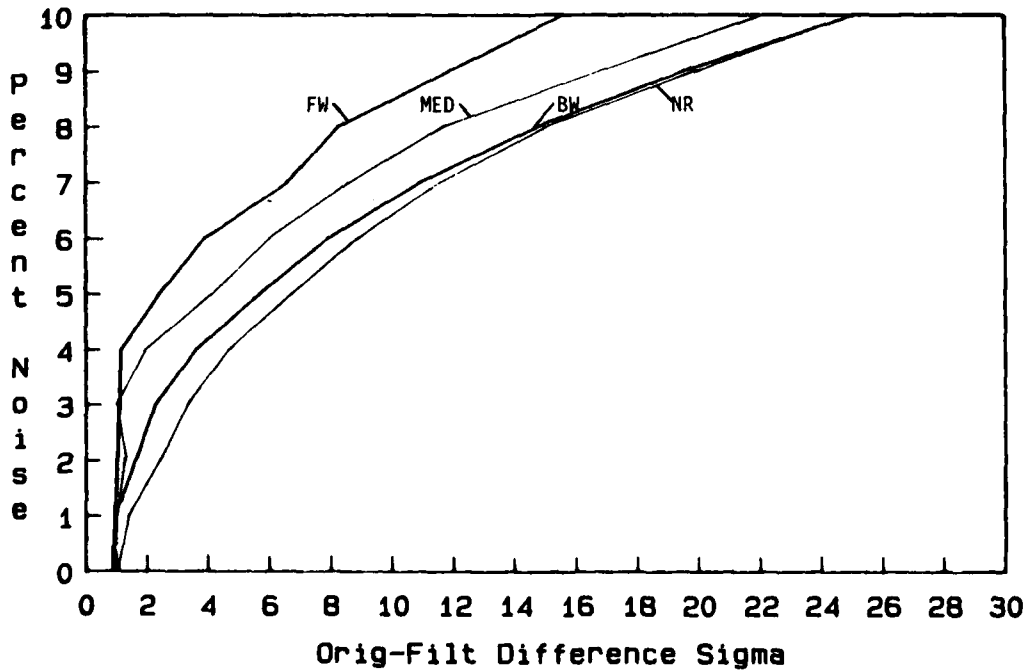
FILTER PERFORMANCE (SF1DED.IMG)

5x5 Bad Pix Ck=2 sig; Gm=0; Gsd=5



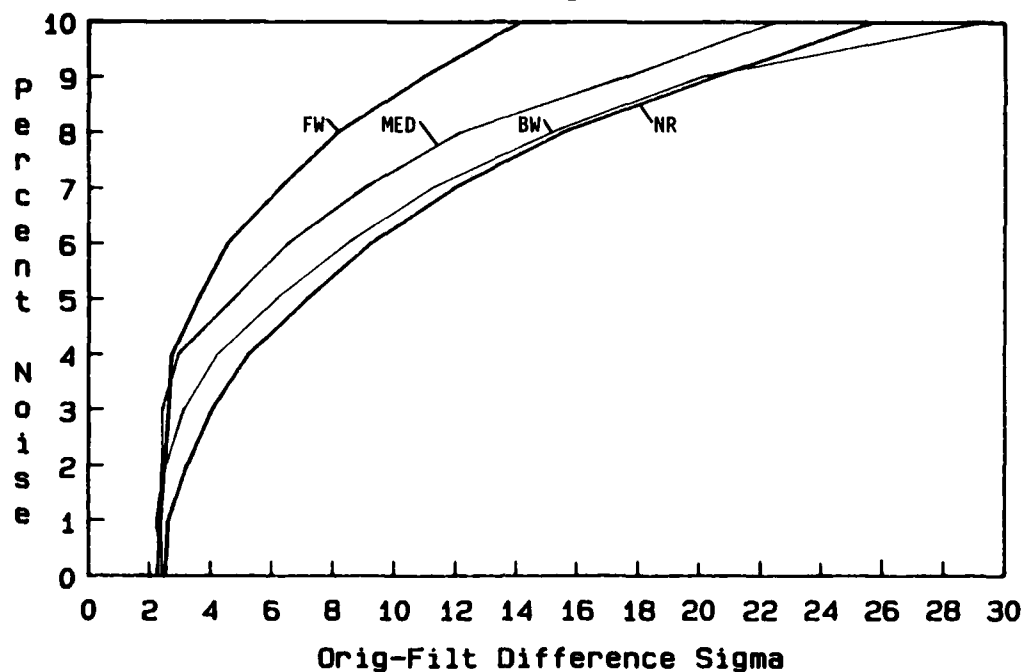
FILTER PERFORMANCE (SF2DED.IMG)

5x5 Bad Pix Ck=2 sig; Gm=0; Gsd=5



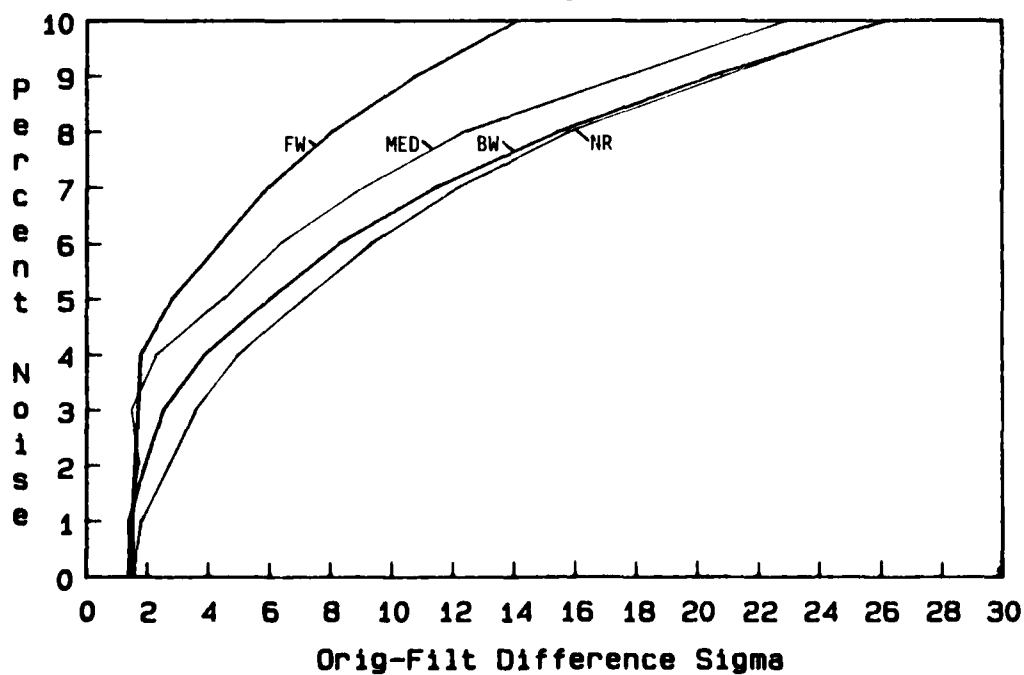
FILTER PERFORMANCE (SF3DED.IMG)

5x5 Bad Pix Ck=2 sig; Gm=0; Gsd=5



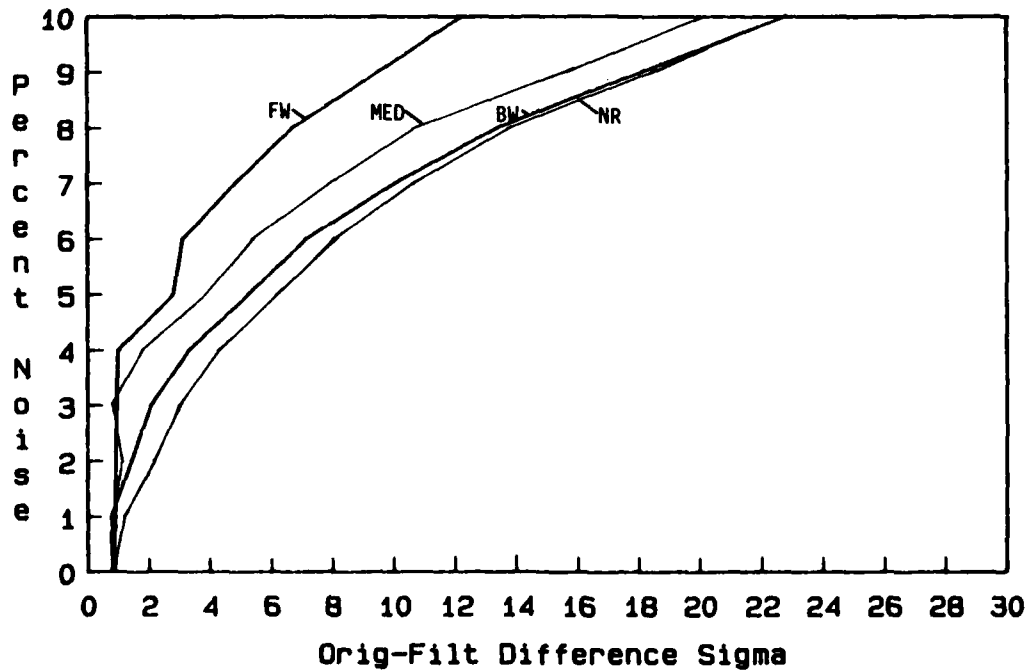
FILTER PERFORMANCE (SF6DED.IMG)

5x5 Bad Pix Ck=2 sig; Gm=0; Gsd=5



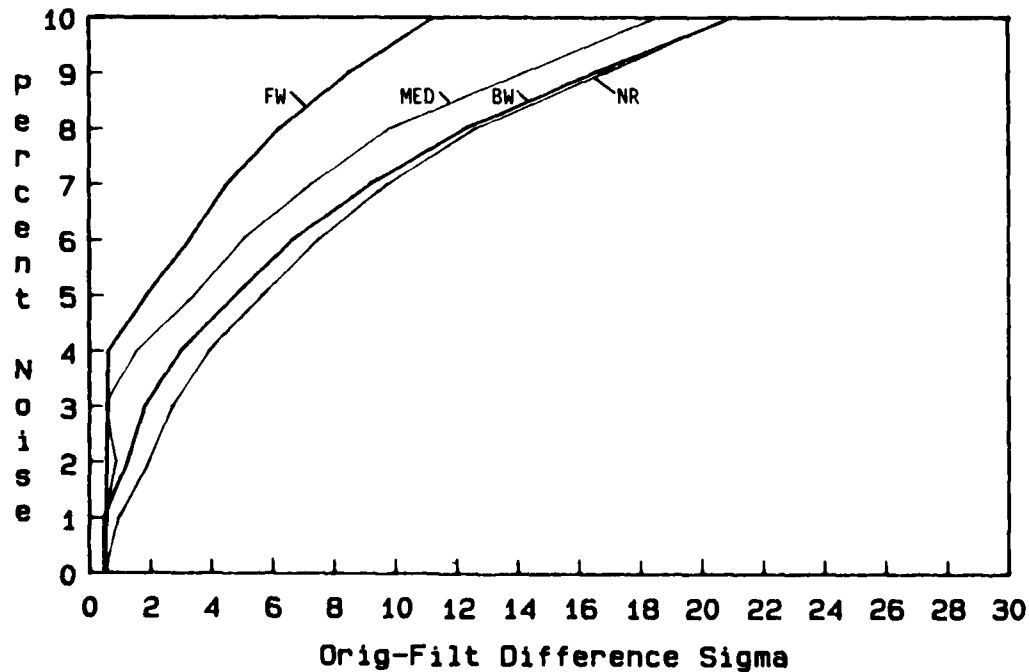
FILTER PERFORMANCE (SF8DED.IMG)

5x5 Bad Pix Ck=2 sig; Gm=0; Gsd=5



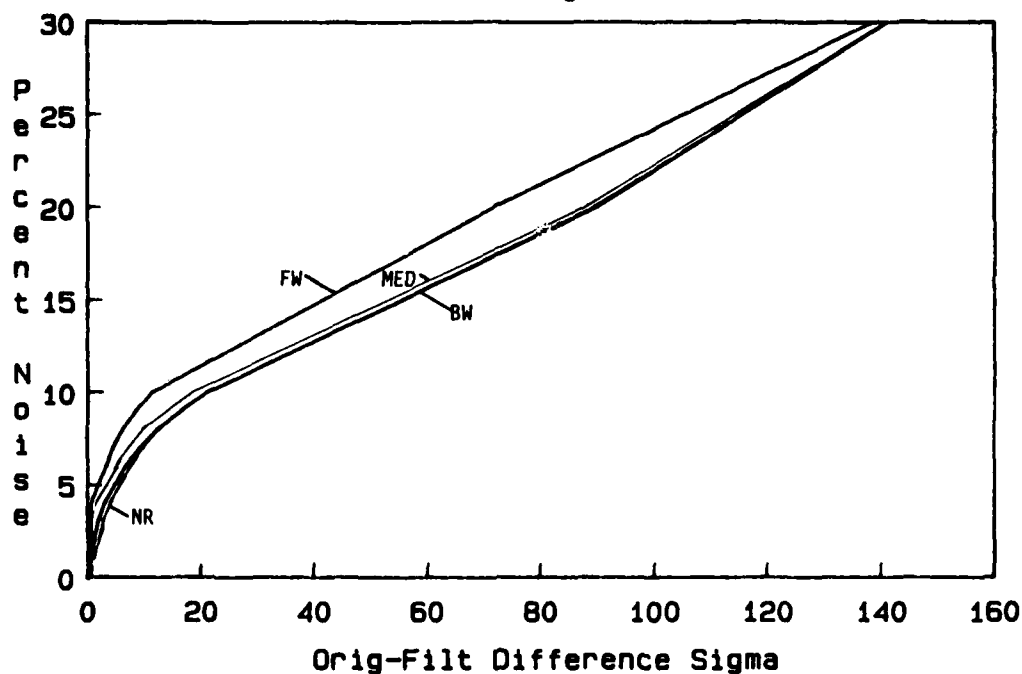
FILTER PERFORMANCE (SF11DED.IMG)

5x5 Bad Pix Ck=2 sig; Gm=0; Gsd=5



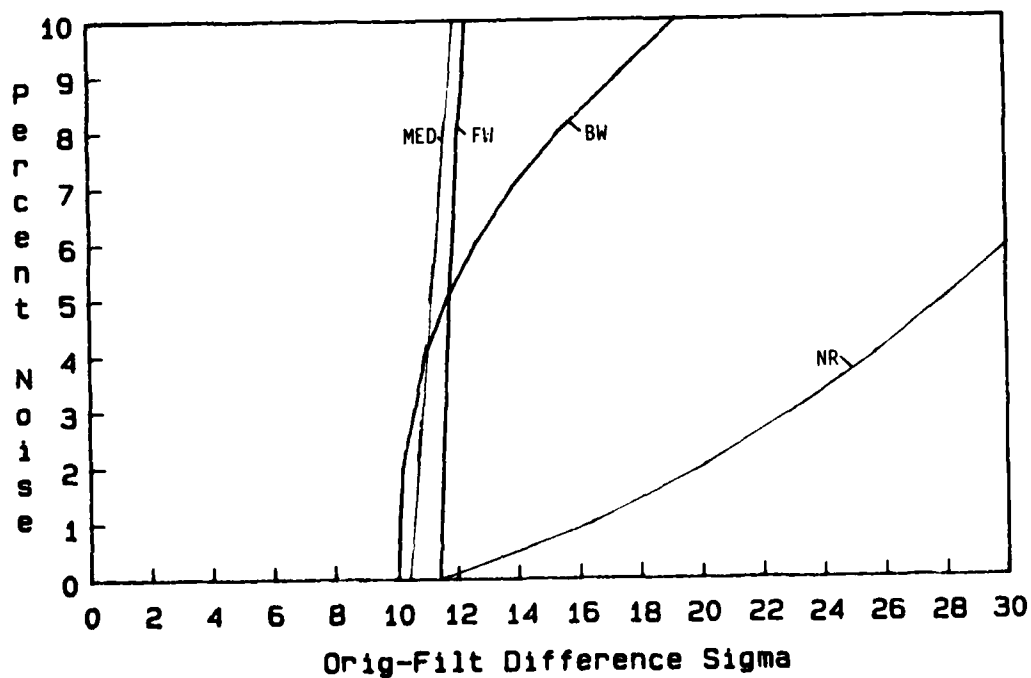
FILTER PERFORMANCE (SF11DED.IMG)

5x5 Bad Pix Ck=2 sig; Gm=0; Gsd=5



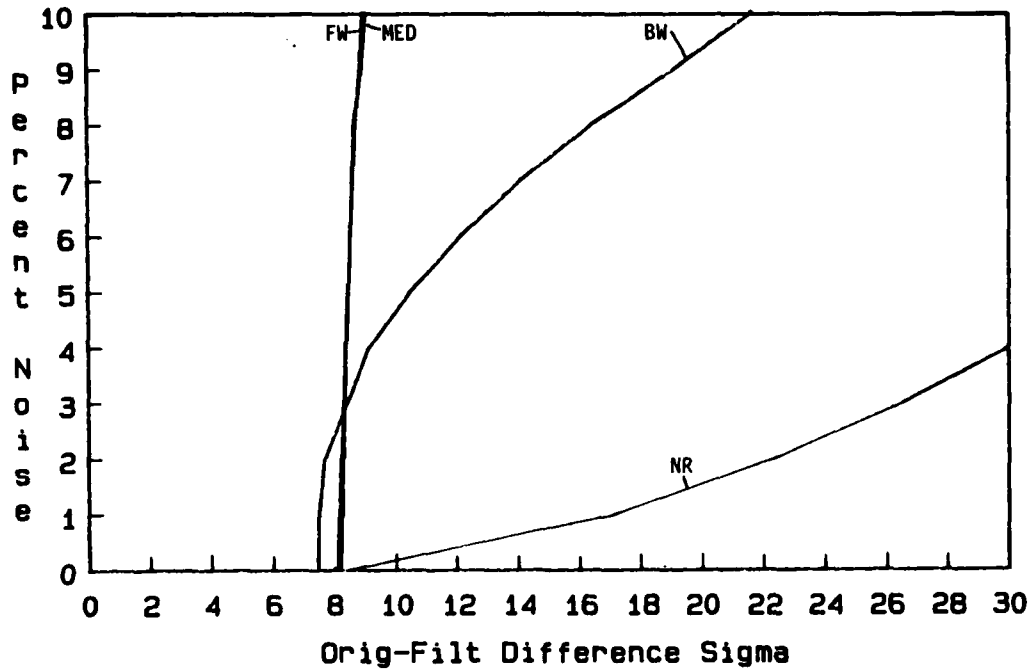
FILTER PERFORMANCE (SF1.IMG)

No Bad Pixel Check Gm=0; Gsd=5



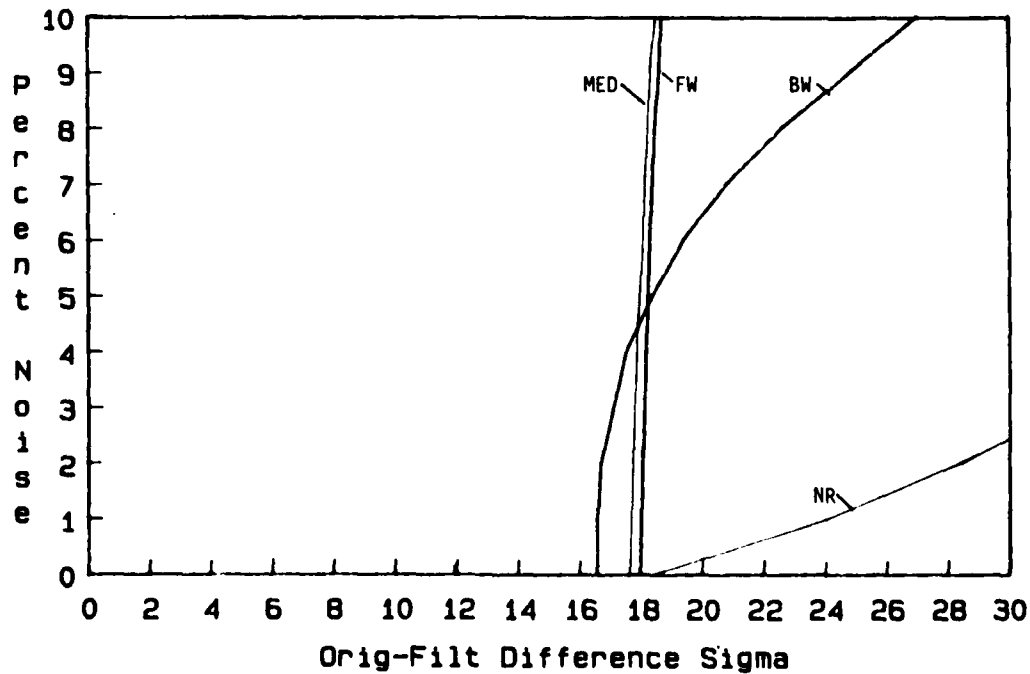
FILTER PERFORMANCE (SF2.IMG)

No Bad Pixel Check Gm=0; Gsd=5



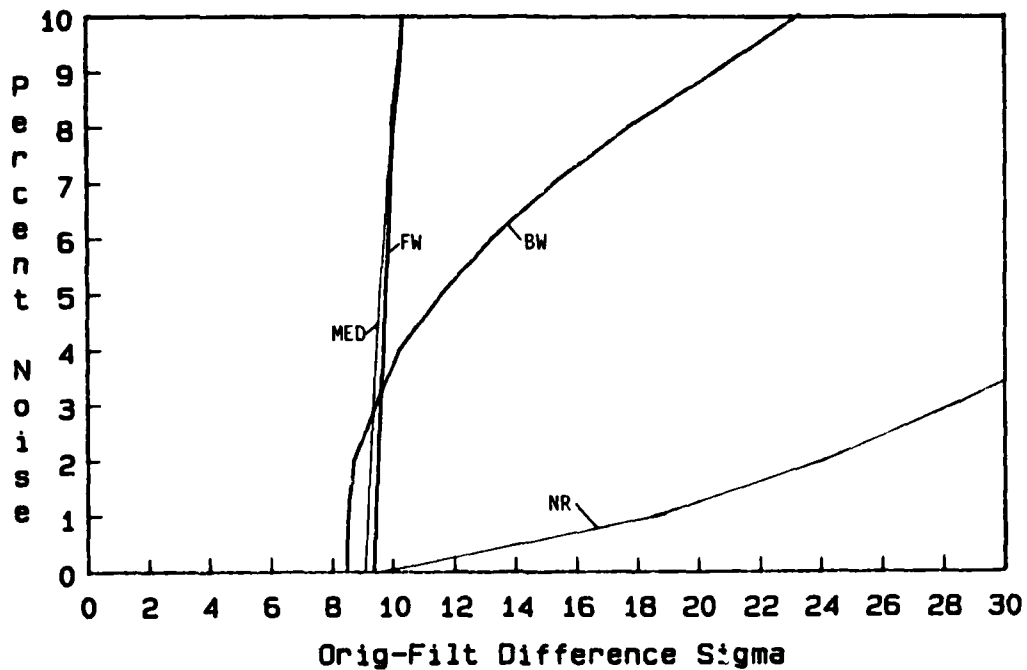
FILTER PERFORMANCE (SF3.IMG)

No Bad Pixel Check Gm=0; Gsd=5



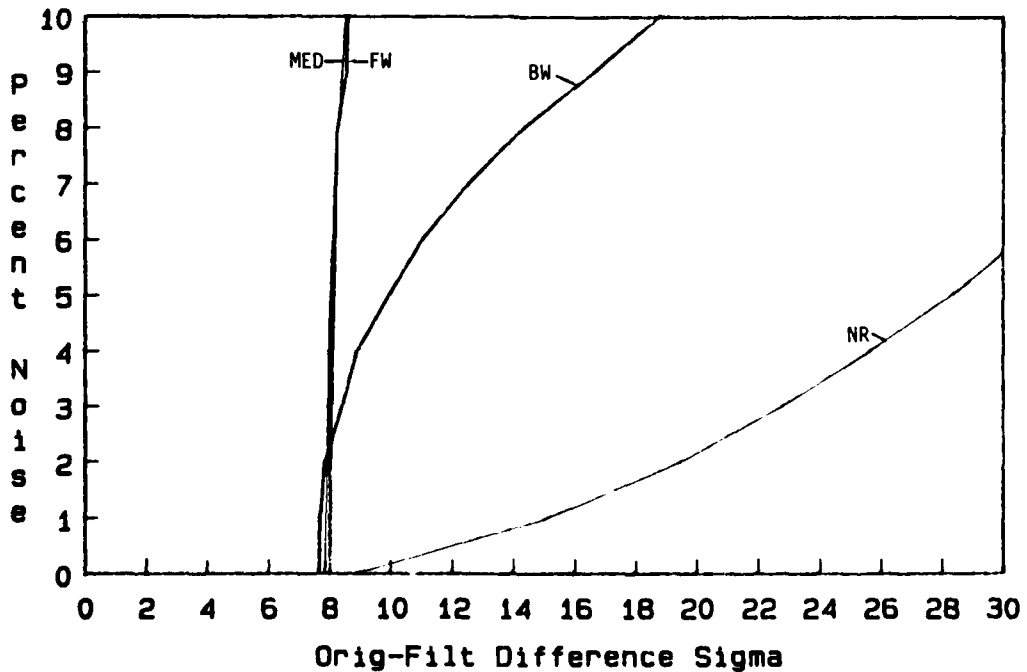
FILTER PERFORMANCE (SF6.IMG)

No Bad Pixel Check Gm=0; Gsd=5



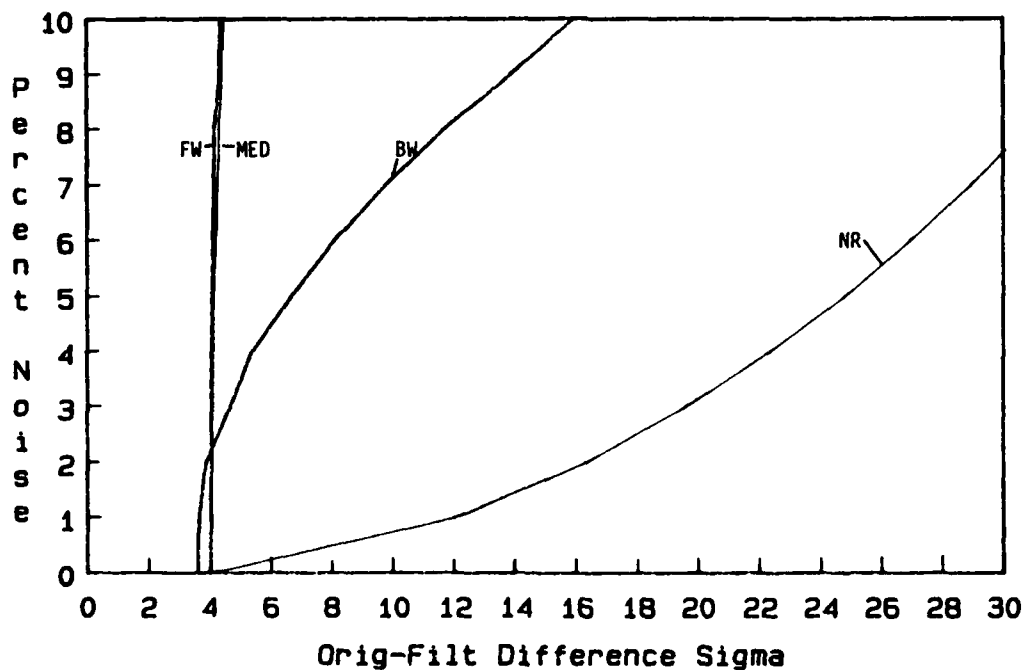
FILTER PERFORMANCE (SF8.IMG)

No Bad Pixel Check Gm=0; Gsd=5



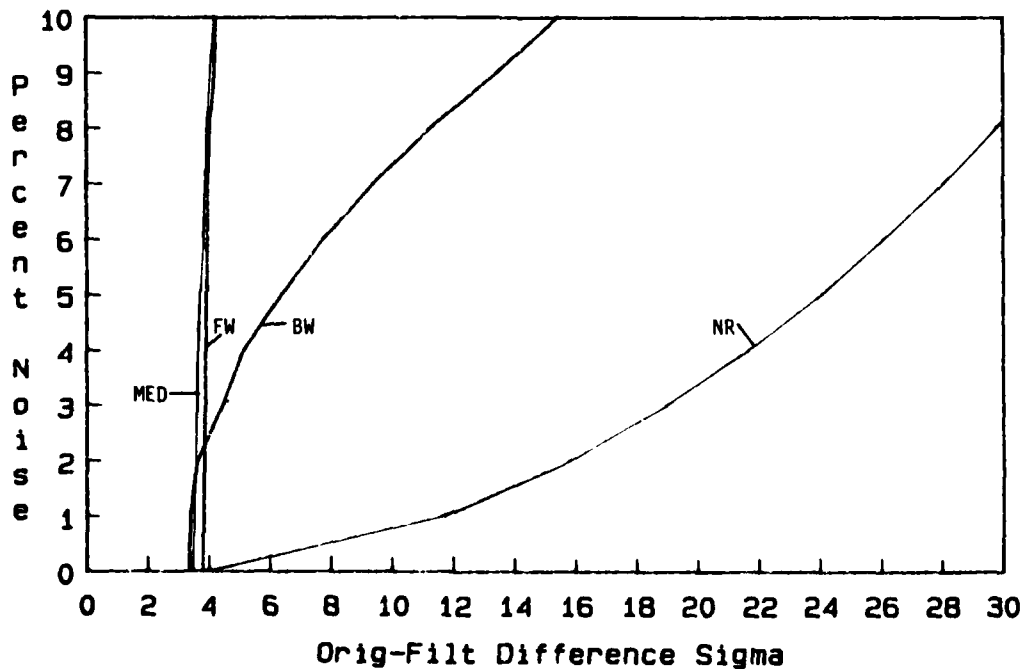
FILTER PERFORMANCE (SF11.IMG)

No Bad Pixel Check Gm=0; Gsd=5



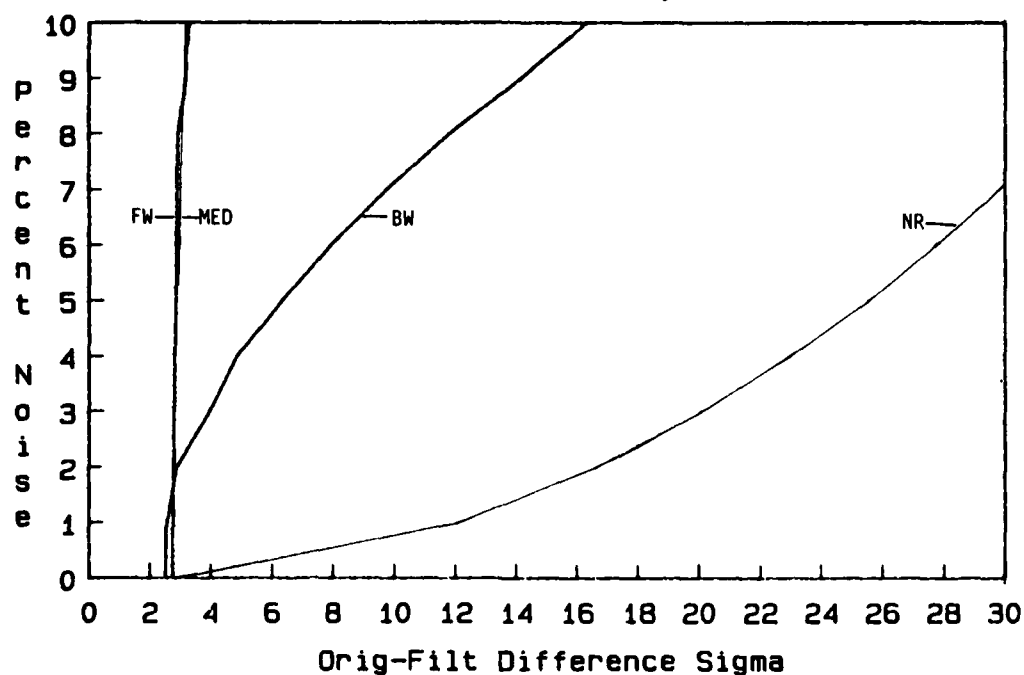
FILTER PERFORMANCE (SF1DED.IMG)

No Bad Pixel Check Gm=0; Gsd=5



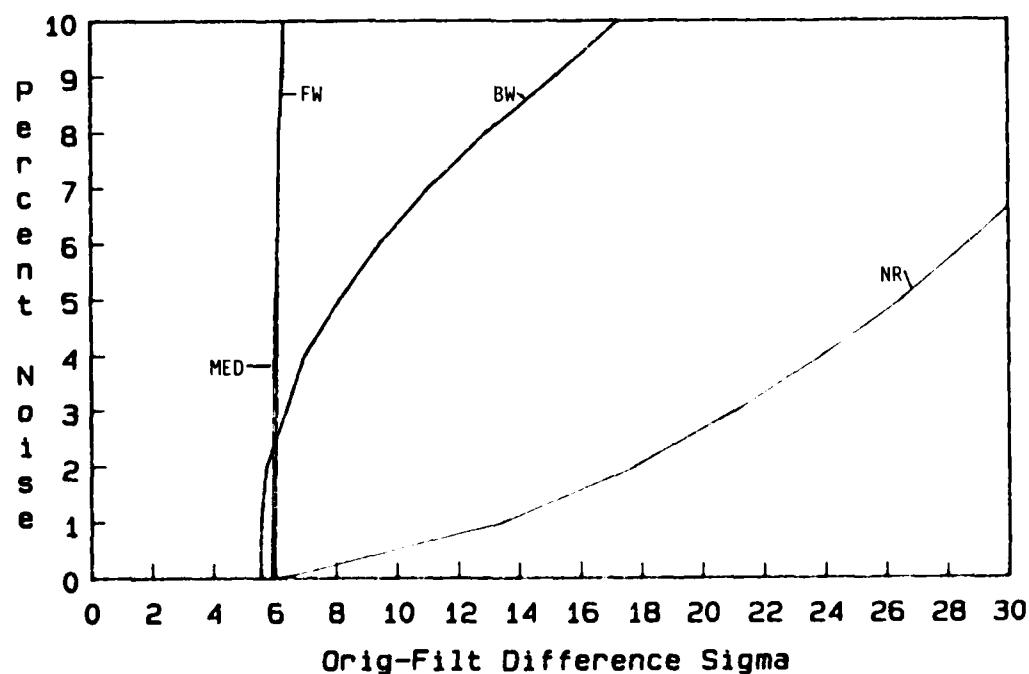
FILTER PERFORMANCE (SF2DED.IMG)

No Bad Pixel Check Gm=0; Gsd=5



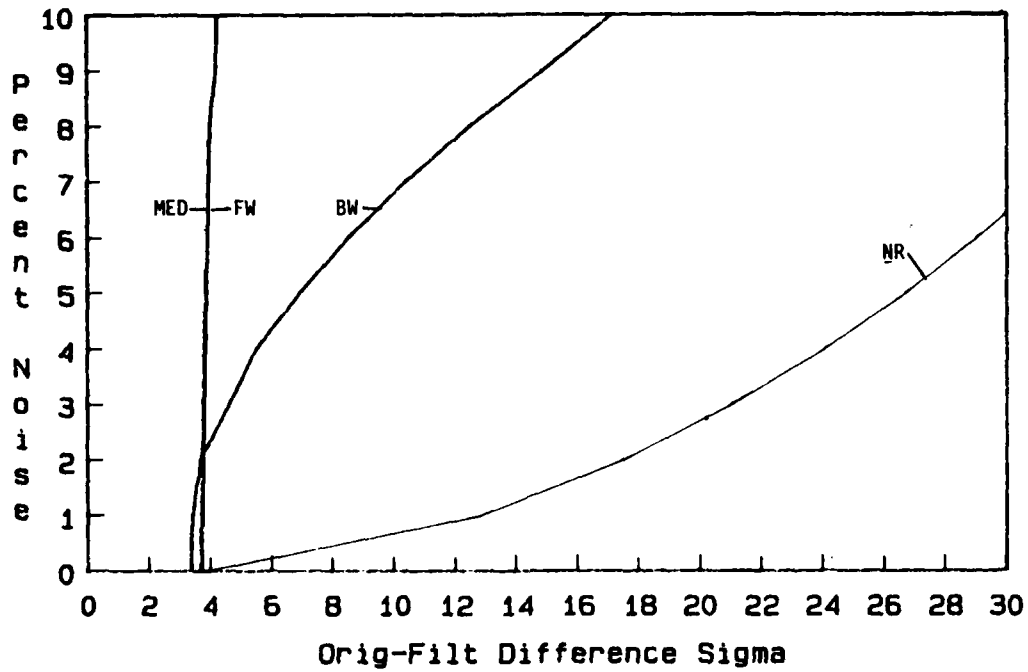
FILTER PERFORMANCE (SF3DED.IMG)

No Bad Pixel Check Gm=0; Gsd=5



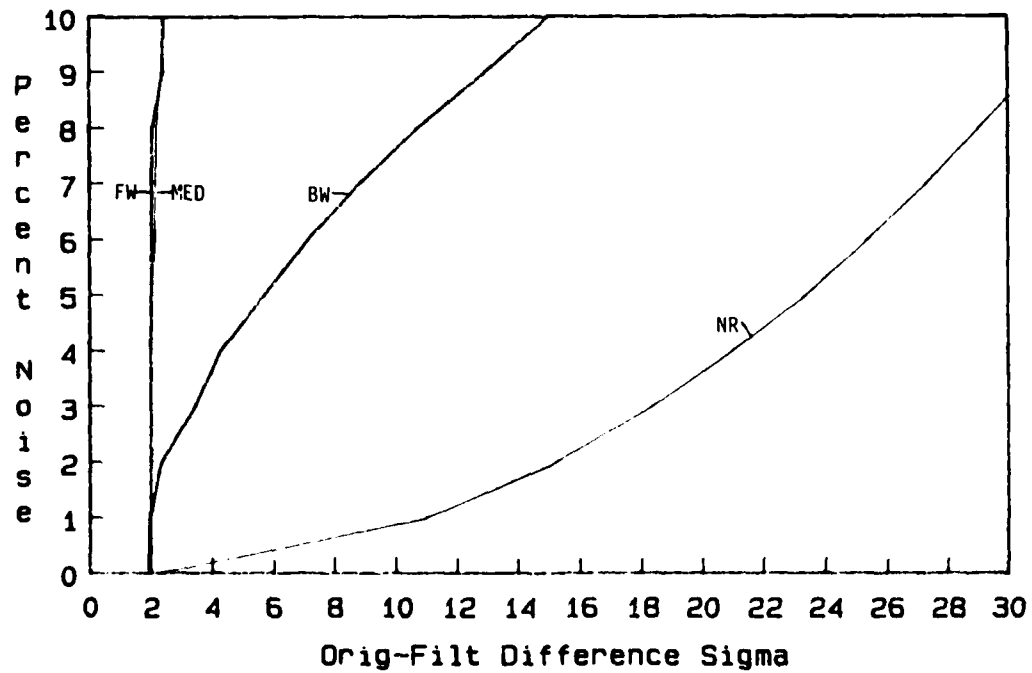
FILTER PERFORMANCE (SF6DED.IMG)

No Bad Pixel Check Gm=0; Gsd=5



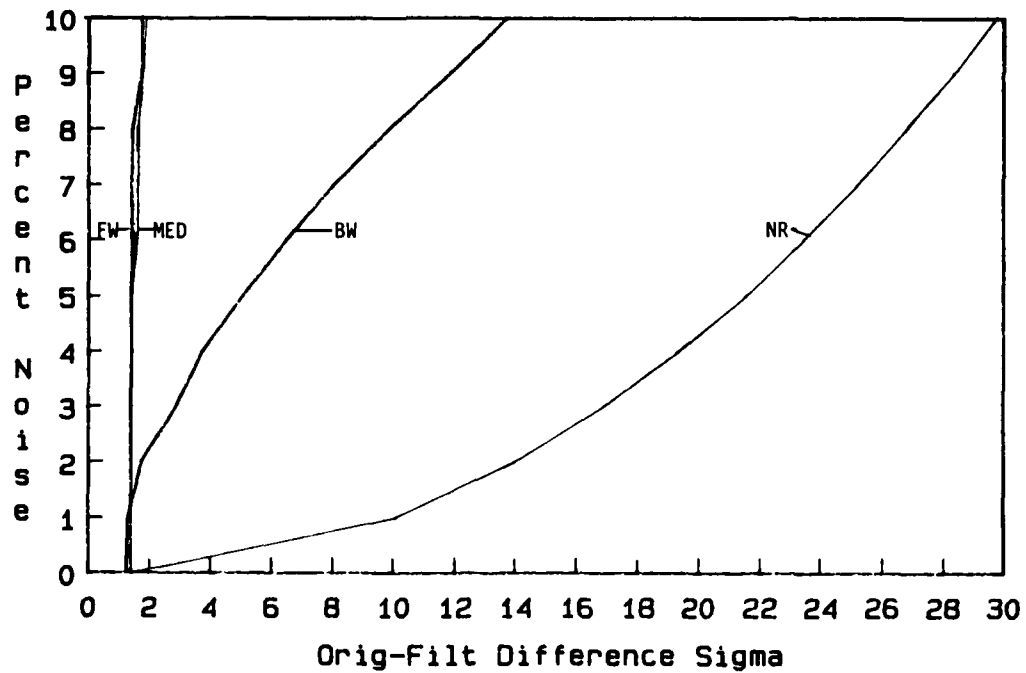
FILTER PERFORMANCE (SF8DED.IMG)

No Bad Pixel Check Gm=0; Gsd=5



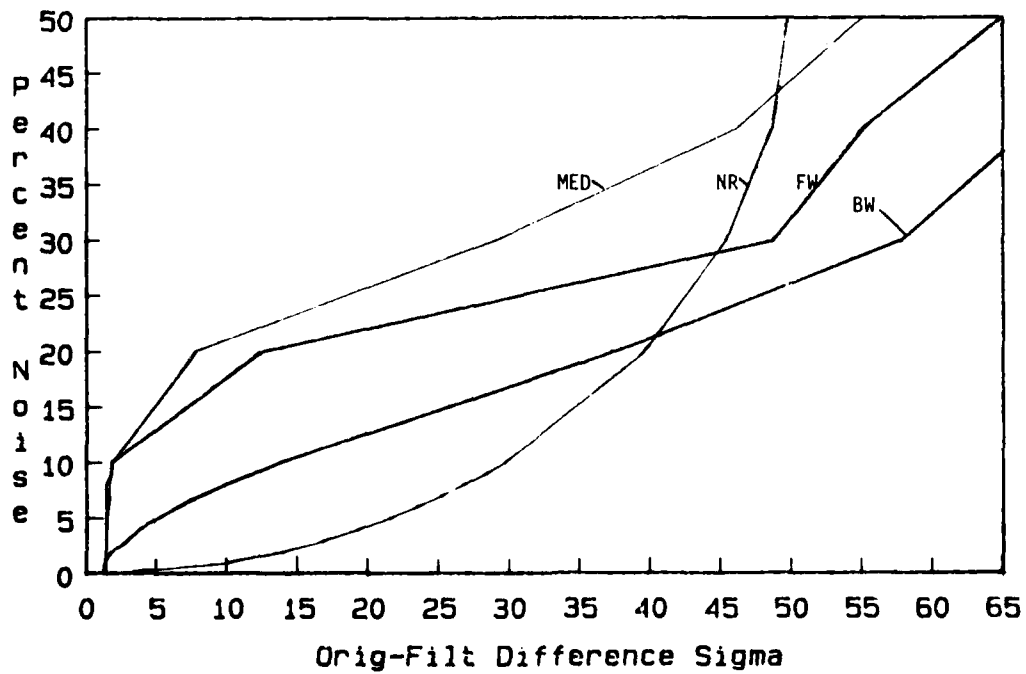
FILTER PERFORMANCE (SF11DED.IMG)

No Bad Pixel Check Gm=0; Gsd=5



FILTER PERFORMANCE (SF11DED.IMG)

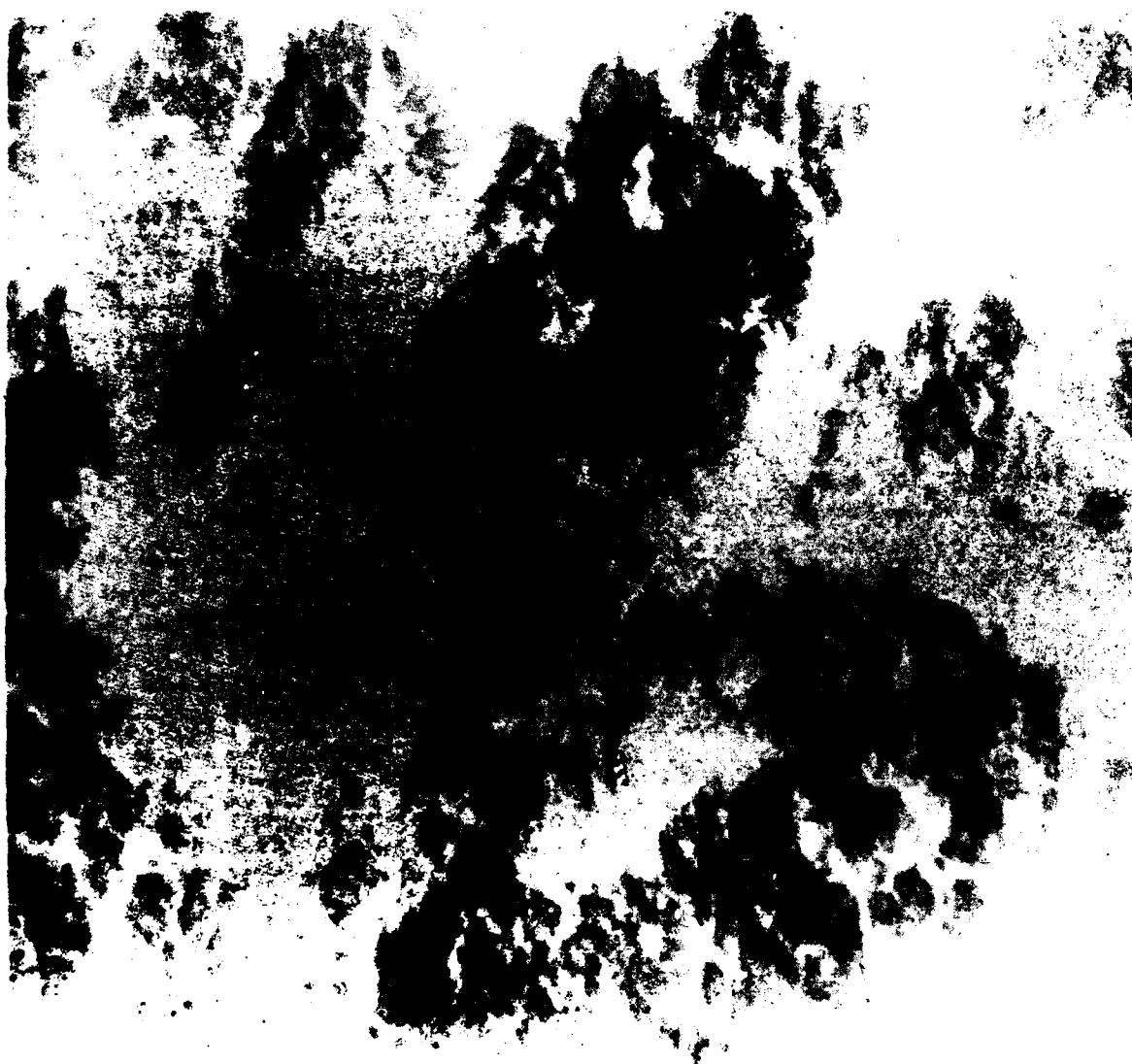
No Bad Pixel Check Gm=0; Gsd=5



APPENDIX B. ILLUSTRATIONS OF ORIGINAL IMAGES



Santa Cruz Mountains (SF6.IMG).



Ocean and clouds off Santa Cruz (SF11.IMG).



Northern San Jose (SF2.IMG).



Santa Cruz (SF8.IMG).



Downtown San Jose (SF3.IMG).



Lower San Francisco Bay (SF1.IMG).

APPENDIX C. PROGRAM LISTINGS

```

program onf1
c   this program reads daedalus images from disk then introduces
c   normal noise then applies biweight filter to remove the noise
c   then computes correlation between original and filtered images
include 'teamdef.for/nolist'
INTEGER*2 cc(262144) ! corrupted image
integer*4 seed,r(512)
real*8 dsum,dsq,corr,so,sc,pdc,ssqo,ssqc
real mod(9) ! array for sorting
byte dd(262144) ! original image
character*20 f ! file name
seed=77737
do 1 i=1,512
1   r(i)=(i-1)*512 ! build row table
i=GT_INITIAL(1,2) ! init term function *****
call GT_PROMPT('ONFI> ') ! terminal prompt
88  i=gt_word('Disk File Name(SFn(DED).IMG)=',f(1:20))
open(unit=7,file=f,form='unformatted',status='old',err=88)
read(7,err=88),dd ! GET IMAGE
type *,'IMAGE= ',f
close(unit=7)
do 7 ir=26,487 ! converting image from byte to i*2
do 7 ic=26,487
ip=dd(r(ir)+ic) ! orig byte
if(ip .lt. 0)ip=ip+256 ! if 128-255 make positive
ip=ip+300 ! shift up to get away from 0 mean gaussian noise
7   cc(r(ir)+ic)=ip ! put in contaminated image, tho not yet contam
i=gt_real('Sigmas for bad pixel criterion=',sig)
i=gt_integ('Neighborhood Size (ie 3x3=1.5x5=2)=',is)
i=gt_integ('0=Bad Pix Check; 1=Skip =',ik)
fn=(is+is+1)*(is+is+1) ! compute n for
g=204304.0 ! # of pixels in 452*452 image
ibp=0 ! # pixels contaminated
if(gt_ask('Want to introduce noise',none).eq.no)goto 67
i=gt_real('Enter Percent contaminated',p)
i=gt_real('Enter Gaussian Mean',gm)
i=gt_real('Gaussian std. dev.',sd)
do 66, ir=26,487 ! this loop enter 1-p % gaussian noise mean 0, sigma=sd
do 66, ic=26,487 ! corrupt whole picture
if(ran(seed).gt.1.0-p)then
cc(r(ir)+ic)=nint(gauss(gm,sd)) ! noise
if(ir.ge. 31 .and. ir.le. 482)then
if(ic.ge. 31 .and. ic.le. 482)ibp=ibp+1 ! # pixels corrupted
end if
end if
66  continue
67  dsum=0.0
dsq=0.0
do 8 ir=31,482
do 8 ic=31,482
ip=dd(r(ir)+ic)
if(ip .lt. 0)ip=ip+256
pd=float(ip+300)-float(cc(r(ir)+ic)) ! dif orig-contam
dsum=dsum+pd ! Sum of diffs
8   dsq=dsq+pd*pd ! sum of diff squared
dmean=dsum/g ! mean of difference of orig. and contaminated images
dsd=sqrt((dsq/g)-dmean*dmean) ! stan. dev. of difference
type *,'Diff(CONTAM.-ORIG.) Mean=',dmean
type *,'Diff(CONTAM.-ORIG.) Sigma=',dsd
icp=0 ! # pixels corrected
igp=0 ! # good pixs disturbed

```

```

nbp=0 !# bad pixs corrected
do 25,ir=31,482 !detect bad pixels and correct
  do 25,ic=31,482
    if(ik .ne. 0)goto 23 !skip bad pixel check, filter all
    x=float(cc(r(ir)+ic))
    dsq=0.0
    dsum=0.0
    do 21 iro=-is,is !fn window compute mean and sigma
      do 21 ico=-is,is
        pc=float(cc(r(ir+iro)+ic+ico)) !get pixel
        dsum=dsum+pc !sum
21      dsq=dsq+pc*pc !sum squared
        cmean=dsum/fn !compute local mean
        csd=sqrt((dsq/fn)-cmean*cmean) !local stan.dev
        if((x.ge.cmean-sig*csd).and.(x.le.cmean+sig*csd))goto 25 !good
23      icp=icp+1 !count corrected pixels subtract from contaminated
        ict=0
        if(cc(r(ir)+ic) .lt. 200)then
          nbp=nbp+1 !bad pixs corrected
        else
          igp=igp+1 !good pixs disturbed
        end if
        do 22 iro=-1,1 !extract 3x3 window
          do 22 ico=-1,1
            pc=float(cc(r(ir+iro)+ic+ico)) !get pixel
            ict=ict+1
22          med(ict)=pc !med index varies from 1-9
            call shellsort(med,9) !req for biwgt & med
            call biweight(med,x)
            cc(r(ir)+ic)=nint(x)
25      continue
        type *, '# noise pixels=',ibp,' # corrected pixels=',icp
        type *, '# noise pixels fixed=',nbp,' # good disturbed=',igp
        so=0.0 !Sum of orig
        sc=0.0 !Sum of filtered
        poc=0.0 !cross prod
        ssqo=0.0 !sum orig squared
        ssqc=0.0 !sum filt squared
        dsum=0.0 !sum of diff
        dsq=0.0 !sum dif squared
        do 18 ir=31,482
          do 18 ic=31,482
            ip=dd(r(ir)+ic) !original pixel
            if(ip .lt. 0)ip=ip+256
            po=float(ip+300)
            pc=float(cc(r(ir)+ic))
            so=so+po !sum of orig
            sc=sc+pc !sum of filtered
            poc=poc+po*pc !sum cross product
            ssqo=ssqo+po*po !sum orig squared
            ssqc=ssqc+pc*pc !sum filtered squared
            dsum=dsum+(po-pc) !sum of diffs
18          dsq=dsq+(po-pc)*(po-pc) !sum of dif squared
            dmean=dsum/g !mean of difference of original and filtered images
            dsd=sqrt((dsq/g)-dmean*dmean) !stan.dev. of difference
            corr=(g*poc-so*sc)/(sqrt((g*ssqo-so*so)*(g*ssqc-sc*sc)))
            type *, 'CORRELATION(original, filtered)=',corr
            type *, 'Diff(FILTERED-ORIG) Mean=',dmean
            type *, 'Diff(FILTERED-ORIG) Sigma=',dsd,'
            type *, 'Biweight FILTER NOISE=',p
            type *, 'NOISE Mean=',gm,' NOISE Std dev=',sd,' Neigh=',is
            call exit
          end
        include 'mathfunc for/nolist'
        include 'shellsort for/nolist'
        include 'biweight for/nolist'

```

```

program onf2
c this program reads daedalus images from disk then introduces
c normal noise then applies the median filter to remove the noise
c then computes correlation between original and filtered images
c option added to skip bad pixel check
c variable neighborhood for bad pixel check ie 3x3, 5x5, etc
include 'teamdef for/nolist'
INTEGER*2 cc(262144) 'this is corrupted image
integer*4 seed,r(512)
real*8 dsum,dsq,corr,so,sc,pcc,ssqo,ssqc
real med(9) 'array for sorting
byte dd(262144) 'original image
character*20 f 'file name
c*****
seed=77737
do 1 i=1,512
1 r(i)=(i-1)*512 'build row table
c*****
i=GT_INITIAL(1,2) 'init term function
call GT_PROMPT('ONF2> ') 'terminal prompt
c*****
88 i=gt_word('Disk File Name(SFn(DED) IMG)=',f(1,20))
open(unit=7,file=f,form='unformatted',status='old',err=88)
read(7,err=88),dd 'GET IMAGE
type *, 'IMAGE= ',f
close(unit=7)
c*****
do 7 ir=26,487 'converting image from byte to i*2
do 7 ic=26,487
ip=dd(r(ir)+ic) 'orig byte
if(ip .lt. 0)ip=ip+256 'if 128-255 make positive
ip=ip+300 'shift up to get away from 0 mean gaussian noise
7 cc(r(ir)+ic)=ip 'put in contaminated image, tho not yet contam
c*****
i=gt_real('# Sigmas for bad pixel criterion=',sig)
i=gt_integ('Neighborhood Size (ie 3x3=1, 5x5=2)=',is)
i=gt_integ('0=Bad Pix Check; 1=Skip =',ik)
fn=(is+is+1)*(is+is+1) 'compute n for bad pix check
g=204304.0 '# of pixels in 452*452 image
ibp=0 '# pixels corrupted
if(gt_ask('Want to introduce noise',none).eq no)goto 67
i=gt_real('Enter Percent contaminated',p)
i=gt_real('Gaussian mean=',gm) 'allow any gaussian mean
i=gt_real('Gaussian std. dev.',sd)
c*****
do 66, ir=26,487 !this loop enter 1-p % gaussian noise mean 0, sigma=sd
do 66, ic=26,487
if(ran(seed).gt.1.0-p)then
cc(r(ir)+ic)=nint(gauss(gm,sd)) 'noise
if(ir.ge.31 and. ir.le.482)then
if(ic.ge.31 and ic.le.482)ibp=ibp+1 '# pixels corrupted
end if
end if
66 continue
c*****
67 dsum=0.0
dsq=0.0
do 8 ir=31,482
do 8 ic=31,482
ip=dd(r(ir)+ic)
if(ip .lt. 0)ip=ip+256

```

```

        pd=float(ip+300)-float(cc(r(ir)+ic)) ! diff orig-corrupted
        dsum=dsum+pd
8      dsq=dsq+pd*pd
        dmean=dsum/g ! mean of difference of orig. and contaminated images
        dsd=sqrt((dsq/g)-dmean*dmean) ! stan. dev. of difference
        type *, 'Diff(CONTAM.-ORIG.) Mean=', dmean
        type *, 'Diff(CONTAM.-ORIG.) Sigma=', dsd
        icp=0 ! # pixels corrected
        nbp=0
        igp=0
C*****
        do 25 ir=31,482 ! detect bad pixels and correct
            do 25 ic=31,482
                if(ik.ne. 0)goto 23 ! skip bad pixel check, filter all
                x=float(cc(r(ir)+ic))
                dsq=0.0
                dsum=0.0
                do 21 iro=-is,is ! fn window compute mean and sigma
                    do 21 ico=-is,is
                        pc=float(cc(r(ir+iro)+ic+ico)) ! get pixel
                        dsum=dsum+pc ! sum
21          dsq=dsq+pc*pc ! sum squared
                cmean=dsum/fn ! compute local mean
                csd=sqrt(dsq/fn-cmean*cmean) ! local stan. dev
                if((x.ge. cmean-sig*csd).and. (x.le. cmean+sig*csd))goto 25 ! good
23          icp=icp+1 ! count corrected pixels
                if(x.lt. 100.0)then
                    nbp=nbp+1 ! found bad pixel
                else
                    igp=igp+1 ! correcting good pixel
                end if
                ict=0
                do 22 iro=-1,1 ! 3x3 window filter
                    do 22 ico=-1,1
                        pc=float(cc(r(ir+iro)+ic+ico)) ! get pixel
                        ict=ict+1
22          med(ict)=pc ! med index varies from 1-9
                call shellsort(med,9) ! req for med
                cc(r(ir)+ic)=nint(med(5)) ! median
                if(float(cc(r(ir)+ic)).eq. x)igp=igp-1 ! same pixel
25          continue
C*****
        type *, '# corrupted pixels=', ibp, ' #corrected pixels=', icp
        type *, '# corrected pixels=', nbp, ' #disturbed pixels=', igp
        so=0.0 ! Sum of orig
        sc=0.0 ! Sum of filtered
        poc=0.0 ! cross prod
        ssqo=0.0 ! sum orig squared
        ssqc=0.0 ! sum filt squared
        dsum=0.0 ! sum of diff
        dsq=0.0 ! sum dif squared
        do 18 ir=31,482
            do 18 ic=31,482
                ip=dd(r(ir)+ic) ! original pixel
                if(ip.lt. 0)ip=ip+256
                po=float(ip+300)
                pc=float(cc(r(ir)+ic))
                so=so+po ! sum of orig
                sc=sc+pc ! sum of filtered
                poc=poc+po*pc ! sum cross product
                ssqo=ssqo+po*po ! sum orig squared
                ssqc=ssqc+pc*pc ! sum filtered squared
                dsum=dsum+(po-pc) ! sum of diffs
18          dsq=dsq+(po-pc)*(po-pc) ! sum of dif squared
C*****
        dmean=dsum/g ! mean of difference of original and filtered images
        dsd=sqrt((dsq/g)-dmean*dmean) ! stan. dev. of difference
        corr=(g*poc-so*sc)/(sqrt((g*ssqo-so*so)*(g*ssqc-sc*sc)))
        type *, 'CORRELATION(original, filtered)=', corr
        type *, 'Diff(FILTERED-ORIG) Mean=', dmean
        type *, 'Diff(FILTERED-ORIG) Sigma=', dsd,
        type *, 'Median FILTER NOISE=', p
        type *, 'Gaussian NOISE Mean=', gm, ' NOISE Std Dev=', sd
        call exit
        end
        include 'mathfunc.for/nolist'
        include 'shellsort.for/nolist'

```

```

program onf3
c      this program reads daedalus images from disk then introduces
c      normal noise then applies the neighborhood replacement filter to
c      remove the noise then computes correlation between original and
c      filtered images.
      include 'teamdef.for/nolist'
      INTEGER*2 cc(262144) !this is corrupted image
      real*8 dsum, dsq, corr, so, sc, poc, ssqo, ssqc
      integer*4 seed, r(512)
      byte dd(262144) !original image
      character*20 f !file name
c*****
      seed=77737
      do 1 i=1, 512
1         r(i)=(i-1)*512 !build row table
c*****
      i=GT_INITIAL(1,2) !init term function
      call GT_PROMPT('ONF3> ') !terminal prompt
c*****
88      i=gt_word('Disk File Name(SFn(DED).IMG)=', f(1:20))
      open(unit=7, file=f, form='unformatted', status='old', err=88)
      read(7, err=88), dd !GET IMAGE
      type *, 'IMAGE= ', f
      close(unit=7)
c*****
      do 7 ir=26, 487 !converting image from byte to i*2
      do 7 ic=26, 487
          ip=dd(r(ir)+ic) !orig byte
          if(ip .lt. 0) ip=ip+256 !if 128-255 make positive
          ip=ip+300 !shift up to get away from 0 mean gaussian noise
7         cc(r(ir)+ic)=ip !put in contaminated image, tho not yet contam
c*****
      i=gt_real('# Sigmas for bad pixel criterion=', sig)
      i=gt_integ('Neighborhood Size (ie 3x3=1, 5x5=2)=', is)
      i=gt_integ('0= Bad Pix Check; 1=Skip', ik)
      fn=(is+is+1)*(is+is+1) !n for neighborhood
      g=204304.0 !# of pixels in 452*452 image
      ibp=0 !# pixels corrupted
      if(gt_ask('Want to introduce noise', none).eq no) goto 67
      i=gt_real('Enter Percent contaminated', p)
      i=gt_real('Gaussian mean=', gm) !allow any gaussian mean
      i=gt_real('Gaussian std. dev.', sd)
c*****contaminate with noise*****
      do 66, ir=26, 487 !this loop enter 1-p % gaussian noise mean 0, sigma=sd
      do 66, ic=26, 487
          if(ran(seed).gt. 1.0-p) then
              cc(r(ir)+ic)=nint(gauss(gm, sd)) !noise
              if(ir .ge. 31 .and. ir .le. 482) then
                  if(ic .ge. 31 .and. ic .le. 482) ibp=ibp+1 !# pixels corrupted
              end if
          end if
      end if
66      continue
c*****dif between orig/contaminated*****
67      dsum=0.0
      dsq=0.0
      do 8 ir=31, 482
      do 8 ic=31, 482
          ip=dd(r(ir)+ic)
          if(ip .lt. 0) ip=ip+256
          pd=float(ip+300)-float(cc(r(ir)+ic)) !diff orig-corrupted
          dsum=dsum+pd
      end do
      end do

```

```

8      dsq=dsq+pd*pd
      dmean=dsum/g !mean of difference of orig. and contaminated images
      dsd=sqrt((dsq/g)-dmean*dmean) !stan. dev. of difference
      type *, 'Diff(CONTAM.-ORIG.)Mean=', dmean
      type *, 'Diff(CONTAM.-ORIG.)Sigma=', dsd
      icp=0 !# pixels corrected
      igp=0 !# good pixels disturbed
      nbp=0 !#noise pixels corrected
*****FILTER IMAGE *****
      do 25 ir=31,482 !detect bad pixels and correct
        do 25 ic=31,482
          x=float(cc(r(ir)+ic))
          if(ik .ne. 0)goto 23 !skip bad pixel check
          dsq=0.0
          dsum=0.0
          do 21 iro=-1,1 !11x11 window compute mean and sigma for window
            do 21 ico=-1,1
              pc=float(cc(r(ir+iro)+ic+ico)) !get pixel
              dsum=dsum+pc !sum
21          dsq=dsq+pc*pc !sum squared
              cmean=dsum/fn !compute local mean
              csd=sqrt(dsq/fn-cmean*cmean) !local stan.dev
              if((x.ge.cmean-sig*csd).and.(x.le.cmean+sig*csd))goto 25 !good
23          icp=icp+1 !count corrected pixels
              if(x .lt. 100.0)then
                nbp=nbp+1 !noise corrected
              else
                igp=igp+1 !good disturbed
              end if
              dsum=0.0
              do 22 iro=-1,1 !3x3 window filter
                do 22 ico=-1,1
                  pc=float(cc(r(ir+iro)+ic+ico)) !get pixel
22          dsum=dsum+pc !summing pixels 1-9
                  cc(r(ir)+ic)=nint((dsum-x)/8.0) ! neighborhood replacement
                  if(float(cc(r(ir)+ic)) .eq. x)igp=igp-1 !same value
25          continue
*****evaluate filter*****
          type *, '# corrupted pixels=', ibp, ' #corrected pixels=', icp
          type *, '# corrected noise=', nbp, ' #disturbed pixels=', igp
          so=0.0 !Sum of orig
          sc=0.0 !Sum of filtered
          poc=0.0 !cross prod
          ssqo=0.0 !sum orig squared
          ssqc=0.0 !sum filt squared
          dsum=0.0 !sum of diff
          dsq=0.0 !sum dif squared
          do 18 ir=31,482
            do 18 ic=31,482
              ip=dd(r(ir)+ic) !original pixel
              if(ip .lt. 0)ip=ip+256
              po=float(ip+300)
              pc=float(cc(r(ir)+ic)) !filtered pixel
              so=so+po !sum of orig
              sc=sc+pc !sum of filtered
              poc=poc+po*pc !sum cross product
              ssqo=ssqo+po*po !sum orig squared
              ssqc=ssqc+pc*pc !sum filtered squared
              dsum=dsum+(po-pc) !sum of diffs
18          dsq=dsq+(po-pc)*(po-pc) !sum of dif squared
*****compute & display results of evaluation*****
          dmean=dsum/g !mean of difference of original and filtered images
          dsd=sqrt((dsq/g)-dmean*dmean) !stan.dev. of difference
          corr=(g*poc-so*sc)/(sqrt((g*ssqo-so*so)*(g*ssqc-sc*sc)))
          type *, 'CORRELATION(original,filtered)=', corr
          type *, 'Diff(FILTERED-ORIG) Mean=', dmean
          type *, 'Diff(FILTERED-ORIG) Sigma=', dsd, '
          type *, 'Neighborhood Replacement FILTER NOISE=', p
          type *, 'Gaussian NOISE Mean=', gm, ' NOISE Std Dev=', sd
          call exit
        end
      include 'mathfunc.for/nolist'

```

```

program onf4
c      this program reads daedalus images from disk then introduces
c      normal noise then applies criss-cross filter to remove the noise
c      then computes correlation between original and filtered images
include 'teamdef.for/nolist'
INTEGER*2 cc(262144) ! corrupted image
integer*4 seed,r(512),rt(4)
real*8 dsq,dsum,corr,so,sc,poc,ssqo,ssqc
byte dd(262144) ! original image
character*20 f ! file name
data rt/512,1,513,511/
C*****
seed=77737
do 1 i=1,512
1      r(i)=(i-1)*512 ! build row table
C*****
i=GT_INITIAL(1,2) ! init term function
call GT_PROMPT('ONF4> ') ! terminal prompt
C*****
88      i=gt_word('Disk File Name(SFn(DED).IMG)=',f(1:20))
open(unit=7,file=f,form='unformatted',status='old',err=88)
read(7,err=88),dd ! GET IMAGE
type *, 'IMAGE= ',f
close(unit=7)
C*****
do 7 ir=23,490 ! converting image from byte to i*2
do 7 ic=23,490
ip=dd(r(ir)+ic) ! orig byte
if(ip .lt. 0)ip=ip+256 ! if 128-255 make positive
ip=ip+300 ! shift up to get away from 0 mean gaussian noise
7      cc(r(ir)+ic)=ip ! put in contaminated image, tho not yet contam
C*****
i=gt_real('# Sigmas for bad pixel criterion=',sig)
i=gt_integ('Neighborhood Size (ie 3x3=1; 5x5=2)=' ,is)
i=gt_integ('0= Bad Pix Check; 1= Skip=' ,ik)
fn=(is+is+1)*(is+is+1) ! n for neighborhood
g=204304.0 ! # of pixels in 452*452 image
ibp=0 ! # pixels contaminated
if(gt_ask('Want to introduce noise',none).eq.no)goto 67
i=gt_real('Enter Percent contaminated',p)
i=gt_real('Enter Gaussian Mean',gm)
i=gt_real('Gaussian std. dev.',sd)
C*****CONTAMINATE WITH NOISE*****
do 66, ir=23,490 ! this loop enter 1-p % gaussian noise mean 0, sigma=sd
do 66, ic=23,490 ! corrupt whole picture
if(ran(seed).gt.1.0-p)then
cc(r(ir)+ic)=nint(gauss(gm,sd)) ! noise
if(ir.ge. 31 .and. ir.le. 482)then
if(ic.ge. 31 .and. ic.le. 482)ibp=ibp+1 ! # pixels corrupted
end if
end if
66      continue
C*****
67      dsum=0.0
dsq=0.0
do 8 ir=31,482
do 8 ic=31,482
ip=dd(r(ir)+ic)
if(ip .lt. 0)ip=ip+256
pd=float(ip+300)-float(cc(r(ir)+ic)) ! dif orig-contam
dsum=dsum+pd ! Sum of diffs

```

```

8      dsq=dsq+pd*pd !sum of diff squared
      dmean=dsum/g !mean of difference of orig. and contaminated images
      dsd=sqrt((dsq/g)-dmean*dmean) !stan. dev. of difference
      type *, 'Diff(CONTAM.-ORIG.)Mean=', dmean
      type *, 'Diff(CONTAM.-ORIG.)Sigma=', dsd
C*****FILTER*****
      icp=0 !# pixels corrected
      nbp=0 !# noise corrected
      igp=0 !# good disturbed
      ip=1
9      do 25 ir=28+ip-1,485-ip+1 !detect bad pixels and correct
          do 25 ic=28+ip-1,485-ip+1
              x=float(cc(r(ir)+ic))
              if(ik .ne. 0)goto 23
              dsq=0.0
              dsum=0.0
              do 21 iro=-is,is !fn window compute mean and sigma
                  do 21 ico=-is,is
                      pc=float(cc(r(ir+iro)+ic+ico)) !get pixel
                      dsum=dsum+pc !sum
21          dsq=dsq+pc*pc !sum squared
              cmean=dsum/fn !compute local mean
              csd=sqrt(dsq/fn-cmean*cmean) !local stan.dev
              if((x.ge. cmean-sig*csd).and.(x.le. cmean+sig*csd))goto 25 !good
23          p2=float(cc(r(ir)+ic-rt(ip)))
              p3=float(cc(r(ir)+ic+rt(ip)))
              xx=x !save
              if(p2 .le. p3)then !x p2 p3
                  if(p2 .ge. x)goto 117 !x p2 p3
                  if(x .le. p3)goto 25 !p2 x p3
116          x=p3
              else !x p3 p2
                  if(p3 .ge. x)goto 116 !x p3 p2
                  if(x .le. p2)goto 25 !p3 x p2
117          x=p2 !p3 p2 x
              end if
              icp=icp+1 !count corrected pixels subtract from contaminated
              cc(r(ir)+ic)=nint(x)
              if(xx .lt. 100.0)then
                  nbp=nbp+1
              else
                  if(x .ne. xx)igp=igp+1 !pixel disturbed
              end if
25          continue
      type *, '# corrupted pixels=', nbp, ' # corrected pixels=', icp
      type *, '# corrected pixels=', nbp, ' # disturbed pixels=', igp
      ip=ip+1
      if(ip .lt. 5)goto 9
C*****evaluate results*****
      so=0.0 !Sum of orig
      sc=0.0 !Sum of filtered
      poc=0.0 !cross prod
      ssqo=0.0 !sum orig squared
      ssqc=0.0 !sum filt squared
      dsum=0.0 !sum of diff
      dsq=0.0 !sum dif squared
      do 18 ir=31,482
          do 18 ic=31,482
              ip=dd(r(ir)+ic) !original pixel
              if(ip .lt. 0)ip=ip+256
              po=float(ip+300)
              pc=float(cc(r(ir)+ic))
              so=so+po !sum of orig
              sc=sc+pc !sum of filtered
              poc=poc+po*pc !sum cross product
              ssqo=ssqo+po*po !sum orig squared
              ssqc=ssqc+pc*pc !sum filtered squared
              dsum=dsum+(po-pc) !sum of difs
18      dsq=dsq+(po-pc)*(po-pc) !sum of dif squared
          dmean=dsum/g !mean of difference of original and filtered images
          dsd=sqrt((dsq/g)-dmean*dmean) !stan.dev. of difference
          corr=(g*poc-so*sc)/(sqrt((g*ssqo-so*so)*(g*ssqc-sc*sc)))
          type *, 'CORRELATION(original,filtered)=', corr
          type *, 'Diff(FILTERED-ORIG) Mean=', dmean
          type *, 'Diff(FILTERED-ORIG) Sigma=', dsd, '
          type *, 'CRISS-CROSS FILTER NOISE=', p
          type *, 'NOISE Mean=', gm, ' NOISE Std dev=', sd
          call exit
      end
      include 'mathfunc for/nolist'

```

```

c      biweighted 3x3 filter compute weighted mean from input array
      subroutine biweight(med,xhat)
      real*4 med(*),xhat
c      # of elements in med=9, median=med(5)
      sp=(med(7)-med(3))/1.349 !compute standard deviation from quarti
      if(sp.eq.0)sp=1.483 !cant have 0 sd
      s=sp*5.0
c      calculate initial xhat value
      s1=0.0 !sum
      s2=0.0 !n
      do 10 i=1,9 !throw out outliers > 5 std devs
        if(abs(med(i)-med(5)) .gt. s)goto 10
        s1=s1+med(i) !sum of non-outliers
        s2=s2+1.0 !n of non-outliers
10     continue
      xhat=s1/s2 !init computed mean
      s=s+sp !6.0 std dev
      s1=0.0 !sum of weighted non-outliers
      s2=0.0 !sum of weights
      do 30 i=1,9
        x=(med(i)-xhat)/s
        x=x*x
        if(x.gt.1.0) x=1.0
        wp=(1.0-x)*(1.0-x)
        s1=s1+wp*med(i)
30     s2=s2+wp
      xhat=s1/s2
      end

*****
      REAL FUNCTION GAUSS(MEAN,DEV)
*      computes a random number from a normal (gaussian) distribution with
*      the given mean and deviation
      real mean,dev,ran,sum,xx,set_gauss_seed
      integer iran,i,idum
      data iran/-1/

*
      sum=0.0
      do i=1,50
        sum=sum+ran(iran)
      enddo
      sum=sum/50.0
      xx=sqrt(12.0 * 50.0) * (sum-0.5)
      gauss=xx*dev + mean
      return

**
      entry set_gauss_seed(idum)
      iran=idum
      set_gauss_seed=0.0
      return
      end

c      shell sort
c      sort the real array elements x(1) to x(n) in
c      ascending order
c
      subroutine shellsort(x,n)
      real*4 x(*),temp
      integer*4 i,n,ndelta
      logical inordr

c
      ndelta=n
10     if(ndelta.gt.1)then
        ndelta=ndelta/2
20     inordr=.true.
        do 30 i=1,n-ndelta
          if(x(i).gt.x(i+ndelta)) then
            temp=x(i)
            x(i)=x(i+ndelta)
            x(i+ndelta)=temp
            inordr=.false.
          end if
30     continue
        if(.not.inordr) goto 20
      goto 10
      end if
      end

```

```

program onfb
c
c
c
c
include 'teamdef.for/nolist'
INTEGER*2 cc(262144),sav,ino !this is corrupted image
real med(9)
real*8 dsum,dsq
integer*4 seed,r(512)
byte dd(262144),bb(262144),s(2),ss(2) !original image,blank image
character*20 f !file name
equivalence (s,sav),(ino,ss)
c*****
seed=77737
do 1 i=1,512
1   r(i)=(i-1)*512 !build row table
c*****
do i=1,262144 !clear noise mask
  bb(i)=0
end do
i=GT_INITIAL(1,2) !init term function
call GT_PROMPT('ONFB> ') !terminal prompt
call ipinit !initiate deanza
c*****
88 i=gt_word('Disk File Name(SFn(DED).IMG)=',f(1:20))
open(unit=7,file=f,form='unformatted',status='old',err=88)
read(7,err=88),dd !GET IMAGE
close(unit=7)
call display(dd,262144,511,0,0) !display raw image
c*****
do 7 ir=26,487 !converting image from byte to i*2
  do 7 ic=26,487
    ip=dd(r(ir)+ic) !orig byte
    if(ip.lt. 0)ip=ip+256 !if 128-255 make positive
    ip=ip+300 !shift up to get away from 0 mean gaussian noise
7   cc(r(ir)+ic)=ip !put in contaminated image,tho not yet contam
c*****
i=gt_real('# Sigmas for bad pixel criterion=',sig)
i=gt_integ('Neighborhood Size (ie 3x3=1,5x5=2)=',is)
fn=(is+is+1)*(is+is+1) !n for neighborhood
g=204304.0 !# of pixels in 452*452 image
ibp=0 !# pixels corrupted
if(gt_ask('Want to introduce noise',none).eq.no)goto 67
i=gt_integ('Color of Noise=',ino)
i=gt_real('Enter Percent contaminated(10%=.1)',p)
i=gt_real('Gaussian mean=',gm) !allow any gaussian mean
i=gt_real('Gaussian std. dev.',sd)
c*****contaminate with noise*****
sav=254
do 66, ir=26,487 !this loop enter 1-p % gaussian noise mean 0, sigma sd
  do 66, ic=26,487
    if(ran(seed).gt. 1.0-p)then
      cc(r(ir)+ic)=nint(gauss(gm,sd)) !noise
      dd(r(ir)+ic)=ss(1) !put noise in picture color selected (ino)
      bb(r(ir)+ic)=s(1) !put noise in blank image (mask) (ino)
      if(ir.ge. 31 .and. ir.le. 482)then
        if(ic.ge. 31 .and. ic.le. 482)ibp=ibp+1 !# pixels corrupted
      end if
    end if
  end do
end do

```

```

66      continue
C*****dif between orig/contaminated*****
67      icp=0 !# pixels corrected
      igp=0 !# good pixels fixed
      nbp=0 !# bad pixels fixed
      call display(dd,262144,511,0.0) !display noise
      if(gt_ask('Display Blank',none).eq. YES)call display(bb,262144,511,0.0)
C*****FILTER IMAGE *****
      do 25 ir=31,482 !detect bad pixels and correct
        do 25 ic=31,482
          x=float(cc(r(ir)+ic))
          dsq=0.0
          dsum=0.0
          do 21 iro=-1,1 !1x1 window compute mean and sigma for window
            do 21 ico=-1,1
              pc=float(cc(r(ir+iro)+ic+ico)) !get pixel
              dsum=dsum+pc !sum
            21      dsq=dsq+pc*pc !sum squared
              cmean=dsum/fn !compute local mean
              csd=sqrt(dsq/fn-cmean*cmean) !local stan. dev
              if((x.ge.cmean-sig*csd).and.(x.le.cmean+sig*csd))goto 25 !good
              icp=icp+1 !count corrected pixels
              ict=0
              do 22 iro=-1,1 !extract 3x3 window
                do 22 ico=-1,1
                  pc=float(cc(r(ir+iro)+ic+ico)) !get pixel
                  ict=ict+1
                22      med(ict)=pc
                  call shellsort(med,9)
                  sav=nint(med(5))-300 !median pixel
                  if(sav.lt. 0)sav=0
                  dd(r(ir)+ic)=s(1) !corrected pixel in orig pic
                  if(cc(r(ir)+ic).lt. 200)then !bad pixel fixed
                    nbp=nbp+1 !bad pixel fixed
                    bb(r(ir)+ic)=0 !black pixel
                  else
                    igp=igp+1 !good pixel messed up
                    bb(r(ir)+ic)=s(1) !yellow pixel
                  end if
                25      continue
C*****evaluate filter*****
                call display(dd,262144,511,0.0) !display corrected pix
                type *, '# corrupted pixels=',ihp,' #corrected pixels=',icp
                type *, '# corrupted pixels fixed=',nbp,' #good made bad=',igp
                if(gt_ask('Display Blank',none).eq. YES)then
                  call display(bb,262144,511,0.0)
                end if
C*****compute & display results of evaluation*****
                type *, 'FILTER NOISE-',p
                type *, 'Gaussian NOISE Mean=',gm,' NOISE Std Dev=',sd
                if(gt_ask('Quit',none).eq. YES)call exit !gives time to display image
                end
                include 'mathfunc.for/nolist'
                include 'shellsort.for/nolist'
                include 'ipinit.for/nolist'

```

END

DTIC

8-86